

Mobile Web Mashups

The long tail of Mobile applications

CRISTOBAL VIEDMA



**KTH Information and
Communication Technology**

Master of Science Thesis
Stockholm, Sweden 2010

Mobile Web Mashups

The long tail of Mobile applications

CRISTOBAL VIEDMA



**KTH Information and
Communication Technology**

Internal report
Master of Science Thesis
Communication Systems, School of ICT
Stockholm, December 2010

Examiner: Konrad Tollmar

Communication Systems, School of ICT

TRITA Number Here!

© Cristobal Viedma, December 2010

Abstract

A Mashup is a Web page or application that combines resources or functionalities from two or more sources to create a new application or service. Combining the concept of Mashups with mobile devices can unveil a world of new Mashups, *Mobile Mashups*, satisfying the needs of niches based on the long tail theory and creating extraordinary business values.

Given the potential of Mobile Mashups it is reasonable to expect a great number of them flourishing on the Internet. However, there are a number of challenges that might slow down Mobile Mashups going mainstream.

This work sets the foundation on how to build Mobile Mashups. A reference framework has been developed in order to serve as a base for future work and help developers exploring Mobile Mashups. This reference framework categorizes Mobile Mashups by type and architecture, points out which are the best protocols and data formats to use in a mobile context, analyses different characteristics of Services providers and, finally, lists advantages and disadvantages of a Web interface for Mobile Mashups and raises a number of issues to take in consideration such as the page model and the different tools available.

Keywords: Mobile Web, Mobile Mashups, Programmable Web, Web services, User-driven development, Cloud computing

Acknowledgements

Let us start with Konrad Tollmar. Without any doubt his guidance and expertise has truly helped me and made possible this thesis. However, I will specially remember and be truly grateful to him for the sympathy he showed in the moments of this journey where I walked aimlessly and hopeless. Thank you.

I profoundly appreciate the help from those good friends who have seen early version of this report and helped me polishing the details. Roberto Castañeda and Luis Guillermo Martinez this is for you.

Joseph Pechsiri deserves a whole paragraph for himself. He is not just one of the wisest person I got to know, but has been always helping me out, not just during this work, when I needed a proper analysis of the situation and guidance. You are the man!

I am also grateful to those who have been fighting at the same time as me on their own thesis and that truly understood the desperation and inner pressure. At times, just the very fact of finishing this before them has pushed me to keep working. Carles Tomas, Victor Garcia and Nicolas Belloni, I am talking about you. ;)

Finally, I wish to thank my parents, sister and brother as well as all those supporting me from the sunny distance. Stockholm might be cold but my heart is not and still loves you.

Stockholm, December 2010

Cristobal Viedma

Contents

1	Introduction	1
1.1	Background	1
1.1.1	The mobile web revolution	1
1.1.2	A world of Mashups	2
1.1.3	The longest tail	2
1.1.4	Examples of Mobile Web Mashups	3
1.2	Problem area	4
1.3	Aim, goals and objectives	6
1.4	Target audience	6
1.5	Methods	7
1.6	Limitations	7
1.7	Structure	8
2	Mobile Web Mashups	10
2.1	The Mobile Web	10
2.2	Mobile Web Mashups	12
2.3	Why Mobile Web Mashups?	13
2.4	Examples	15
2.5	Advantages and disadvantages	17
2.6	Challenges	18
3	Reference Framework	20
3.1	Introduction	20
3.2	Types	21
3.2.1	Data Mashups	22
3.2.2	Consumer Mashups	22
3.2.3	Business Mashups	23
3.3	Service Providers	24
3.3.1	OAuth 1.0	25
3.3.2	OAuth 2.0	27

3.4	Protocols	27
3.4.1	SOAP	28
3.4.2	REST	28
3.5	Data formats	30
3.5.1	XML	31
3.5.2	JSON	31
3.6	Architectures	33
3.6.1	Server-based architecture	33
3.6.2	Client-based architecture	34
3.6.3	Mobile architecture	36
3.7	The mobile interface	37
3.7.1	Tools	38
3.7.2	The page model	38
3.7.3	Advantages and Disadvantages	39
3.7.4	Native Web Applications	39
3.8	Reference Framework	41
4	Case Studies	44
4.1	Telar	44
4.2	SoundSquare	48
4.3	Antipodes	51
4.4	MobActi	53
4.5	Deventus	55
5	Results	58
5.1	SoundSquare	58
5.2	Antipodes	62
6	Conclusions	67
6.1	Goals and objectives dissection	67
6.1.1	Architectures	67
6.1.2	Tools	68
6.1.3	User experience	69
6.1.4	Motivation	70
6.1.5	Advantages and disadvantages	70
6.1.6	Types	71
6.2	Guidelines	71
6.3	Contribution	72
6.4	Future work	72
6.4.1	Performance indicators	72
6.4.2	Mashups categorization	73

6.4.3	Native applications	73
6.4.4	OAuth aggregator	73
Glossary		75
Bibliography		76
Index		79

List of Figures

1.1	A new service is created when combining FourSquare and Sound-Cloud	3
1.2	Mashup editor Yahoo Pipes	5
1.3	Structure of the thesis	9
2.1	Mobile Growth in Traffic May 2008-2010 by Admob	10
2.2	World Wide device manufacturer share May 2010 by Admob	11
2.3	Top 10 categories of Mashups by ProgrammableWeb.com	12
2.4	The-user driven model [10]	13
2.5	The Long Tail theory by Hay Kranen	14
2.6	Revenue VS costs with Mashups	14
3.1	High-level representation of a Mashup	20
3.2	Data Mashup Skyscanner providing information from 9 different airlines	22
3.3	Consumer Mashup Housingmaps with rental information	23
3.4	Business Mashup PivotalTracker with Twitter	24
3.5	Top 10 APIs by ProgrammableWeb.com	24
3.6	OAuth protocol work-flow by LinkedIn	26
3.7	Most used protocols for APIs by ProgrammableWeb.com	28
3.8	XML overhead [22]	32
3.9	Server-based architecture	34
3.10	Client-based architecture	34
3.11	Mobile architecture	36
4.1	Telar Mobile Mashup platform [4]	45
4.2	Architecture of the Telar Mashup [4]	46
4.3	Telar example architecture	47
4.4	SoundSquare architecture	50
4.5	Antipodes architecture	52
4.6	MobActi architecture	54
4.7	Deventus architecture	56

5.1	Screen-shot of SoundSquare running on a browser	59
5.2	Screen-shot 2 of SoundSquare running on a browser	61
5.3	Screen-shot of Antipodes running on a browser	63
5.4	Screen-shot 2 of Antipodes running on a browser	66

List of Tables

2.1	Advantages and disadvantages of Mashups	17
2.2	Challenges for Mobile Web Mashups [37] [15]	19
3.1	Components of a Mashups	21
3.2	Types of Mashups [18]	22
3.3	Prices for Web Service Akismet	25
3.4	HTTP methods and CRUD actions	29
3.5	Most common data formats	30
3.6	Architectures for Mobile Web Mashups	33
3.7	Workarounds to perform Cross-Domain XMLHttpRequests	35
3.8	Reasons to use client or server based architectures	37
3.9	User interface tools	38
3.10	Advantages and disadvantages of using a Web Interface	40
3.11	Native Web applications tools	40
3.12	Reference framework for Mobile Web Mashups	43
4.1	Mashup ID of Telar	48
4.2	Mashup ID of SoundSquare	48
4.3	Mashup ID of Antipodes	51
4.4	Mashup ID of MobActi	53
4.5	Mashup ID of Deventus	55
5.1	Mashup ID of SoundSquare	58
5.2	Mashup ID of Antipodes	62

Chapter 1

Introduction

This chapter describes the thesis and introduces the reader to the subject of Mobile Mashups. First, the background of the thesis is introduced. Secondly, a definition of the problem and key research questions followed by the aim, goals and objectives are presented. Then, the methods adopted to give answers and the scope of the research are outlined. Finally, the structure explains how the chapters are constructed and related to each other.

1.1 Background

1.1.1 The mobile web revolution

Today, there are more mobile computers than there are any other forms of computing devices [15]. Mobile Internet is nowadays ramping up faster than desktop Internet did due to the convergence of five trends: 3G Internet connection, social networking, video, VoIP and high-end mobile devices [19]. Furthermore, it is expected that mobile devices will become the number one gate to access the Internet within 5 years and the number of mobile devices will be over 10 times the number of *fixed* devices accessing the Internet *ibid*. In spite of the hardware constraints that mobile devices suffer, the revolutionary Web browser developed by Apple for the iPhone and iPod Touch, Safari mobile, has been proven to be a user-friendly and satisfactory way of surfing the Internet [15].

1.1.2 A world of Mashups

Before becoming popular in the Internet, *Mashup* was a term used in the music industry. Music mashups are the mixture of two or more tracks to create a new song [11]. The technique was also called multi track recording and re-recording, where the famous group the Beatles made notable advances on the field [6]. Nowadays, in the web development, a Mashup is a web page that combines resources from two or more sources creating a new service [27]. For example, Haiku.kayac.com is a Mashup where messages in Twitter¹ tagged with the word *Haiku*² and Flickr³ pictures tagged with the same word are combined. LivePlasma⁴ on the other hand, presents products of Amazon⁵ with a different graphic interface, showing the relationship between movies, bands, actors, etc, and lets the user go directly to buy the product. More than 4900 mashups can be found in the mashups' directory ProgrammableWeb⁶ among with more than 2000 web *APIs*, the building blocks of Mashups.

From a business perspective, Mashups allow to use different services and to break down business processes into smaller pieces accelerating considerably the speed at which new business services are deployed. According to Peenikal, this will represent a substantial competitive advantage and organizations that are not pioneers in the usage of Mashups will unavoidably find themselves non-competitive [18]. Additionally, reusing existing services and data reduces the initial capital investment required for any new project. Mashups do not require extensive IT skills and can be developed in a rapid-prototyping fashion improving time-to-market and reducing application development costs. Finally, Mashups are better tailored to the end users thanks to requiring less resources and spurring innovation—widely agreed as the antidote to the current global economic slump we are in.

1.1.3 The longest tail

Mashups fit in with the long tail theory developed by Chris Anderson [1]. In the article, the business strategy of niches is presented:

The distribution and inventory costs of businesses successfully applying this strategy allow them to realize significant profit out of sell-

¹<http://www.twitter.com>

²Haiku is a form of short Japanese poetry

³<http://www.flickr.com>

⁴<http://www.liveplasma.com>

⁵<http://www.amazon.com>

⁶<http://www.programmableweb.com>

ing small volumes of hard-to-find items to many customers instead of only selling large volumes of a reduced number of popular items. The total sales of this large number of "non-hit items" is called the Long Tail [26].

This theory describes a business strategy based on selling many small volumes of hard-to-find items instead of a few large volumes of popular items. It argues that as a whole, these hard-to-find items can make up a value that equals or exceeds the few bestsellers. This is possible when the distribution channel is large enough, such as the Internet, and the marginal cost is low, such as in the Software industry. I.e. the Long Tail theory is based on *selling less of more*.

Combining the concept of Mashups with mobile devices can unveil a world of new Mashups, *Mobile Mashups*, satisfying the needs of niches based on the long tail theory and creating extraordinary business values [36].

We propose that, when a Mobile Mashup is built using Web technologies, i.e it executes within the mobile's Internet browser, it should be called *Mobile Web Mashup* in order to distinguish it from generic Mobile Mashups built with any other technology. Any application developed using Web technologies will work on all the different mobile platforms (see Figure 2.2) being the mobile browser the only common platform across all mobile operating systems [9]. As explained in the limitations section (1.6), this work will only focus on Mobile Web Mashups.

1.1.4 Examples of Mobile Web Mashups

One example of a Mobile Web Mashup could be using Foursquare. FourSquare is a service that lets users "check-in" at venues. As of now, the only motivation for people to use services such as FourSquare is to get the "mayorship" of the place, i.e. being the one that checks in the most times at the same place. This basic game mechanics is attracting a number of early adopters but it is still not proved to work for the great majority of users. An interesting value added to this service could be to mash it up with SoundCloud, an online audio distribution platform, and let users tie tracks to lo-

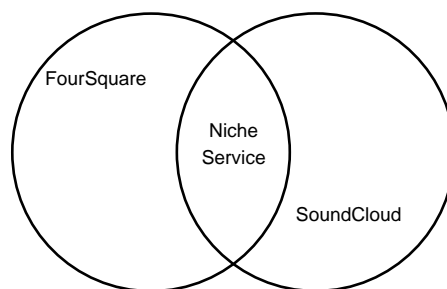


Figure 1.1: A new service is created when combining FourSquare and SoundCloud

cations. In this way, any restaurant or café can have their own play-lists created by the very customers of the venue. Combining FourSquare and SoundCloud is possible to create a new service as shown in Figure 1.1 that we will call *Sound-Square*.

Another example of a Mobile Web Mashup might be when people are feeling curious about what exactly is happening on the other side of the world. How does it look? What kind of pictures people take there? Or, at what time the sun rises? A Mobile Web Mashup that pinpoints this location on Google Maps, displays sunsetting and sunrising information from a meteorological service and shows user generated pictures taken from Flickr can help them satisfying this curiosity. The Mashup could use mobile's geographical location to get the information of the antipodes. When users' location changes, so does their antipodes and thus the information on the Mashup. Giving the nature of this Mashup, we will call it simply *Antipodes*.

1.2 Problem area

Given the potential of Mobile Mashups it is reasonable to expect a great number of them flourishing on the Internet. However, as Maximilien pointed out [15], there are a number of challenges—the unreliable connections, battery and interface constraint issues as well as some social, legal and political implications—that might slow down Mobile Mashups going mainstream.

If we have a look at the industry, several Mashup editors have already been introduced, such as Yahoo Pipes ⁷ (Figure 1.2), Intel Mash Maker ⁸ and IBM Mashup Center ⁹. These tools let the users create Mashups by dragging and dropping different services and connecting them. However this task is not as simple as it seems and the knowledge required is not trivial. Users are still required to understand the inputs and outputs of the different services and to figure out all the intermediate steps needed [7]. Moreover, these tools are not yet designed to produce Mashups for mobile devices and do not address the additional problems that they present such as the computational power restrictions and screen size limitations. A commentator of the blog company *Techcrunch*¹⁰ described this kind of tools as:

"Too complicated for casual users, not powerful enough for profes-

⁷<http://pipes.yahoo.com/>

⁸<http://mashmaker.intel.com/>

⁹<http://www.ibm.com/software/info/mashup-center/>

¹⁰<http://www.techcrunch.com>

sional users."¹¹

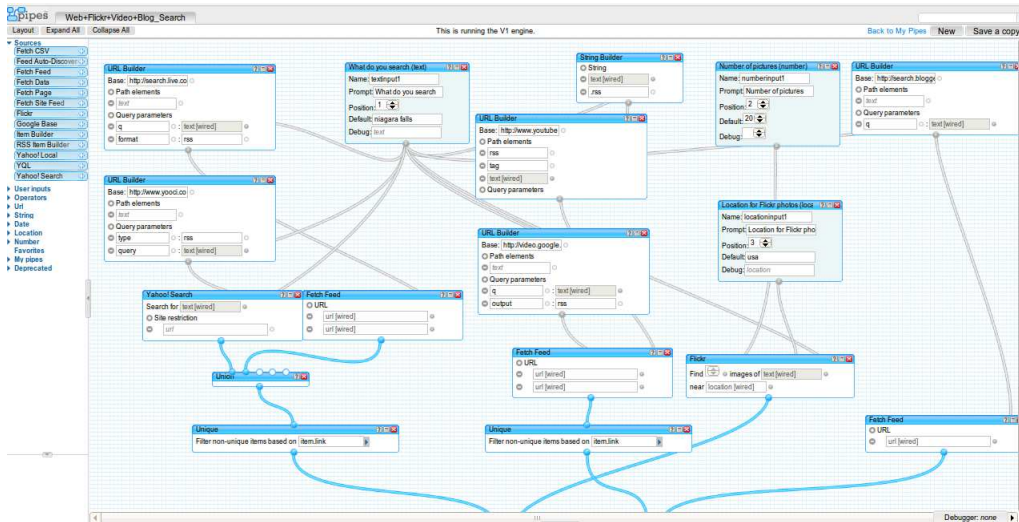


Figure 1.2: Mashup editor Yahoo Pipes

Recently, several authors have presented different architectures for Mobile Mashups [4], some based on the *Service Oriented Architecture (SOA)* design principles [37, 14], and others based on a *Peer to Peer (P2P)* approach [38]. However all these architectures are relatively high-level and specific structures need to be detailed in the future.

Other researchers tried to define theoretical frameworks to develop Mobile Mashups addressing Quality of Service (*QoS*)[36] issues or focusing on Location Based Services (*LBS*)[23]. Nonetheless these frameworks concentrate on very specific problems from a developers' point of view and do not provide general guidelines to develop Mobile Mashups with good user experience using Web technologies.

The situation gives rise to the key question that this research has studied:

- **What are the best practices to develop Mobile Web Mashups?**

And more specifically this question is divided into the following sub-questions:

- What kind of architectures can be used to develop Mobile Web Mashups?
- What are the best tools to develop Mobile Web Mashups?
- How to build Mobile Web Mashups with a good user experience?

¹¹<http://techcrunch.com/2009/07/17/microsoft-popfly-gets-squashed/>

1.3 Aim, goals and objectives

The aim of this work is to build a reference framework to understand and analyse Mobile Web Mashups. In order to fulfill this aim and to address the problems presented in the previous section, our goals are the following:

- Describe and analyse possible architectures for Mobile Web Mashups.
- Describe and analyse different tools to build mobile web interfaces.
- Describe and analyse best practices for building Mobile Web Mashups with a good user experience.

Additionally we have a number of objectives:

- Foster Mobile Web Mashups.
- Describe advantages and disadvantages of Mobile Web Mashups.
- Describe different types of Mobile Web Mashups.

1.4 Target audience

The main target audience of this thesis are persons who want to develop mobile web applications. Particularly this work has been targeted to two groups—designers and amateur developers—since normally they tend to be in need of creating quick prototypes and solutions for daily problems.

Designers with some prototyping experience that are willing to start building small mobile applications but cannot put together all the different pieces of the mobile web service puzzle will be able to understand the scope of this field.

Amateur developers that are used to work with programming tools on desktop applications but are not yet into the Internet technologies or the mobile development area will find this work as a starting point to dive into the Mobile Web Mashups world.

Finally, researchers and persons willing to have an understanding of Mobile Web Mashups current status and future directions might find this work interesting and could use it as a reference framework.

1.5 Methods

To start with, a *literature review* has been carried out to give a solid knowledge foundation and understanding of different architectures and frameworks developed by previous researchers and scientists within the field. This literature review helped to understand the current state of the art and to establish a reference framework with possible patterns and available development tools.

Subsequently a *mini-case study* has been put through. First an existing Mobile Mashup—Telar—is analysed using the reference framework developed. Then four examples of Mobile Mashups are analysed and described in order to provide guidelines for their development.

Finally we have *implemented* two examples of Mobile Web Mashups, Sound-Square and Antipodes as introduced in Subsection 1.1.4, to understand them from the developers' point of view and test the reference framework.

To sum up, the different methods that have been used during the research are:

- Literature review
- Mini-case studies
- Artifact development

1.6 Limitations

Even though there is quite an extensive literature on the field of mobile devices, *Mobile Mashup* is still in an early phase that just caught attention within the academia. This term was first cited in scientific articles in 2008 [15, 4, 14]. It is therefore hard to conduct a detailed literature review.

As mentioned in Subsection 1.1.3, this work will only focus on Mobile Web Mashups for multiple reasons: firstly, the mobile Web is the only ubiquitous platform across all mobile devices [9] and secondly in order to reduce the scope of this work since it is limited to 20 weeks. Previous authors have taken this approach as well and focus only on Mashups working on the mobile Browser [4, 5].

Additionally, given the user-driven development nature of Web Mashups [10], the number of frameworks existing today is very limited and only a few researchers have covered the topic. For this reason, it is not in our objectives to create a fully optimized framework ready for real-world deployment.

Given the dimensions of the mobile fragmentation with more than eight major operating systems [28] (see Section 2.1 for more details), it is very difficult to target all of them. Android, being an open mobile platform and, according to a Gartner report ¹², one with the biggest growth nowadays, is the chosen platform for carrying out the tests.

No performance indicators have been chosen nor has any benchmarking been carried out for two reasons: firstly, because the overall aim of this work is to look upon the entire system as a whole without digging into details and, secondly, due to time constraints. This detailed study might be interesting and could be carried out in the future as a continuation of this thesis.

Finally, according to Peenikal, there are three types of Mashups: Consumer, data and business (Section 3.2 for more details) [18]. In order to reduce the scope of the work, only the Consumer type of Mashup has been developed since it is the most popular.

1.7 Structure

Figure 1.3 depicts how this thesis is structured. Below a description of each chapter can be found.

Chapter 1, Introduction, gives an introduction to the topic and describes the problem, aim, goals, objectives, target audience and limitations of the thesis.

Chapter 2, Mobile Web Mashups, presents a deep background on Mobile Web Mashups: what are they, why should they be used, what are their advantages and disadvantages and, finally, the challenges presented.

Chapter 3, Reference Framework, presents the reference framework that has been developed in this work and analyses each one of the identified components of a Mashup.

Chapter 4, Case studies, firstly, an already existing Mobile Mashup—Telar—is analysed using the reference framework developed. Subsequently four examples of Mobile Mashups are analysed and described in order to provide guidelines for their development.

Chapter 5, Results, describes the details and results of the two Mobile Web Mashups developed—SoundSquare and Antipodes—which have been previously analysed in Chapter 4.

¹²<http://www.gartner.com/it/page.jsp?id=1421013>

Chapter 6, Conclusions, presents the conclusions and gives suggestions for future work.

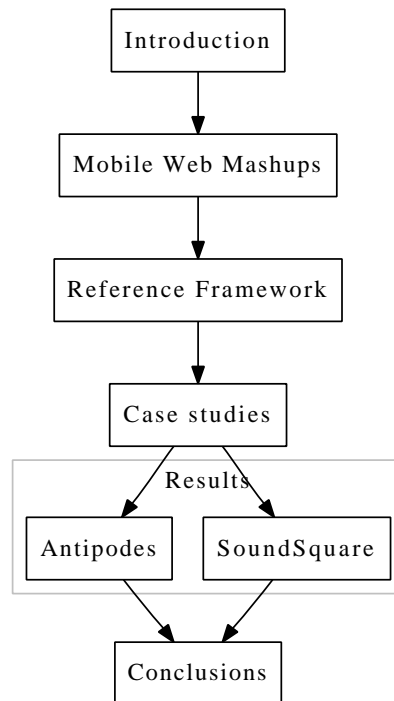


Figure 1.3: Structure of the thesis

Chapter 2

Mobile Web Mashups

This chapter presents a deeper view on Mobile Web Mashups. First the Mobile Internet is introduced followed by a description of Mobile Web Mashups, why they are relevant and a number of examples. Subsequently, advantages and disadvantages of using them and possible challenges are presented.

2.1 The Mobile Web

As Michael Maximilien pointed out, today there are more mobile computers than there are any other forms of computing devices [15]. According to *Morgan Stanley*, mobile Internet is nowadays ramping up faster than the desktop Internet did due to the convergence of five trends: 3G Internet connection, social networking, video, VoIP and high-end mobile devices. Furthermore, *Morgan Stanley* expects that mobile devices will become the number one gate to access the Internet within 5 years and the number of mobile devices will be over 10 times the number of *fixed* devices accessing it [19]. Figure 2.1 shows how the mobile traffic in the selected markets has increased over 600 percent in the last 2 years.

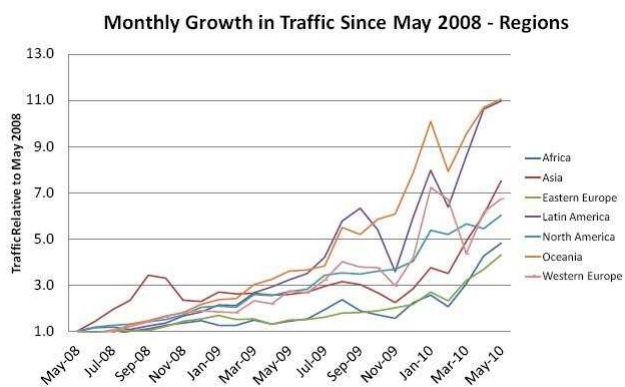


Figure 2.1: Mobile Growth in Traffic May 2008-2010 by Admob

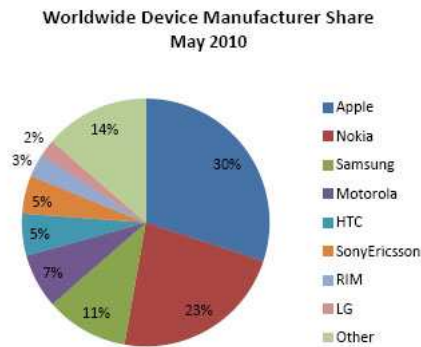


Figure 2.2: World Wide device manufacturer share May 2010 by Admob

compatible between each other. Meaning that any application develop in one platform does not work in any of the other platforms and it has to be sometimes fully re-implemented.

Taking a look at the iPhone platform, we can see that more than 100.000 native applications have been developed and are available in the App Store [3]. This has led to a situation where some people question if we are back to the desktop era of platform dependent applications [21] in spite of the number of advantages that Software as a Service (SaaS) present for mobile devices [2].

Remarkably, most of these Operating Systems include a Web browser based on the Webkit layout engine¹²³ with HTML5 and CSS3 support. Any application developed using Web Standards will work on all these platforms⁴ making the mobile browser the only common platform across all mobile operating systems [9].

In spite of the hardware constraints that mobile devices suffer, the Web browser developed by Apple for the iPhone and iPod Touch, Safari mobile, has been proven to be a user-friendly and satisfactory way of surfing the Internet [15].

According to Brian Fling, the great majority of native mobile applications⁵, excluding games, could be developed using Web standards and run on the mobile browser. This would not only increase the development pace but would require

¹Windows Mobile comes with Internet Explorer Mobile based on the Trident layout engine

²Blackberry announced the release of their new Webkit-based browser: <http://devblog.blackberry.com/2010/08/developing-new-blackberry-browser/>

³First Blackberry phone with Webkit-based browser: <http://na.blackberry.com/eng/newsroom/news/press/release.jsp?id=4238>

⁴Additionally in other platforms such as Desktop Computers

⁵He pointed out it could be as much as 70 percent of the mobile applications

However, the market is quite fragmented. If we have a look at the number of different Operating Systems available nowadays for smartphones, we find iOS for Apple's iPhone, iPod and iPad, Android from Google, WebOS from Palm (now HP), Windows Mobile from Microsoft, Symbian and Maemo from Nokia, Bada from Samsung and Blackberry OS from Research in Motion (RIM) among others (Figure 2.2). Each one of them with different Application Frameworks in-

less testing, transparent software updates and no third-party certification [9].

2.2 Mobile Web Mashups

Commonly, a Web Mashup is described as web page or application that combines resources or functionalities from two or more sources creating a new application or service [27]. For example the web Mashup *Woices*⁶ uses Google Maps to geo-localized audio-guide tours, mashing-up the mapping service with *User Generated Content* (UGC).

In the Mashups' directory *ProgrammableWeb* more than 4900 Mashups can be found among with more than 2000 web APIs. In the following chart we can see the top 10 categories of Mashups from *ProgrammableWeb* with mapping services heading the list.

The border between what is a Mashup and what is not is very thin. If we take into account for example Flickr, a photo gallery service where users can upload their pictures and store them, it has a feature that uses a map to show the pictures geo-localized. However, it is normally not considered a Mashup since its core functionality, storing pictures and visualizing them, does not require mashing up any service.

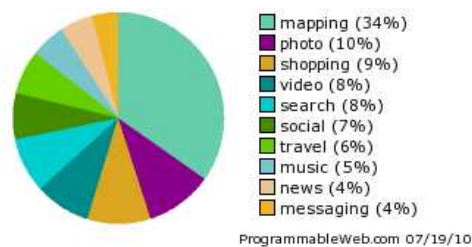


Figure 2.3: Top 10 categories of Mashups by ProgrammableWeb.com

On the other hand, *Panoramio*⁷, a service that lets the user upload their pictures and geo position them in a map, offers basically the same functionality as Flickr. However, the very core functionality of *Panoramio* is to see visually all the user's pictures on a map and it is considered a Web Mashup. For this reason we could consider that a web page or application is a Mashup when the core functionality of it, its very reason of existence, requires mashing-up services.

A Mashup implies rapid prototyping and fast integration of different APIs and data sources with a user-oriented approach [37]. The re-usage of code and services is not innovative from a technical point of view, however the power of Mashups lies

⁶<http://woices.com/>

⁷<http://www.panoramio.com/>

on enabling people to quickly create proofs of concept of creative ideas in a short time with low costs [10].

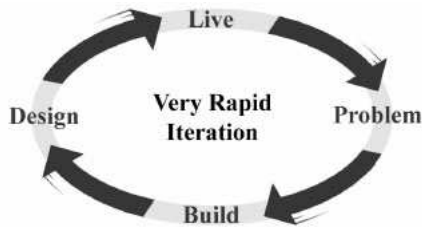


Figure 2.4: The-user driven model [10]

The development of a Mashup commonly starts when people encounter a problem in their everyday life. The problem might be with existing technologies or something that is not currently using technology and could be solved using it. Since it is fairly easy to start building a Mashup connecting different services, a quick fix is built followed by a reflection of how it works. Then the Mashup is used until

a new problem arises and the cycle starts again. This *User-driven model* is noticeable for how quickly it is possible to cycle around it with continuous improvements over time [10]. Figure 2.4 illustrates this model.

The next question that arises is, why should you build a Mobile Web Mashup?

2.3 Why Mobile Web Mashups?

In the Introduction (Section 1.1.3) was pointed out that Mashups accord with the Long Tail theory. This theory described a business strategy based on selling many small volumes of hard-to-find items instead of a few large volumes of popular items. It argues that as a whole, these hard-to-find items can make up a value that equals or exceeds the fewer bestsellers. This is possible when the distribution channel is large enough, such as the Internet, and the marginal cost is low, such as in the Software industry. I.e. the Long Tail theory is based on *selling less of more*. Figure 2.5 shows the Long Tail on the right and the most popular items on the left. It is important to notice that the areas of both regions match and, in some cases, the area of the Long Tail is bigger.

An application that helps people to locate the Nintendo Wii video game console might have a small number of users. On the other hand, the costs of developing and maintaining a database with retailers information as well as a Geographical Information System to provide the geo-localized information can be considerable. Yet, if the application is built mashing-up information from Amazon, eBay and Google Maps⁸ the development costs as well as the maintenance costs are almost

⁸<http://wii.findnearby.net/>

reduced to zero.

To manage and streamline all of your e-Commerce customer service and support operations, such as emails, eBay and Amazon.com questions, can be a tedious task. However eComm-Source⁹ allows you to easily and quickly answer customer questions. Messages and questions of emails, Amazon and eBay flow directly into Salesforce CRM (customer relationship manager), allowing retailers to visualize all their inquiry responses with the power of Salesforce CRM. This kind of innovative services spur thanks to the power of Mashups, letting people rapidly combine multiple resources into a better tailored application.

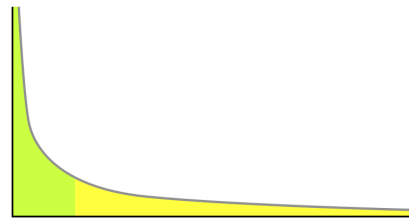


Figure 2.5: The Long Tail theory by Hay Kranen

Mashups reuse existing services and data, do not require extensive IT skills and can be developed in a rapid-prototyping fashion reducing application development and maintenance costs. All this allows the creation of a new kind of services, such as the two previous examples, that were not possible before. Figure 2.6 is a simplified representation of how Mashups allow the creation of these new services. In the figure we can see two ways to develop a new service, with or without Mashups. When using Mashups the profits (subtracting costs to revenues) of the service remain positive while without them the service becomes unviable given that the revenue of a service is independent of the underlying technology.

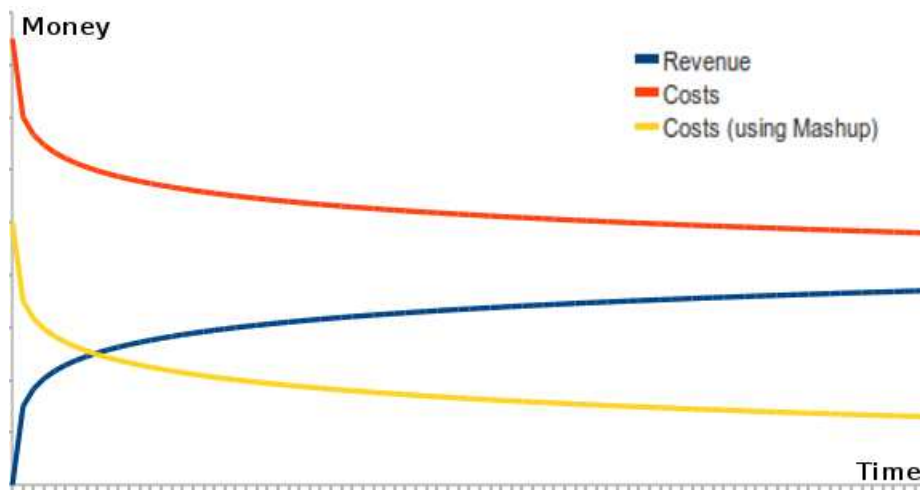


Figure 2.6: Revenue VS costs with Mashups

⁹<http://sites.force.com/appexchange/listingDetail?listingId=a0N300000019z6bEAA>

As described earlier, from a business perspective Mashups allow to use different services and to break down business processes into smaller pieces accelerating considerably the speed at which new business services are deployed. Peenikal pointed out that this will represent a substantial competitive advantage and organizations that are not pioneers in the usage of Mashups will unavoidably find themselves non-competitive [18].

In the following section a few other examples will be depicted in order to inspire the reader and show the potential of Mobile Web Mashups.

2.4 Examples

One of the biggest advantages of using Mobile devices is that they provide additional contextual information, such as the location, making it possible to develop innovative Mashups. Below, a number of examples that explore different use cases of Mobile Web Mashups are presented. As it can be seen one of the major common factors is the usage of the location information.

FourSquare is not unique in its class. There are a number of other competitors, such as BrightKite, Gowalla, Triout and Whrrl, that provide similar features. As a user I might have different friends in different services but I do not want to have to check in in each one of them manually. Moreover, most of these services have a native interface forcing me to install a new application for each one of them. It would be possible to log in in a service that mashes all of them up in a single web interface and let me check-in once, for all of them. This service is already available at <http://m.check.in>.

Another interesting Mobile Mashup could be one that allows users to compare prices among different stores to find the best price. How do you know you are getting a good price when shopping in a nearby electronic store? You could introduce the information of the product on your mobile phone and this could get prices and information from services such as Amazon or eBay and compare them for you. Even more, it could provide a map with nearby stores with bargains that would make the most out of your money.

It could also be possible that the phone uses information from your social graph in FourSquare to notify you when there is somebody you know close by. Additionally, with their birthday information from Facebook and their wish list information from Amazon, the phone could notify you of what to buy before dropping by, giving you time to bring the right present.

Have you ever been stuck a bit longer than expected in an unfamiliar airport? Using a service called *Boarding*¹⁰ you can send a Twitter message with the airport code of where you are stranded and receive a list of other people in the same situation creating serendipity encounters. This service could be expanded with coupons and discounted prices from the airport's restaurants and shops using information from *Groupon*¹¹.

For those who enjoy traveling it would be possible to create a mobile blogging platform that works off-line to avoid roaming costs (using HTML5 specific features) allowing the traveler to take notes on the go and tie them to a location using the GPS (which works off-line). When the travelers get Internet connection again this information can be automatically posted into their WordPress blog and shared among their friends on Twitter and Facebook.

Nowadays, one of the first things to do when a conference is arranged is to create an event in LinkedIn or Facebook and share it among the people. However, when it comes to the day, most of the information is still provided in paper, with the agenda of the event, the list of attendees, etc. With a Mobile Mashup that takes the information from LinkedIn and Facebook and the list of attendees from the same places, it will be possible to create an extended experience for every participant. This Mashup could contain real-time information of what the people is saying on Twitter about the event, or even let them raise questions live for the speakers using that same channel. The Mashup could also contain a map to the event using Google Maps, tagged pictures on Flickr and even generate QR-codes for drink tickets on the fly using Kaywa¹². Additionally, when the event is finished, videos of the speaks should be posted and mashed up from services such as Youtube and Vimeo.

From a social and political perspective, a Mobile Mashup that combines services such as GetSatisfaction and Google Maps could be used to raise awareness about issues in a city. Users would be able to report problems directly when they see them and list them on GetSatisfaction. Once there, the issues could be voted and sorted out by the community and the local authorities would have clear guidelines on what their citizens would like them to work on. Additionally, using the GPS location issues could be geo-localized creating a heat-map of hot-spots on the cities were more work needs to be done. Once decisions are taken or the work is done, daily status reports could be broadcasted to the community using Twitter.

On a more serious note, as Maximilien pointed out, the lack of doctors on de-

¹⁰<http://www.boarding.fr>

¹¹<http://www.groupon.com/>

¹²<http://api.qrcode.kaywa.com/>

veloping countries means that sometimes there is only one doctor available for miles and miles and they have to move from region to region on a regular basis [15]. It would be possible to combine services such as Google Health or Microsoft HealthVault to store information from all the different patients among the different locations. Patients could additionally provide regular status updates via a Web interface, e-mail or SMS that would update directly their report online. Additionally the doctor could be using external devices, such as the Wi-Fi scale Withings¹³, that connected via the phone to Internet would add extra data to the Mashup.

However the potential of Mobile Mashups, there are a number of advantages and disadvantages to take into consideration before starting to build them.

2.5 Advantages and disadvantages

Mashups have a number of advantages and disadvantages as described in [6]. Table 2.1 presents a summary of them.

Advantages	Disadvantages
Reuse of existing services and data	Service reliability and QoS
No extensive IT skills required	Integrity of content no warranted
Rapid development	Scalability issues
Cost-efficient (cheap)	Most data sources are not made as a service
Better tailored thanks to less resources required	There are no standards: difficult to implement security mechanisms

Table 2.1: Advantages and disadvantages of Mashups

Even though these issues were pointed out for Web Mashups, all of them still apply to the mobile world and it is reasonable to extrapolate them. In addition to these considerations, we have to bear in mind the new challenges presented when creating Mobile Web Mashups.

¹³<http://www.withings.com>

2.6 Challenges

As pointed out in the introduction (Section 1.2), there are a number of challenges that might slow down Mobile Mashups going mainstream. These challenges, as described by Maximilien, are frequently outside the realm of engineering and technology but rather social, legal and political [15]. He argues that this is due to the fact that Mobile Mashups impact directly human activities and social fabric.

An example that Maximilien pointed out of a social challenge is the controversial privacy problem. An application that has knowledge about your whereabouts and can not only share this information but infer habits such as where we are at certain times and with who. A legal implication might be whether this information could be used as an alibi when one is accused of a crime. Additionally, we have problems with intellectual property when people is able to broadcast directly an event, such a football match, directly with their phone to the Internet. On the political scene, mobile devices and the Internet are a global phenomenon and laws in one area might be sidestepped if the service is moved to another, therefore international cooperation will be required.

Xu et al. present more technical problems. They state that openness is one of the essences of Mashups but security has to be ensured. Additionally, they remark the superiority of the telecommunication industry when it comes to reliability and propose a more secure work-flow in order to ensure the quality of service [37].

Maximilien, on the other hand, emphasis more the technical problems on the mobile devices. He pinpoints that mobile devices' connection to the Internet is prone to high-failure rates and slow bandwidth and suggests that minimizing the communication connection might be a solution to improve the user experience and to extend the battery life. Another important problem are the constrains existing on the user interfaces meaning that usage of novel human interfaces are required. Table 2.2 presents all this challenges.

Realm	Challenges	Suggestions
Technological	Slow Internet connections	Minimize communication, Usage of novel human interfaces such as voice, orientation and gestures
	High-failure connections	
	Battery life	
	Constrained Interfaces	
	Device fragmentation	
Social, legal and political	Privacy implications (user location)	Improve legislation, countries cooperation
	Location could be used as alibi when accused of certain crime	
	Broadcasting real-time: content ownership and protection laws	
	Services moving to another countries to avoid laws	

Table 2.2: Challenges for Mobile Web Mashups [37] [15]

Chapter 3

Reference Framework

This chapter starts presenting an overview of the reference framework developed to, subsequently, explain in details each one of the components. First, a categorization of Mashups is presented. Following, we analyze different architectural patterns and dive deeper on how to develop Mobile Web Mashups describing different protocols, data formats and content providers. Finally we analyze factors when building the mobile interface and available tools.

3.1 Introduction

In the previous chapter we described a Mashup as a web page or application that combines resources or functionalities from two or more sources creating a new application or service. This description fits on the high-level representation depicted in Figure 3.1, which is an abstraction of previous architectures presented on the literature [15, 37, 4].

A number of components or characteristics can be identified in Figure 3.1. To start with, we have several *Service Providers* that provide access to their resources, in the figure represented as Resource 1 and Resource 2. These service providers are accessed using a particular set of rules

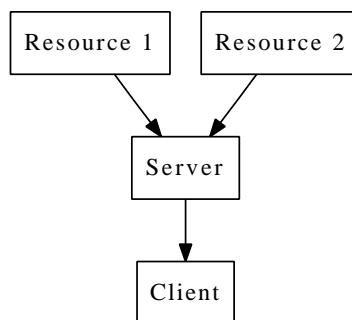


Figure 3.1: High-level representation of a Mashup

and guidelines to exchange messages called *Protocol*. A protocol is used in the communication between the Resources and the Server and between the Server and the Client. The exchanged messages contain the required information in a pre-defined *Data Format*. This data format describes how the information is stored in the message. The overall figure represents a particular *Architecture* where the resources are combined in the server and transmitted to the client. Different kinds of architectures are possible, such as combining the information in the server or directly on the client. Finally, we have the client using the Mashup via a *Web Interface*, in our particular case, in a mobile device. Additionally, as Peenikal pointed out [18], there is a particular aim for each Mashup. These aims can be categorized by *type*; having data, consumer or business Mashups. Each category will have different characteristics and target audiences. Table 3.1 lists these components.

Components
Type
Service Provider
Protocol
Data format
Architecture
Interface

Table 3.1: Components of a Mashups

In the following sections each one of these components will be detailed and a final summary will be presented in Section 3.8.

3.2 Types

Mashups can be categorized in three types: Data Mashups, Consumer Mashups and Business Mashups [18]. The most common category of Mashups is the Consumer Mashup, designed for the general public. Business Mashups are becoming popular for the reason that they allow a better usage of the resources of the enterprises. Data Mashups are normally used to combine and reformat information.

3.2.1 Data Mashups

Data Mashups combine multiple data sources of similar types (ex. RSS feeds) into a single representation. This new representation is a new and distinct Web service from the sources and provides data in a new way that was not available originally [27]. *Skyscanner.com*¹ (Figure 3.2) is an example of a Data Mashup since it combines data and provides flight tickets from multiple sources such as *British Airways*, *Air France* and *Iberia*.

Types of Mashups	
Data	Combine many sources of similar types into a single representation
Consumer	Combine many sources of different types into a visual representation
Business	Similar to consumer with the aim to solve a business problem

Table 3.2: Types of Mashups [18]



Figure 3.2: Data Mashup Skyscanner providing information from 9 different airlines

3.2.2 Consumer Mashups

Consumer Mashups are designed for the general public. Consumer Mashups normally combine multiple source of different data types into a visual representation. It has been proven to be a very effective mean for perzonalizing data according

¹<http://www.skyscanner.com>

to the customer's needs [18]. *Wikipediavision*² for example, combines real time data from the Wikipedia with Google Maps allowing users to see real time edits to Wikipedia on a world map. In *HousingMaps.com*³ (Figure 3.3) they combine information of houses to rent from Craiglist⁴ and Google Maps. The user can also filter the results by different parameters, such as rental price, and show nearby results on the map.

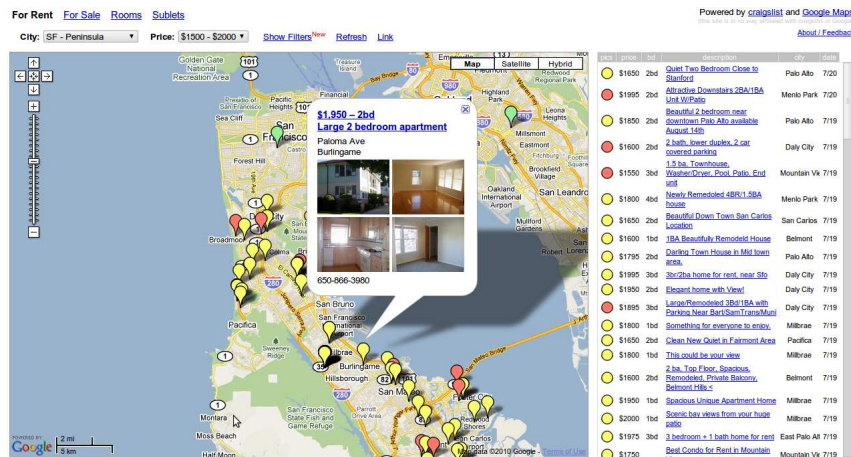


Figure 3.3: Consumer Mashup Housingmaps with rental information

3.2.3 Business Mashups

Business Mashups are similar to consumer Mashups with the aim to solve a business problem. However, they differ from Consumer Mashups in other aspects such as the security level required, a need of quality of Service (QoS), the level of sophistication, etc. Business Mashups generally combine internal information and services of an enterprise with external resources into a visually rich web application [27].

Many enterprises are embracing Mashups in order to keep up with the rapid rate at which businesses need change or when they do not have the resources or competences required to develop some services [6]. Additionally, a number of these businesses are offering their data as discrete Web services. These services are being mashed up creating new services and applications allowing collaboration between developers and businesses [18].

²<http://www.lkozma.net/wpv/>

³<http://www.housingmaps.com/>

⁴<http://www.craigslist.org>

An example of a Business Mashup could be PivotalTracker⁵ (Figure 3.4). PivotalTracker, a web-based agile project management tool, allows integration with Twitter, where project's members can see real-time updates, and GetSatisfaction⁶, where users can report bugs and request new features, among others.

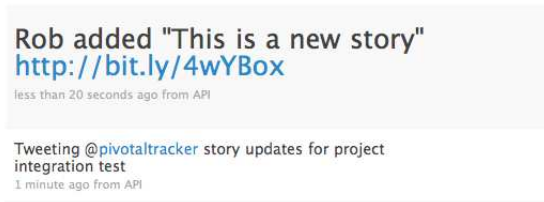


Figure 3.4: Business Mashup PivotalTracker with Twitter

Once the type of Mashup that is going to be built is clear, the next step is to decide which Web Services will be used.

3.3 Service Providers

By Service providers we understand all the third-party applications that provide some kind of service or content to mash up. The most popular and convenient way to expose their services is by providing an API to them. An API provides a set of methods to retrieve and modify information by remote calls. Figure 3.5 presents the top 10 APIs, out of 2100, according to ProgrammableWeb. It is also possible to provide content via a feed, like most of the blogs and newspapers, using for example the RSS or ATOM format. Information can also be gathered using Web scraping on any website, making all of them potential content providers.

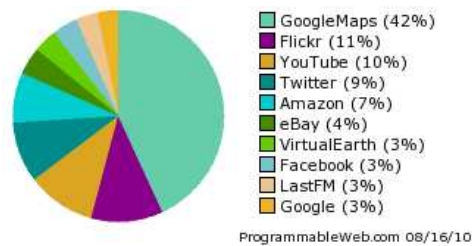


Figure 3.5: Top 10 APIs by ProgrammableWeb.com

Most Web Services are free to use but they do not warrant the quality of service. There are several of them that need to be paid on a number of call basis such as *Akismet*⁷, a service that helps bloggers detect SPAM comments. This means that the first amount of calls to their service are free, and subsequently a monthly fee

⁵<http://www.pivotaltracker.com>

⁶<http://www.getsatisfaction.com>

⁷<http://akismet.com/>

has to be paid for a given amount of calls. Table 3.3 shows an example of different prices for Akismet.

The *Terms of Service*(ToS) are the rules that must be agreed in order to use a service. These rules can define limitations on the number of calls, geographical restrictions, type of device, etc. These Terms of Service must be read carefully and agreed, especially if the application is going to be used in a business environment. For example, Amazon Product Advertising API does not allow the usage of their API on mobile devices as we can see in the following paragraph from their Terms of Service⁸:

4(e): You will not, without our express prior written approval, use any Product Advertising Content on or in connection with any site or application designed or intended for use with a mobile phone or other handheld device.

Number of API calls	Monthly price
50,000	\$50
100,000	\$100
200,000	\$200
500,000	\$500
1,000,000	\$1,000

Table 3.3: Prices for Web Service Akismet

Another important aspect of the Service Providers is if the content they provide is private or public. By private content we understand anything private to a user, such as pictures, personal information, private emails, etc. Whenever there is user's private data the Mobile Mashup puzzle becomes harder to solve since it is required to deal with some kind of authentication. One way to deal with authentication is to provide a basic HTTP Authentication⁹ from the Web Service. However this method is not recommendable since the username and password of the user are directly given to the Mashup to let it access the private data. A newer approach which is gaining a lot of supporters, among them Facebook¹⁰, is the OAuth protocol¹¹ which does not share private login details with the Mashups.

3.3.1 OAuth 1.0

OAuth is a protocol that allows users to share private information of one service with another service. It works providing tokens instead of passwords. Each token grants access to a specific service (e.g. LinkedIn) for a specific resource (e.g. just

⁸<http://blog.programmableweb.com/2010/07/28/making-mobile-apps-not-with-these-apis/>

⁹http://en.wikipedia.org/wiki/Basic_access_authentication

¹⁰<http://developers.facebook.com/docs/authentication/>

¹¹<http://oauth.net/>

contacts) and for a defined duration (e.g. for two weeks). I.e. it grants access to a limited part of the private data without exposing the password of the user. The idea is well depicted in the following metaphor:

Many luxury cars come with a valet key. It is a special key you give the parking attendant and unlike your regular key, will only allow the car to be driven a short distance while blocking access to the trunk and the on-board cell phone. Regardless of the restrictions the valet key imposes, the idea is very clever. You give someone limited access to your car with a special key, while using another key to unlock everything else.¹²

Figure 3.6 shows the work-flow of the OAuth protocol. In step A the Mashup asks for a *Request Token* to the service provider (e.g. LinkedIn). In step B the Request Token is returned and the user is redirected to the service provider (step C) where it is asked for permission to access private data. When the user has granted the access (step D) the Mashup requests an *Access Token* (step E). When this Access Token is provided (step F), the Mashup can start querying for the private data it has been given access to (step G).

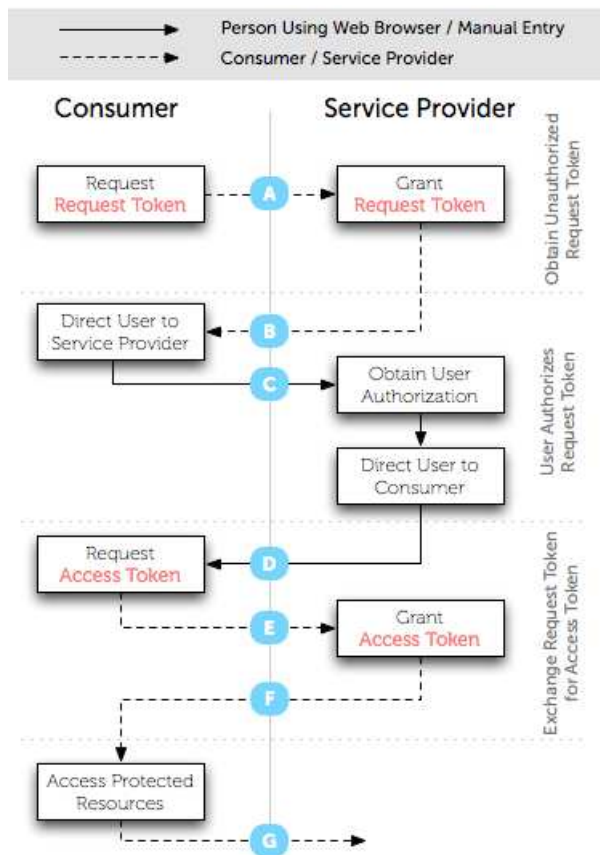


Figure 3.6: OAuth protocol work-flow by LinkedIn

¹²<http://hueniverse.com/oauth/guide/intro/>

consider this on the context of a mobile device, it can be fairly difficult. Some companies¹³ have realized this problem and started providing alternative protocols¹⁴ with the tradeoff of exposing the user's password to the Mashup. Additionally, it can become quite cumbersome if the Mashup involves more than one service provider and the user has to be redirected to each one of the services to grant access with all the redirections that this involves. For these and other reasons a new version of the protocol is being developed. A suggestion for future work is presented in 6.4.4 based on this problem.

3.3.2 OAuth 2.0

The next version of the OAuth protocol is called OAuth 2.0 and is not backward compatible with the previous implementation. OAuth 2.0 provides 6 different workflows as opposed to a unique work-flow available previously (figure 3.6). These 6 work-flows are meant to be used in different contexts such as in web applications, desktop applications or mobile devices. The primary aim of the new protocol is to ease the consumer's implementation.

The specification is being developed¹⁵ within the IETF OAuth Working Group and is expected to be finalized by the end of 2010¹⁶.

Nowadays the biggest implementation of the OAuth 2.0 protocol is presented in the new Facebook's Graph API¹⁷ which provides access to private data of the largest Social Network in the world with over 500 million users.

In order to retrieve the information the Mashup needs to communicate with these service providers. This communication has to follow a set of specified rules called *protocol*.

3.4 Protocols

Mashups are composed of Web Services and they communicate between each other using a *protocol*. A protocol is a set of rules and guidelines to exchange messages in a defined format such as XML. The protocol defines *how* they communicate.

¹³<http://groups.google.com/group/foursquare-api/web/oauth>

¹⁴<http://dev.twitter.com/pages/xauth>

¹⁵<http://tools.ietf.org/html/draft-ietf-oauth-v2-10>

¹⁶<http://hueniverse.com/2010/05/introducing-oauth-2-0/>

¹⁷<http://developers.facebook.com/docs/authentication/>

Some commonly used protocols for Web Services are XML-RCP, JSON-RCP, SOAP and REST among others¹⁸. Figure 3.7 presents the most used protocols among the APIs available in ProgrammableWeb. Remarkably, SOAP and REST are the most common protocols with a 89% of the Web Services understanding them. Notice that the third most common protocol, Javascript with a 5%, is not really a protocol. Javascript as a protocol means that the Web Service provides a Javascript library to be imported from the client's browser to facilitate the interaction avoiding the browser's sandbox (see subsection 3.6.2 and the problem with the Same Origin Policy). SOAP and REST will be the focus of this section.

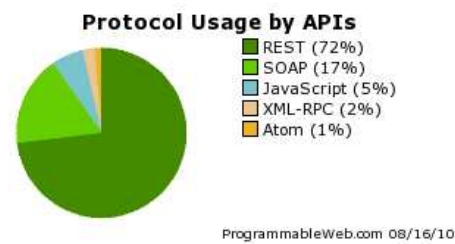


Figure 3.7: Most used protocols for APIs by ProgrammableWeb.com

3.4.1 SOAP

SOAP stands for Simple Object Access Protocol. It is a protocol for exchanging XML messages in Web Services. It can rely on different other protocols for the message transmission such as SMTP or other but most commonly HyperText Transfer Protocol (HTTP) . HTTP is supported by all Internet browsers and servers. SOAP consists of four elements: An *envelope* to identify the XML document as a SOAP message, a *header* that contains application-specific information about the message, a *body* with the payload information such as calls and responses of procedures and a *fault* container with errors and status information.

W3C defines a standard for Web Services, commonly called WS-* services, using SOAP and XML. WS-* services have become an industry standard for connecting remote computers [12].

3.4.2 REST

REST (Representational State Transfer) is not really a protocol but an architecture for distributed hyper-media systems such as the World Wide Web. REST was initially described using the HTTP protocol, but is not limited to it.

¹⁸See http://en.wikipedia.org/wiki/List_of_web_service_protocols

REST is build around the concept of *resources* and their states. In contrast, WS-* Services focus in methods' calls. A resource can be any meaningful concept that needs to be addressed. Resources are used using a set of simple, well-defined operations. The four basic ones, Create, Read, Update, Delete, are commonly referred as *CRUD*.

REST is fundamentally a client-server architecture. Each time a client communicates with a server, *representations* of resources are transferred. A representation of a resource indicates the current or intended state of the resource. Every resource is identify by an URL and each state has a different URL. REST is designed to use a stateless protocol such as HTTP [31].

RESTful Web services are services built upon the REST architecture and use HTTP as application protocol. Table 3.4 shows how the HTTP methods are normally mapped into the CRUD actions.

HTTP method	CRUD Action
GET	Retrieve resource
POST	Create resource
PUT	Update resource
DELETE	Delete resource

Table 3.4: HTTP methods and CRUD actions

A study was conducted showing that RESTful web services perform better than SOAP

web services on mobile devices [12]. Some advantages that REST showed over SOAP include smaller message size (about 4-5 times smaller) and better response time (about 2-3 times faster). As the authors point out, smaller message size and response time means less processing and transmission time and, consequently, lower power consumption and faster service. Notice that this study was conducted using a Java implementation on the client. On a mobile browser it would be necessary to build all the SOAP messages using Javascript¹⁹ which could slow down considerably more the response time. On the other hand, REST does not require any extra library since it uses directly HTTP methods .

As important as the chosen protocol is the data format in which the information is transmitted. The next section describes different formats and gives suggestions on which ones are better in the mobile context.

¹⁹For an existing library see: <http://www.codeproject.com/KB/ajax/JavaScriptSOAPClient.aspx>

3.5 Data formats

A Mashup communicates with a Web Service sending and receiving messages in a specified data format. A data format is the structure in which these messages are built. Without knowing the data format, a Mashup cannot know exactly how the information is structured. There are a number of different data formats in which Web Services provide their content. Table 3.5 describes the most common formats.

Format	Description
TXT	Text. The document contains the requested data in plain text without explicit format
CSV	Comma-Separated Value. A simple text format to store information in the form of tables. Each line of the file is a row in the table, and the values of the line are separated by commas
XML	eXtensible Mark-up Language. A generic format for encoding information in a structured way. Produced by the W3C, it is one of the most widespread formats
RSS	Really Simple Syndication. Based on XML, it is a web feed format for publishing frequently updated content such as Blogs
GeoRSS	is an emerging format to include location on web feeds' content. Derived from RSS
ATOM	Atom Syndication Format. It is an XML language used for web feeds. It was developed as a community-driven alternative to RSS aiming to improve its deficiencies
KML	Keyhole Markup Language. It is a format based on XML to represent geographical information
HTML	HyperText Markup Language. It is the mark-up language in which the web pages are written
JSON	Javascript Object Notation. It is a text-based format derived from the nomenclature of the Javascript's arrays
YAML	a recursive acronym for "YAML Ain't Markup Language". Its syntax was designed to be easily mapped to data types common to most high-level languages. Additionally, it is highly human-readable due to its indented outline

Table 3.5: Most common data formats

The most common formats are XML and JSON with 1473 and 661 (respectively) APIs providing content in such a format according to ProgrammableWeb²⁰. Therefore, XML and JSON will be the focus of this section.

3.5.1 XML

As shown in Table 3.5, XML stands for *eXtensible Mark-up Language*. XML was produced as a generic format for encoding information in a structured way. In contrast with HTML, which was designed to display data, XML was designed to transport and store data. XML tags are not predefined allowing the developer to define its own tags according to his needs. Nowadays it is one of the most widespread formats and is a W3C recommendation. Below an XML example is shown:

```
<?xml version="1.0" encoding="UTF-8"?>
<email>
  <to>Konrad</to>
  <cc>Natalia</cc>
  <cc>Maria</cc>
  <from>Cristobal</from>
  <subject>An XML-JSON example</subject>
  <body>Hope this example is witty enough!</body>
</email>
```

3.5.2 JSON

JSON stands for Javascript Object Notation. It is a lightweight text-based format derived from the nomenclature of the Javascript's arrays. However, it is language-independent, i.e. it is possible to use it with any programming language provided it has a JSON parser²¹. It serves as an alternative to XML and its primary use is to transmit data between a server and an application. It gained popularity with AJAX since JSON is a subset of Javascript and therefore it is possible to parse JSON text into a Javascript object just invoking the *eval* function albeit, this has some security problems. Below the JSON equivalent to the XML example in subsection 3.5.1 is shown:

```
{"email" : {
  "to" : "Konrad",
```

²⁰Data extracted in August 2010

²¹See <http://www.json.org>

```

"cc" : ["Natalia", "Maria"],
"from" : "Cristobal",
"subject" : "An XML-JSON example",
"body" : "Hope this example is witty enough!"
}}

```

Even though XML is one of the most widespread formats, it is not usually the best way to exchange data in the mobile context.

A study has shown that the overhead presented in XML messages can be considerably high compared with the actual size of the content (Figure 3.8) [22]. XML messages are more verbose mainly due to producing content which is human readable, even though most frequently is going to be consumed by a machine. Other than the message size, it has

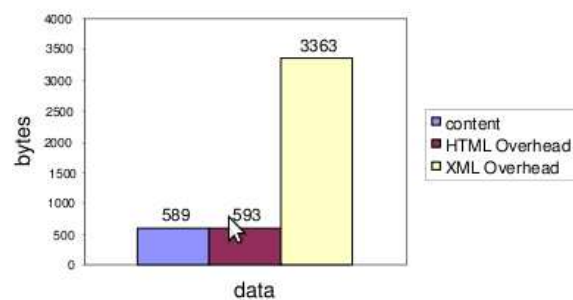


Figure 3.8: XML overhead [22]

been also pointed out that the speed of a JSON parser is almost 10 times the speed of an XML parser [13]. Smaller message sizes and faster parsers mean less processing and transmission time and, consequently, lower power consumption and faster service.

Recently desktop web browsers, such as Firefox 3.5+, Internet Explorer 8, Opera 10.5+ and Webkit-based browsers (e.g. Google Chrome, Apple Safari), have or are working on native JSON encoding/decoding. This makes it faster because it does not parse using Javascript functions and remove the security concerns of using the *eval* function. Even though, as far as the writer knows, there is not yet native JSON on mobile devices, it is likely that it will soon follow suit.

Finally, JSON allows using the technique JSONP (see Table 3.7) which avoids the Same Origin Policy (see Section 3.6.2) and allows cross-domain requests from within the browser without a server-side proxy. This allows pure client-based architectures.

Once the service providers, protocols and data formats are studied and selected, the next step is deciding which kind of architecture is the most appropriate one.

3.6 Architectures

As Sunilkumar Peenikal pointed out, from an architectonic point of view there are two styles of Web Mashups: *Client-based* and *Server-based* [18]. Additionally, we can have a *Mobile* architecture where both styles are combined leveraging on the advantages of each one. This Mobile architecture has been previously seen in works such as [4] but has not been properly described. Table 3.6 presents these three architectures.

Architectures	
Server-based	Information gathered and processed in the server
Client-based	Executed within the client's browser
Mobile	A combination of Client and Server-based architectures

Table 3.6: Architectures for Mobile Web Mashups

Client-based Mashups are executed within the client's browser

which gathers the information directly and presents it. On the other hand Server-based mashups gather, analyze, combine and reformat the information on the server side and transmit it to the client. A Mobile architecture is when some information is processed on the server side and is then combined with more information on the client side. An example of a Mobile architecture could be any of the many Google Maps Mashups available on the Internet where the data is gathered on the server side from a third party service, like Craigslist, and presented on the client side on top of a map.

3.6.1 Server-based architecture

As the name indicates, Server-based Mashups integrate services and data on the server side. The server mediates all the messages between the client and the Web Services acting as a proxy. The server is therefore in charge of gathering, transforming and combining the data from third-party services before sending it to the client. Additionally, when the client is pushing data to other services, the information goes through the server [16]. Figure 3.9 shows a diagram of this architecture.

A Server-based architecture complies with the *Facade pattern*. A facade pattern is a design pattern that provides a simplified and unique interface to multiple Web Services [18].

As pointed out in [16] there are multiple reasons to use this kind of architecture. Firstly, the number of libraries and available protocols is considerable extense and it can simplify many necessary tasks. The server can cache information and act as a buffer for slow or faulty services. It can also pre-process all the information reducing the data size, changing the format or combining it with other data. It is easier to handle security requirements such as those required by *OAuth* (see Section 3.3.1). Finally, it is possible to make concurrent and asynchronous calls to multiple Web Services, something which is not always possible from the Web Browser since it is commonly limited to only two or three connections. Additionally we have to consider the limited computational power available on most mobile devices and the possibility to transfer all this load to the server side.

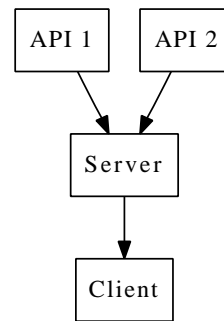


Figure 3.9: Server-based architecture

3.6.2 Client-based architecture

In contrast with a Server-based architecture, in a Client-based architecture the integration of services and content is done in the client side. In the particular case of Mobile Web Mashups this happens in the web browser. Therefore, the client's browser is in charge of gathering the content, transforming it to the right format and combining it before showing it on the screen [17]. Figure 3.10 illustrates this kind of architecture.

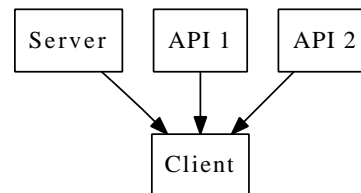


Figure 3.10: Client-based architecture

There are also multiple reasons to use Client-based architecture. To start with, it can be easier to implement than Server-based architectures. Nowadays more Web Services are providing their APIs as a Javascript library (ex. Google Maps²²) that is ready to use in the Web application's code. Therefore, there is no need of a server-side component. The moment the Web is loaded on the client the server becomes secondary since all the calls are done directly between the client and the different Web Services avoiding the bottleneck presented in a Server-based architecture. Finally, the response does not have to route via the server, saving one

²²<http://code.google.com/apis/maps/documentation/javascript/>

network hop, which normally increases the performance and reduces the server load [17].

Interestingly, even though in occasions the data does not need to be processed on the server side and could be simpler to access it directly from the client, some services need to be accessed via the server using it as a proxy. This is due to a security concept, the *Same Origin Policy*, that does not allow to access third-party servers from within the client browser.

The Same Origin Policy²³ permits browser-side scripts, such as Javascript, to access without restrictions methods and properties of other scripts from the same origin. Yet, it prevents accessing scripts from other sites. The term *origin* is defined using the domain name, protocol and port. In this way, *https://www.example.com* is a different origin than *http://www.example.com* or *https://www.example.com:3000*. Two scripts are from the same origin when the three values, domain, protocol and port, are the same²⁴.

To load content from the browser, an asynchronous HTTP request is done via Javascript. This kind of requests are called XMLHttpRequest (XHR). When the content is not located in the same origin it is said that the script is doing a Cross-Domain XMLHttpRequest (CD-XHR). However, due to the Same Origin Policy described above, scripts cannot directly do Cross-Domain XMLHttpRequests. There are three workarounds for doing these kind of requests: using the server as a proxy²⁵, Flash²⁶ and the JSONP²⁷ technique.

Workarounds for CD-XHR	
Server-proxy	Using the server as a proxy to access to third party servers
Flash	A crossdomain.xml file is placed on the Server that allows the client's browser to bypass the Same Origin Policy using Flash
JSONP	This technique implies dynamically inserting Javascript tags (that are not bonded to the policy) importing files from third-party servers

Table 3.7: Workarounds to perform Cross-Domain XMLHttpRequests

²³See http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy for a full description of the Policy and comparisons in different browsers

²⁴See <http://tools.ietf.org/html/draft-abarth-origin-07> for a discussion of the concept

²⁵See <http://developer.yahoo.com/javascript/howto-proxy.html>

²⁶See http://jimbojw.com/wiki/index.php?title=Cross-domain_Ajax_via_Flash

²⁷See <http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/>

The Same Origin Policy has historically caused security problems since it is not well-defined and its implementation in different browsers can differ in edge cases. This happens for example with protocols that do not have a clearly defined host name or port, such as *file://*.

Noticeable, The iPhone browser does, with a few exceptions²⁸, allow Cross domain requests easing the task of building Mobile Web Mashups on the client side. However, as far as the writer knows, this option is not possible in others mobile browser.

Looking to the future, the *Web Applications Working Group* within the W3C has proposed a mechanism to enable client-side cross-origin requests²⁹ based on HTTP headers. This standard has been already implemented in the Desktop browser Mozilla 3.5³⁰ (and above) and could be implemented in the future in Mobile Browsers allowing CD-XHR.

3.6.3 Mobile architecture

The third type of architecture that we can build is called Mobile. The idea behind this design is to leverage into the advantages of each architecture and overcome their disadvantages. Figure 3.11 shows a diagram representing this architecture.

The TELAR Mobile Mashup platform developed by Brodt and Nicklas follows this kind of architecture [4]. This platform contacts directly a map service from the device while different data providers are wrapped and proxied via the server. Another example of this kind of architecture could be HousingMaps.com (Figure 3.3). This Web Mashup loads the maps using the Javascript library provided by Google Maps directly from the client's browser avoiding to load the server with unnecessary information. On the other hand, the information from Craigslist is not available via an API, therefore the server is using a technique called *Web scraping* that gets the information directly from the HTML code of the website. This

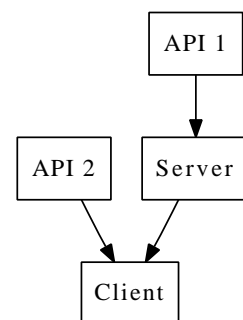


Figure 3.11: Mobile architecture

²⁸See <http://developer.apple.com/library/safari/#documentation/AppleApplications/Conceptual/SafariJSProgTopics/Articles/XHR.html>

²⁹See <http://www.w3.org/TR/cors/>

³⁰See https://developer.mozilla.org/En/HTTP_access_control

requires loading all the Craigslist.org website to extract the desired information. This can be considerably slow to do on a mobile phone due to the restricted computational power and the limited bandwidth. It also increases the consumption of the battery significantly.

Table 3.8 shows reasons to use client or server based architectures. Note that a Mobile architecture can combine advantages from both.

Server-based	Client-based
Several libraries	Easier to implement
Security	No server-code required
Concurrent and asynchronous calls	No bottleneck
Pre-process the data	Better performance
Cache information	Avoid server load
More computational power	

Table 3.8: Reasons to use client or server based architectures

Once the architecture is in place, how the interface of the Mashup needs to be dicussed. The mobile interface is the topic for the next section.

3.7 The mobile interface

A Mobile Web Mashups is a Mashup built with Web technologies and accessible from mobile web browsers. The first consideration is therefore what kind of browsers are the most common ones among the mobile devices. As we pointed out in section 2.1, most of the smartphones have a Webkit-based browser including those with the operating systems iOS, Android, WebOS, Symbian, Maemo, Bada and Blackberry OS with the notable exception of Windows Mobile. Webkit has a great support for the new HTML5 standard with offline capabilities³¹ and CSS3 with hardware-based animations³².

Even though all these browsers are Webkit-based, not all of them behave in the same way. Depending on the implementation done by the device manufacturer,

³¹<http://www.w3.org/TR/html5/offline.html>

³²<http://www.w3.org/TR/css3-animations/>

some differences exist and have to be addressed in order to truly provide a cross-platform solution³³.

To ease the task of developing Mobile Web Mashups, it is possible to use one of the existing Mobile Web tools.

3.7.1 Tools

Table 3.9 shows some of the most common Mobile Web tools. These tools provide a generic functionality that can be extended by the user. Most of them focus on the user interface, with animated transitions between different pages and AJAX requests to improve the user experience.

UI tool	URL	Notes
jQTouch	http://www.jqtouch.com/	Based on JQuery. Native animations. Active development
iUI	http://code.google.com/p/iui/	Simple. No Javascript coding required
iWebkit	http://iwebkit.net/	Mature (v5.04)
WebApp.net	http://webapp-net.com/	Extensive documentation
SproutCore	http://www.sproutcore.com/	Early stage, originally for desktop web. Used by Apple

Table 3.9: User interface tools

An important aspect of Mobile Web Applications is which kind of *page model* is used. Traditionally, Websites and Web applications are built using a Multi-Page Interface. However, most Mobile Web tools use the Single-Page Interface.

3.7.2 The page model

When it comes to the page model there are two possible options, a Multi-page interface or a Single-page interface.

³³For a comparison of different Webkit-based browsers check: <http://www.quirksmode.org/webkit.html>

In the Multi-page Interface the user navigates a hierarchical tree of Web pages. This model is appropriate for linear processes such as reading articles. Articles can be broken down into several pages and let the user navigates over the pages with Next and Previous buttons. Since reading articles is a linear process, this navigation model does not present problems for the users [35].

With a Single-page Interface instead of having a hierarchy of Web pages, the browser loads one page that acts as a container and then additional content is loaded using Ajax calls based on the users' actions. Thus, all users' actions take place on one page and the content shows or hides based on those actions. The page can be partially refreshed instead of having to load a whole new page with each interaction, therefore increasing the system's responsiveness and speed. Single-Page Interfaces allow users to work in a non-linear way and can create rich user experiences [9]. On the other hand, there are a number of issues to take into consideration such as searchability, history management (Back and Forward buttons) and accessibility. Most of these problems arise due to the fact that there is no URL for each different state of the application [8].

A deeper discussion into the Single-page Interface and its advantages can be found in http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php.

Using one or other page model can affect considerably the user experience. However, this kind of problem does not arise in Native Mashups, i.e. native applications that do not run on a browser.

3.7.3 Advantages and Disadvantages

Mobile Web Mashups present a number of advantages, mainly in its rapid-prototyping approach and multi-platform support, and disadvantages as we can see on Table 3.10.

Remarkably, there is a compromise solution that, leveraging on Web technologies, allows having all the advantages of native applications while still fulfilling the cross-platform requirement: Native Web Applications.

3.7.4 Native Web Applications

A Native Web Application consist on developing the application using web technologies that runs on a browser wrapped in a native application. The functionality of this browser has extended hardware features, allowing access to, for example,

Advantages	Disadvantages
Easy and cheap to create and maintain	Overall user experience limited to the browser capabilities
Developers can use known technologies, tools and techniques	No access to OS (ex. address book) and most of the Hardware.
Easy to publish: There is no need to install any software	Harder to discover, no centralized <i>App store</i>
No need to update the software: cloud based	Scalability issues: cloud applications might create bottlenecks
Apps stores review processes no needed	Privacy issues: Confidential information that should not be on the cloud
	Can be slower due to be running in a browser and the network connection
	Limited off-line mode support

Table 3.10: Advantages and disadvantages of using a Web Interface

the accelerometer, within the Javascript code. Table 3.11 shows several tools that ease the development of this kind of applications.

Native tool	URL
PhoneGap	http://phonegap.com/
AppAccelerator Titanium	http://www.appcelerator.com/
Rhodes	http://rhomobile.com/

Table 3.11: Native Web applications tools

An example of Javascript code using the mobile's accelerometer with PhoneGap³⁴ is shown below:

```
function onSuccess(acceleration) {
    alert('Acceleration X: ' + acceleration.x + '\n' +
        'Acceleration Y: ' + acceleration.y + '\n' +
        'Acceleration Z: ' + acceleration.z + '\n');
};

function onError() {
    alert('onError!');
}
```

³⁴Example from: http://docs.phonegap.com/phonegap_accelerometer_accelerometer.md.html

```
};
```

```
navigator.accelerometer.getCurrentAcceleration(onSuccess, onError);
```

With all the blocks in place, now is possible to define a reference framework that summarizes all the different aspects of Mobile Web Mashups.

3.8 Reference Framework

With all the information analyzed in the previous sections, a reference framework has been built. This framework (see Table 3.12) summarizes all the different aspects when building Mobile Web Mashups and helps taking early decision on the development.

The first step to develop a new Mobile Web Mashup using this framework is to have a rough idea on what kind of service will be and who will be using it. Then, decide what kind of Mashup it is—Consumer, Data, Business—based on the target audience. A rough estimation of the number of potential users and how they might use the Mashup are also recommendable.

Subsequently, information about the different services that provide the functionality required should be gathered. Things to take in consideration are if the data is private or public, if it is freely accessible or a fee is required and the limitations of the Terms of Service. Additionally it is needed to know what protocols these services understand and which data formats are available.

Based on this information, an architecture should be chosen. For example, if the Mashup has a potentially big number of users, a client-based architecture might be a good solution to reduce the load on the server. On the other hand if the service providers only work with a SOAP protocol, a proxy will be needed and some kind of server-based architecture would be more appropriate.

Finally, what kind of information is provided and the requirements and limitations of the interface should help deciding the type of the mobile interface. For example, depending on the type of data to visualize, a Multi-page model or Single-page might be more appropriate. On the other hand, if the application requires the usage of some hardware features such as accelerometer or the camera, a pure browser-based application is not possible and a native web application should be developed.

Additionally, it is possible to use Table 3.12 as a *Mashup ID* to see at a glance its most important characteristics and to understand how it works. This Mashup ID

will be used in the next chapter whenever a new Mashup is introduced.

Table 3.12 can be used to analyse Mobile Web Mashups. It can be useful to take into consideration all the different requirements and aspects of new services for early decisions. On the following chapter a number of examples for Mobile Web Mashups is presented and analyzed using this framework.

Section		Notes
Type (3.2)	Data Mashup	Combine many sources of similar types into a single representation.
	Consumer Mashup	Combine many sources of different types into a visual representation.
	Business Mashup	Similar to consumer type with the aim to solve a business problem.
Service provider (3.3)	Open/Proprietary	Check Terms of Service for usage limitations and fees.
	Public/private data	Private data might need to use Authentication protocols (such as OAuth).
Protocol (3.4)	SOAP	W3C standard. Widely used in the industry. Avoid on client side.
	REST	Based on HTTP protocol. Widely available. Best performance on client side.
Data format (3.5)	XML	W3C recommendation. Widely used. Avoid on client side.
	JSON	Based on Javascript notation. It has the best performance on client side.
Architecture (3.6)	Server-based	Information gathered and processed in the server. Can cause bottlenecks.
	Client-based	Executed within the client device. Problems with the Same Origin policy.
	Mobile	A combination of Client and Server-based architectures.
Interface (3.7)	Mobile Web tools	Provide a generic framework and functionality for the user interface. Most common ones: jQTouch and iUI.
	Page model	Multi-page model for hierarchical tree of Web pages. Single-page for Ajax based applications.
	Native Web Apps	A web application wrapped into a native application. Use if it is required to access hardware. e.g. PhoneGap.

Table 3.12: Reference framework for Mobile Web Mashups

Chapter 4

Case Studies

In this chapter the reference framework developed in the background has been applied to an already existing use case of a Mobile Web Mashup—Telar—as described in [4]. Subsequently, it has been used to make a pre-analysis of the requirements and characteristics of four examples taken from 1.1.4 and 2.4. This pre-analysis is meant to help taking early decisions like which kind of data formats to use, whether to have a certain server-side component or which page-model is the most appropriate. In the next chapter two of these examples will be developed and further results presented.

4.1 Telar

Telar is a Mobile Mashup platform that enables the creation of Geo-Mashups using heterogeneous third-party data providers. Telar presents the information of these data providers on top of a map provided by Google Maps where users are able to visualize their surroundings. The map is centered in the user's current location and provides geo-localized information from services such as Fon access points¹, Panoramio pictures and geo-referenced Wikipedia articles offered by GeoNames² (Figure 4.1). Telar was developed by Brodt and Nicklas and presented in [4].

In the following subsections each one of the factors described on the Reference Framework (Section 3.8) will be analysed.

¹<http://www.fon.com>

²<http://www.geonames.org>



Figure 4.1: Telar Mobile Mashup platform [4]

Type

Telar as a platform is meant to be used to create generic Geo-Mashups for any kind of target audience. However, the example presented in [4] is presumably a consumer type of Mashup given the data that is shown: Fon access points, Panoramio pictures and Wikipedia articles.

Service providers

As said earlier, Telar makes use of different service providers: Google Maps, Fon, Panoramio and GeoNames. All the services are open without fee. Since they provide public data, no authentication is required.

Protocols

Google Maps provides a Javascript library (Section 3.4) that communicates directly with their servers, being browser-based presumably it is using a REST protocol. Fon, Panoramio and GeoNames are wrapped in the server side and commu-

nication is carried out directly between the server and the client's browser, therefore a REST protocol might have been used as well. However, this information is not available.

Data formats

Google Maps is using a Javascript library and all the calls are to functions that returned directly Javascript objects therefore it is difficult to tell which format was used. Most likely it is JSON given the Javascript nature of the library. GeoRSS was originally used as data format for the data providers wrapped on the server-side. However, after testing it, the performance was not optimal, consequently the user experience suffered. In an improved version of the demo Mashup they shifted to JSON and it was proved to be a much better approach providing a faster response. This is in line with the recommendation given in 3.5 of using the JSON format to transmit data to mobile devices.

Architecture

Figure 4.2 represents the generic architecture used by the Telar platform. In the particular case that they implement in [4] the architecture looks like Figure 4.3.

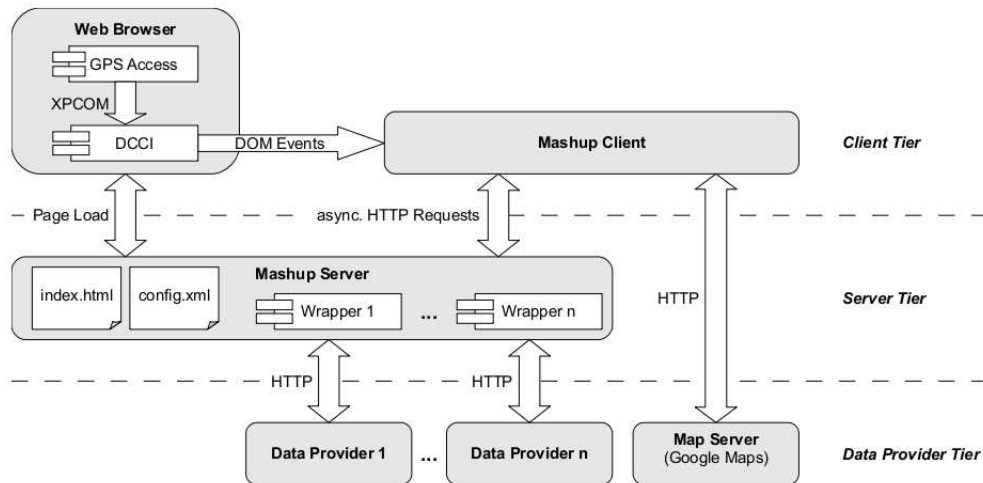


Figure 4.2: Architecture of the Telar Mashup [4]

We can see that the communication from the mobile device with the mapping service is done directly without proxying it in the server. On the other hand, all

the different data providers are proxied in order to avoid the Same Origin Policy (Section 3.6.2) and to homogenize the different data formats used by them.

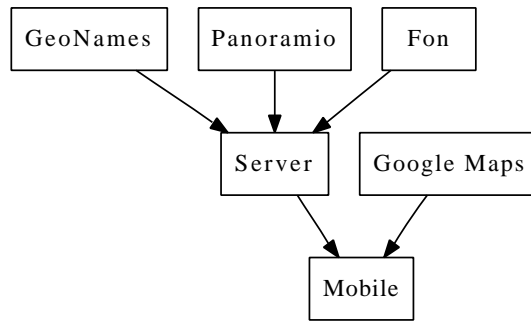


Figure 4.3: Telar example architecture

As we can appreciate, this architecture fits into the mobile architecture presented in 3.6 where some of the services are accessed directly from the mobile device and some of them via the server. This leverages into the advantages of both, server-based and client-based Mashups.

Interface

According to the information provided in [4], data is asynchronously loaded using Javascript. The first time the application is started, all the Javascript files and containers are loaded and subsequent calls just retrieve partial information. This accords with the Single-page model introduced in 3.7.2. Other than that, no Javascript framework was used to build the interface.

Mashup ID

Given all this information, we can now represent it in the Mashup ID proposed in 3.8 as we can see in Table 4.1.

The reference framework developed in this work and introduced in 3.8 has successfully dissected and described how the Telar Mashup works. Additionally, we have seen that some of the problems the authors faced could have been avoided using this framework.

In the following sections the reference framework will be used to carry out a pre-analysis of four examples taken from 1.1.4 and 2.4. This pre-analysis will help

Telar	
Type	Consumer
Service providers	Google Maps, Fon, Panoramio, GeoNames
Architecture	Mobile
Protocols	REST
Data formats	JSON
Interface	Single-page

Table 4.1: Mashup ID of Telar

us to understand more aspects of the Mashup, to take early decisions on which services to use and to begin designing the architecture.

The chosen examples were taken in order to see different aspects of Mobile Mashups. Some of them are more server-based oriented than others or with a bigger importance on the social or location aspect. In the following chapter, two of these examples will be implemented and the experience while developing them described.

4.2 SoundSquare

SoundSquare was originally introduced in Subsection 1.1.4 and its high-level representation is shown in Figure 1.1. SoundSquare is a Mobile Web Mashup that merges the services of Foursquare and SoundCloud. As mentioned before, FourSquare allows users to "check-in" at venues using text messaging or a device specific application³. On the other hand,

SoundSquare	
Type	Consumer
Service providers	Foursquare, SoundCloud
Protocols	REST
Data formats	JSON
Architecture	Server-based
Interface	Multi-page. jQ-Touch

Table 4.2: Mashup ID of SoundSquare

³<http://www.washingtonpost.com/wp-dyn/content/article/2009/03/18/AR2009031802819.html>

SoundCloud is an online audio distribution platform which allows musicians to collaborate, promote and distribute their music⁴. The idea behind this Mashup is to let people link FourSquare venues with music, creating a soundtrack list for each different location. New visitors to a venue could listen to what other users have selected as soundtrack of the place. In this way, any restaurant or cafe can have their own playlists created by the very customers of the venue.

Type

SoundSquare is meant to be a Consumer type of Mashup. The target audience of the application is regular users of FourSquare that have a special interest in music and would like to discover new tracks.

Service providers

Both services, FourSquare and SoundCloud, are open, without limit of usage and do not require paying any fee. Both services allow users to have their own profile and private information. This information is available with the right authentication, being possible to use an HTTP basic authentication or the OAuth 1.0 protocol.

Protocols

Noticeably, neither SoundCloud nor FourSquare support SOAP or any other protocol other than REST. Therefore REST is the chosen protocol when communicating with their services.

The communication between the Server and the Client will be carried out as well using the REST protocol. This decision also fits with the recommendation given in 3.4 of using the REST protocol when communicating with a mobile device.

Data formats

Both services allowed XML and JSON format. JSON should be the format used in both cases since the data size is generally smaller.

⁴<http://eu.techcrunch.com/2010/05/18/now-a-million-on-soundcloud-this-startup-is-scaling-globally/>

Architecture

As described in the Subsection 3.6.2, the Same Origin Policy does not allow scripts from one origin accessing information of a different origin. In order to access the information of FourSquare and SoundCloud a server-based proxy solution should be implemented. SoundSquare should perform queries against its own server and the server is in charge of the communication with FourSquare and SoundCloud. Figure 4.4 represents the architecture of SoundSquare.

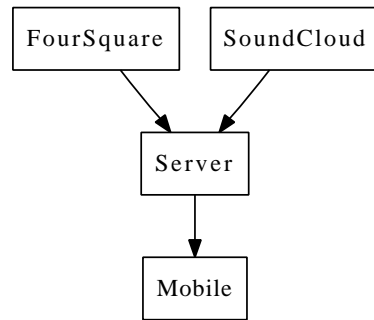


Figure 4.4: SoundSquare architecture

In order to create a rapid prototype of the Mashup, a back-end framework such as Ruby on Rails would be appropriate.

Interface

In 3.7.2 we introduced two types of page models: Multi-page and Single-page. Since the type of data that is going to be presented in SoundSquare, lists of venues with their own lists of tracks, seems to be quite linear, a Multi-page model is the chosen option. The jQTouch framework could be used to ease the task of building the interface.

4.3 Antipodes

Antipodes, also introduced in Subsection 1.1.4, lets the users visualize their current location as well as their antipodes location in the world. The idea is that, at any given moment the user is able to see where he is located using a map provided by Google Maps and be able to toggle the view and visualize the same kind of information for the antipodes location in the world, i.e. the place on Earth's surface which is diametrically opposite to the current location. More information from other services is included such as the times for sunrise and sunset provided by the Norwegian Meteorological Institute (DNMI) as well as geo-located pictures from Flickr.

Antipodes	
Type	Consumer
Service providers	Google Maps, DNMI, Yahoo YQL, Flickr
Protocols	REST
Data formats	XML, JSON
Architecture	Mobile
Interface	Single-page. iUI

Table 4.3: Mashup ID of Antipodes

Type

Antipodes is meant to be a Consumer type of Mashup. The target audience of the application is potentially any mobile user that is curious to know information about the other side of the world.

Service providers

All the services are open and do not require any payment. However, according to their license agreement, Google Maps can demand a payment if the usage grows excessively. All the information is public and no authentication protocol is required.

Protocols

Google Maps provides a Javascript library that communicates directly with their servers being impossible to know exactly which protocol they are using. However, most likely it is REST. DNMI and Yahoo YQL provide their information using a REST protocol.

Data formats

DNMI provided most his content in XML format. Some of the actions have binary proprietary format but none of them was used. Google Maps is using a Javascript library and all the calls were to functions that returned directly Javascript objects therefore it is difficult to tell which format was used, most likely JSON. Yahoo YQL used JSON as data format and the JSONP technique in order to make calls directly from the client.

Architecture

DNMI should be proxied by the server to be able to get the information complying with the Same Origin Policy. On the other hand, as mentioned before, Google Maps makes use of a Javascript library to access its data and thus it is not necessary to use the server as a proxy. Consequently Google Maps information is retrieved directly in the client's browser. Yahoo YQL could be used in order to avoid using the server as a proxy, which can become quite bandwidth costly since it would require to receive and send all the different pictures. As it can be seen in Figure 4.5, the architecture is Mobile.

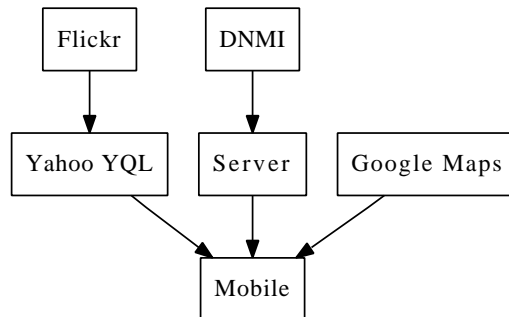


Figure 4.5: Antipodes architecture

Interface

There is only a couple of views showing maps and times with AJAX loaded content. Therefore the Single-page model might be the best option. The iUI framework is chosen to ease the task of building the interface.

4.4 MobActi

MobActi, also introduced in Section 2.4, is a tool for social activism. It allows people to raise awareness about issues in their cities. Users can report problems directly when they see them and list them on GetSatisfaction. Once there, the issues can be voted and sorted out by the community providing clear guidelines for the local authorities on what their citizens would like them to work on. Additionally, using the GPS location issues are

geo-localized creating a heat-map of hot-spots on the cities where more work needs to be done. Once decisions are taken or the work is done, daily status reports are broadcasted to the community using Twitter and are available on MobActi.

MobActi	
Type	Consumer
Service providers	Google Maps, GetSatisfaction
Protocols	REST
Data formats	JSON
Architecture	Mobile
Interface	Single-page. iWebKit

Table 4.4: Mashup ID of MobActi

Type

MobActi is meant to be a Consumer type of Mashup. The target audience of the application are citizens of any city that are willing to contribute to their community with feedback.

Service providers

All the services are open not requiring to pay any fee. However if the usage of Google Maps grows too much, Google is able to demand for a fee according

to their license agreement. All the information is public and no authentication protocol is required.

Protocols

As mentioned before, Google Maps presumably uses a REST protocol. Get Satisfaction uses a REST protocol as well.

Data formats

GetSatisfaction provides its data in HTML or JSON formats, in order to manipulate it in a programmatically way the JSON format is a better option. Google Maps is most likely using JSON as well as mentioned earlier.

Architecture

GetSatisfaction should be proxied by the server to be able to get the information avoiding the Same Origin Policy and store meta-information (such as the location in where the issue was raised). On the other hand, Google Maps provides a Javascript library to access its data and thus it does not need to be proxied by the server. Figure 4.6 shows that the architecture of MobActi is Mobile.

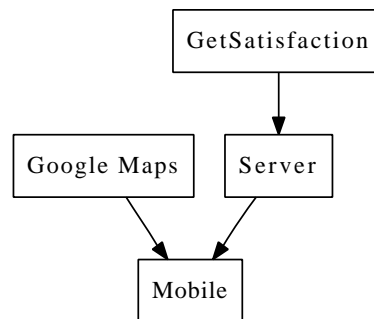


Figure 4.6: MobActi architecture

Interface

There are just a few views and the number of features is quite reduced, therefore a Single-page model might be better to implement. The iWebKit framework could be used to ease the task of building the interface.

4.5 Deventus

Deventus, introduced in Section 2.4, is a mobile conference system that allow attendees have all the information they need at the palm of their hand. Not only they can connect with other attendees via Facebook and LinkedIn, but real-time interaction with them is possible via Twitter, where users can express their opinion and raise questions live. The application contains information about the venue and its location using Google Maps. Convenient QR-codes generated with Kaywa are available for the user to exchange for drinks in the after party. When the event is finished, all the different talks are available via Vimeo.

Deventus	
Type	Business
Service providers	Google Maps, LinkedIn, Facebook, Twitter, Kaywa, Vimeo
Protocols	REST
Data formats	JSON
Architecture	Mobile
Interface	Multi-page. jQ-Touch

Table 4.5: Mashup ID of Deventus

Type

Deventus is meant to be a Business type of Mashup. The target audience of the application are attendees of events and conferences.

Service providers

All the services are open not requiring to pay any fee. Using the OAuth protocol for authentication is required in Facebook and LinkedIn.

Protocols

Google Maps provides a Javascript library that communicates directly with their servers being impossible to know exactly which protocol they are using although,

presumably, it is REST. Vimeo, Kaywa, Facebook and LinkedIn uses the REST protocol as well.

Data formats

LinkedIn provides its content only in the XML format. Google Maps is using a Javascript library and all the calls were to functions that returned directly Javascript objects therefore it is difficult to tell which format was used, most likely JSON. Vimeo, Kaywa and Facebook have all the content available in JSON format as well.

Architecture

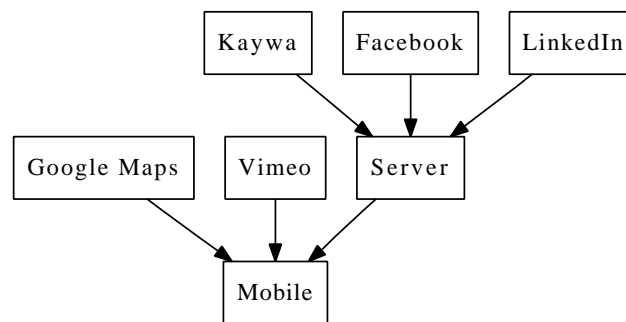


Figure 4.7: Deventus architecture

Facebook and LinkedIn required being proxied by the server to be able to store meta information as well as to handle the authentication via OAuth. On the other hand, as explained earlier, Google Maps provides a Javascript library to access its data and thus it does not need to be proxied by the server. Figure 4.7 shows that the architecture should be Mobile.

Interface

Since the type of data that is going to be presented in Deventus is quite varied—Maps, lists of users, Videos, Real Time information, etc—it might be better to use a Multi-page model in order to decouple the code in multiple files. The jQTouch framework could be used to ease the task of building the interface.

In order to test the described reference framework, the two first Mobile Web Mashups have been developed. In the following chapter results of these implementations are shown.

Chapter 5

Results

In order to empirically analyze and test the reference framework, the development of two Mobile Web Mashups that have been previously analysed in Chapter 4 has been carried out. In this chapter we describe the details of both examples and we share the experience acquired while implementing them.

5.1 SoundSquare

The Mashup can be found in <http://soundsquare.viedma.es>

Its source code can be found in <http://github.com/cviedmai/SoundSquare>

Originally SoundSquare was developed and tested using standard Desktop Web browsers. Standard tools for web development, such as the Firebug extension of Firefox and the Inspector of Chrome (common to all the Webkit-based browsers) were useful for debugging and testing the application. However, once the application was tested on a mobile device, it was much slower than expected due to a false impression of good performance on the computer's browser. The mobile

SoundSquare	
Type	Consumer
Service providers	Foursquare, SoundCloud
Protocols	REST
Data formats	JSON
Architecture	Server-based
Interface	Multi-page. jQ-Touch

Table 5.1: Mashup ID of SoundSquare

browser, and especially the execution of Javascript code, was slow and the application responsiveness was poor causing a bad user experience. Figure 5.1 and show screen-shots of SoundSquare running on a browser.

Subsequently, results for all the different aspects of Mobile Web Mashups are presented.

Type

There are no requirements defined for Consumer type Mashups therefore it cannot be stated whether we are conforming to them. This might be something worth working on future work (see 6.4.2).

Service providers

SoundCloud required the application to be registered in order to perform queries to their services, even if it was public data available to everyone. FourSquare did not require the application to be registered. To access private information, user authentication was required in both services. Public information was available directly via their REST API without any requirement.

Even though there are some libraries that ease the task of authenticating via OAuth, for a novice developer all the different steps and tokens exchanged might be slightly confusing (see Sub-section 3.3.1). Not only this, but the fact that a redirection to a third party web site is required seemed a bit confusing in the mobile context. For these reasons, when authentication was required in FourSquare the simpler xAuth¹ credentials, a sub-protocol of OAuth,

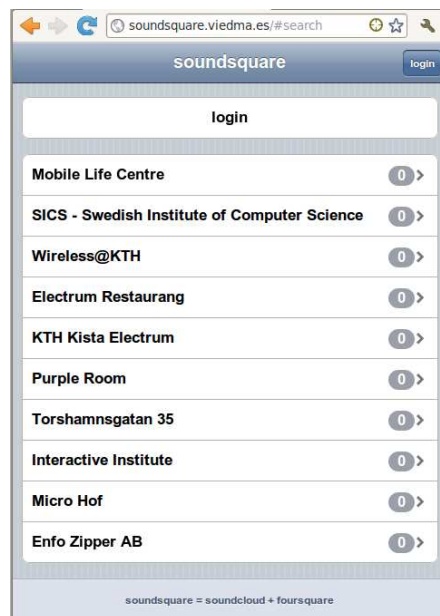


Figure 5.1: Screen-shot of SoundSquare running on a browser

¹<http://dev.twitter.com/pages/xauth>

were used. This sub-protocol allows to get an access token by simply providing the username and password of the user.

The service offered by FourSquare proved to be unreliable during the development. It seemed that their service, being quite new, experienced peaks of usage for which it was not prepared. In other occasions, FourSquare and SoundCloud were available but with huge delay times. Originally SoundSquare was blamed for the poor user experience discovering later that it was a problem with FourSquare. SoundSquare assumed that the third-party services were always available and it did not notify of the problems, it just did not react to user's actions causing a poor user experience by looking unreliable and faulty.

Protocols

FourSquare and SoundCloud were accessed using the REST protocol which does not include any kind of formal description such as the *Web Service Description Layer (WSDL)*². Without having a proper documentation it would be impossible to know which actions or resources are available, therefore, the documentation provided by the services is extremely important. Both services had an extensive documentation with several examples which was really useful.

Data formats

The information from both services was retrieved directly from the server using the JSON data format. On the other hand, the server sent the information to the mobile device using HTML, i.e. the information was sent already formatted. The reason for this was that Ruby on Rails has a rather easy way to generate HTML views making it possible to have the results directly visible on the Mobile without much extra work. However the application was slow and unresponsive leading to a poor user experience. This approach proved to be unsuccessful since the overhead of HTML is considerable. HTML, being a markup language like XML, might have an overhead similar to it (Figure 3.8). A much better approach would have been sending the information directly in JSON to the mobile device, as suggested in Section 3.5, and format the content directly using Javascript.

Architecture

²http://en.wikipedia.org/wiki/Web_Services_Description_Language

One of our goals was to analyze possible architectures for Mobile Web Mashups (Section 1.3). As described earlier in the reference framework (Section 3.6) one of them is server-based. SoundSquare used this kind of architecture as we suggested in the case study (Section 4.2). The mobile sent requests to the server and this communicated directly with the services acting as a proxy. The server-side was implemented using the Web development framework Ruby on Rails. This tool is famous for its rapid development approach, making possible to generate an skeleton of a web service with a few automatic scripts.

SoundCloud had a library wrapping all the calls to their servers ³ written in *Ruby*, the language used on the back-end. This wrapper proved to be very convenient, facilitating the development. FourSquare did not have such a library, consequently, following the case of SoundCloud, a module wrapping all the calls to the FourSquare service was implemented.

Interface

It was in our goals to test different tools for the mobile interface (Section 1.3). The jQTouch tool proved to be a useful tool to create good-looking rapid prototypes dealing automatically with issues like the history management. However, this tool is focused on a Single-page Interface and it was cumbersome to use it with the chosen page model. Additionally the documentation available is rather scarce and it took a considerable amount of time to understand how it works. One of the best tutorials available can be found in a book written by Jonathan Stark [20], who is now the person in charge of maintaining the code of jQTouch⁴.

Ruby on Rails is a rather Multi-page Interface oriented framework making it easy

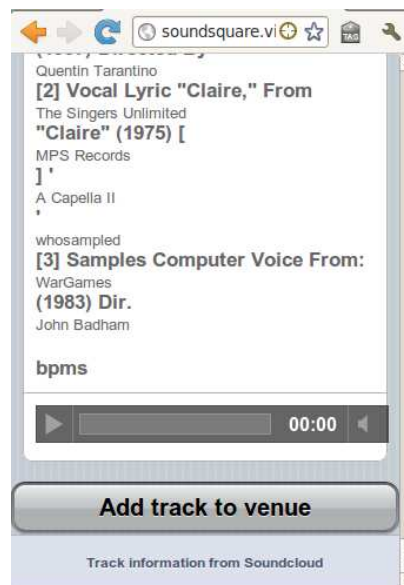


Figure 5.2: Screen-shot 2 of SoundSquare running on a browser

³<http://wiki.github.com/soundcloud/ruby-api-wrapper/>

⁴<http://github.com/senchalabs/jQTouch>

to use this page model (see Subsection 3.7.2. However, as described earlier, sending multiple HTML files to the mobile device was slow and caused considerable delays. A Single-page model might have been a better approach for this kind of applications.

When reading about how to access the location information using Javascript from the mobile device, it was found that there are many different implementations depending on which platform the application is going to be used. Even though the Mashup was just tested on Android the *geo-location-javascript v0.4.3* library was used. This library provides a geo location tool that wraps all the underlying platform specific implementations in a Javascript library which is aligned to the W3C Geo-location API Specification⁵.

5.2 Antipodes

The Mashup can be found in <http://antipodes.viedma.es>

Its source code can be found in <http://github.com/cviedmai/Antipodes>

As with the previous example, Antipodes was originally developed and tested using standard Desktop Web browsers. Once again, standard tools for web development like Firebug and Inspector were very useful during testing and debugging. However this time the application was frequently tested in an Android mobile phone. More than debugging, the tests on the mobile platform were carried out to see how well the application was perform-

Antipodes	
Type	Consumer
Service providers	Google Maps, DNMI, Yahoo YQL, Flickr
Protocols	REST
Data formats	XML, JSON
Architecture	Mobile
Interface	Single-page. iUI

Table 5.2: Mashup ID of Antipodes

ing, how fast it was loading and the overall user experience. This early-testing approached seemed more successful than our previous experience. Figure 5.3 and 5.4 show screen-shots of Antipodes running on a browser.

The following subsections present results for all the different aspects of Mobile

⁵<http://code.google.com/p/geo-location-javascript/>

Web Mashups.

Type

Once again, the application was designed as a Consumer Mashup type being not possible to state if it conforms to such a type (see 6.4.2).

Service providers

The service providers used were Google Maps, the Norwegian Meteorological Institute (DNMI), Flickr and Yahoo YQL. None of these services required authentication neither an API key.

Even though Flickr allows to access the data from a third-party service using the JSONP technique (see 3.6.2), *Yahoo Query Language* (YQL) was used as a proxy in order to evaluate it as an alternative to a server-side solution.

YQL allows the developer to access multiple APIs from a single Web Service using a SQL-like syntax. Instead of having to deal with different APIs with different syntaxes, YQL allows the developer to access multiple services using a common language. It is possible not only to access the already available web services on YQL, but the developer can also define his own services mapping correctly the API into verbs such as SELECT, DESC and SHOW⁶. Finally, it also allows getting data from any valid HTML document using XPATH. As stated in the home-site of YQL⁷:

⁶<http://datatables.org/>

⁷<http://developer.yahoo.com/yql/>

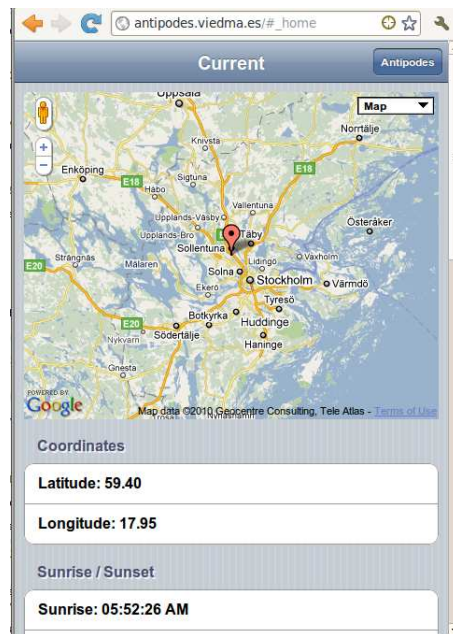


Figure 5.3: Screen-shot of Antipodes running on a browser

With YQL, developers can access and shape data across the Internet through one simple language, eliminating the need to learn how to call different APIs.

Using YQL was notably easy with mainstream Javascript tools such as *Prototype*⁸ and *JQuery*⁹. Following a full example of the written code:

```
function get_pictures_yql(lat, lon){

    var BASE_URI = 'http://query.yahooapis.com/v1/public/yql';

    var yql_query = "select * from flickr.photos.search where "
    yql_query += "min_taken_date='2008-09-01' and lat='" + lat + "' "
    yql_query += "and lon='" + lon + "' and radius='20'";

    new Ajax.JSONRequest(BASE_URI, {
        onComplete: processJSON,
        callbackName: 'callback',
        parameters: {q: yql_query, format: 'json'}
    });
}
```

The potential of YQL will be discussed in the conclusions (Subsection 6.1.1).

Protocols

Google Maps provided a convenient Javascript library, dynamically loaded, that allowed to show a map with a couple of lines of code. This kind of Javascript's libraries bypass the same origin policy and are quite convenient from a developers perspective since it does not require dealing directly with the Web Service, building URLs or wrappers. The functionality is ready to use.

The Norwegian Meteorological Institute only offered a REST protocol while Flickr was the most complete allowing calls in REST, SOAP or even XML-RCP. As described earlier Flickr was accessed via YQL which made querying for photographs a very easy task. A point to improve might be to provide directly the URL of the picture on the request data instead of instructions on how to build the URL. The following Javascript code was used for such a task.

```
var src = "http://farm"+farmid+".static.flickr.com/"
```

⁸<http://www.prototypejs.org/>

⁹<http://jquery.com/>

```
src += serverid+"/"+id+"_"+secret+"_b.jpg";
```

Note that, even though it is quite simple, it is not very versatile. If in a later stage they decide to change their URL structure all the already existing applications will face problems.

Data formats

The two data formats used were XML and JSON. XML was used only on the server side while all communications with the mobile were done using JSON, either from the server or from the third-party Web Services. This, as pointed out in 3.5.2, JSON usage is recommended for this kind of devices and the overall application speed increased considerable. Remarkably, even though the type of content provided on Antipodes are mostly images (maps, pictures) as opposed with SoundSquare (tracks, locations), Antipodes user experience is much more fluid and the application loads almost instantly over Wi-Fi.

JSON was also very convenient to work with since it maps directly into Javascript objects. For example, in order to access a *photo* object the code required in Javascript is directly *result.query.results.photo[0]*.

XML was easy to access from the server side and the number of libraries, tools and documentation available is outstanding.

Architecture

As it was reminded earlier, one of our objectives was to analyze possible architectures for Mobile Web Mashups (Section 1.3). Antipodes uses a Mobile architecture, as described in Section 3.6, using the server component only to proxy the information of the Norwegian Meteorological Institute. Other than this, the role of the server component was secondary just using the Ruby on Rails framework as a container and not much more. Google Maps and Flickr (via Yahoo YQL) were accessed directly from the mobile client as depicted in Figure 4.5.

Interface

In order to test a different tool for the mobile interface and to achieve our objectives (Section 1.3), the interface tool used in Antipodes was iUI.

In this occasion the page model chosen was the Single-page model which, in combination with iUI, was very simple to develop. Both, iUI and jQTouch rely almost completely on the Single-page web application paradigm making it difficult to use multi-pages (as seen with SoundSquare). However, having most of the code in the same page might not be the best solution for large scale mobile applications due to coupling and performance issues.

iUI documentation was deficient. Getting to know how does it exactly work took some time and the number of tutorials available on the Internet is much less than jQTouch in spite of being an older tool. Additionally, iUI does not have all the transition animations between pages like jQTouch.

Those transitions help the user understand the navigation among the different states of the application.

There is a Rails wrapper available for the server-side¹⁰. The aim of this wrapper is to simplify building common tasks such as a back button.

Something that would have been good to have is a website to test the application in different resolutions and layouts. As far as the author knows, it does not exist a good web-based tool for this task.

Once again the Geo-Location-Javascript library was used and proved to be very convenient (see 5.1).

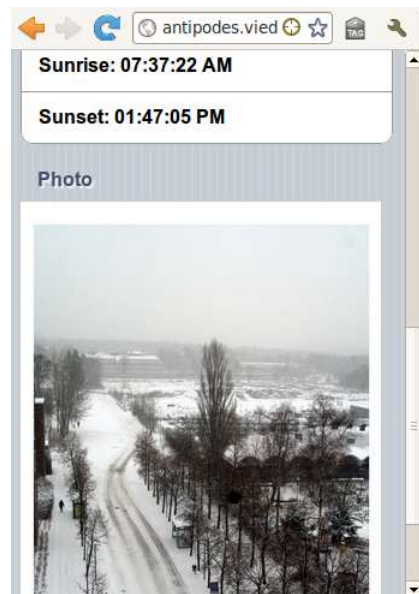


Figure 5.4: Screen-shot 2 of Antipodes running on a browser

¹⁰<http://www.railsloodge.com/plugins/1071-rails-iui>

Chapter 6

Conclusions

In this work we examine Mobile Web Mashups from a developer's perspective sharing our experience with Mobile Web Mashups. In the introduction several goals and objectives were listed (Section 1.3). In this chapter, those goals and objectives are revised. Subsequently, general guidelines to developed Mobile Web Mashups and the main contribution of this work are described. Finally, possible paths for future work are drawn.

6.1 Goals and objectives dissection

The aim of this work is to build a reference framework to understand and analyze Mobile Web Mashups. In order to fulfill this aim and to address the problems presented in Section 1.2, a number of goals and objectives were listed in Section 1.3. The following subsections dissect one by one these goals and objectives.

6.1.1 Architectures

The first goal was to describe and analyze possible architectures for Mobile Web Mashups.

As we have seen in Section 3.6, there are three kind of architectures available: Server-based, Client-based and Mobile. Client-based Mashups are executed within the client's browser which gathers the information directly and presents it. On the other hand Server-based mashups gather, analyze, combine and reformat the information on the server side and transmit it to the client. A Mobile architecture is

when some information is processed on the server side and is then combined with more information on the client side.

The main problem with a Server-based architecture is that it can cause bottlenecks and is harder to develop. A Client-based architecture might not be the best option when high computational power is required and it is important to note that it is limited to the mobile browser's protocols. A Mobile architecture leverages into the advantages of each architecture.

As seen in Antipodes, Flickr was accessed via Yahoo Query Language (YQL). The potential of YQL not only lies in the fact that it is a common language, but on the possibility to use YQL as a proxy to create client-side applications. For example, in the example of Housingmaps (see 3.6.3), we pointed out that Craigslist does not provide an API to access its content and a server-side was needed. With YQL the developer does not need to provide this server component and can just roll out a pure client-based mobile service without needing to worry too much about scaling problems. Additionally it is possible to access Web Services that do not provide a solution for the same origin policy using the JSONP technique¹. To sum it up, with YQL you can:

- Access multiple APIs with a common language
- Use the JSONP technique for Web Services that do not provide this option
- Screen scrap content from the web without a server-side

6.1.2 Tools

The second goal was to describe and analyze different tools to build mobile web interfaces.

A number of tools that provide a basic framework and functionality for the user interface has been listed in Table 3.9. Two tools for building interfaces have been used, jQTouch and iUI. Both of them are meant to be used with a single-page interface making it difficult to use a multi-page model. However, as seen with SoundSquare, a multi-page model is slower and not appropriate with mobile devices. In terms of documentation jQTouch is better documented and there are a number of screen-casts available with at least one book using it.

There is a Javascript library available called Geo-Location-Javascript that was very useful in both applications which provides a geo location tool that wraps all the underlying platform specific implementations.

¹<http://ajaxian.com/archives/yql-converting-the-web-to-json-with-mock-sql>

Table 3.10 presented advantages and disadvantages of using a Mobile Web interface. The multi-platform and easiness of development of a Mobile Web interface stand out as advantages while the limited access to the underlying OS and hardware features are remarkable drawbacks.

Additional tools for building Native Web Application to overcome the above mentioned limitations have been presented in Table 3.11. Among them, PhoneGap is one of the most popular tools.

6.1.3 User experience

The third goal was to describe and analyze how to build Mobile Web Mashups with good user experience.

One of the first issues to take in consideration is the speed and responsiveness of the application. As shown earlier is recommendable to not use the SOAP protocol or the XML data format directly on the mobile device. The communication with Web Services should be done using a RESTful API and the JSON data format. A good practice is also to implement the calls in an asynchronous way, so that the application is responsive and can show the user something is happening in the background.

Service providers quality of service cannot be warranted and with new services, as seen with FourSquare, it might fail frequently. Therefore is recommendable to always prepare for it. Giving constant feedback to the user and messages keeping him constantly updated with the current status of the application. Another possibility is to ensure the quality of service having fall-backs services, for example if Google Maps does not respond it might be possible to use Yahoo Maps as a fall-back solution.

When the user has to authenticate in a third-party service, the OAuth work-flow requires the application to redirect the user to that service where the user logs in, approves the application and then is redirected back to the application. While this seems to be a good solution, it quickly becomes too cumbersome when many services are involved, a situation quite common in a Mashup. Therefore it might be good to require user authentication only when is indispensable. It might be possible to create some kind of OAuth aggregator. This idea will be further explained in the Future Work section (6.4).

6.1.4 Motivation

The first objective was to motivate Mobile Web Mashups.

From a business perspective we have seen that Web Mashups accord with the long tail theory which is based on selling less of more. Combining the concept of Web Mashups with mobile devices can unveil a world of new mashups, Mobile Web Mashups, satisfying the needs of niches and creating extraordinary business values. Additionally it was pointed out that the ability to use different services and to break down business processes into smaller pieces can accelerate considerably the speed at which new business services are deployed. According to Peenikal [18], this will represent a substantial competitive advantage and organizations that are not pioneers in the usage of Mashups will unavoidably find themselves non-competitive.

From a developer's perspective Mobile Web Mashups can reuse existing services and data reducing the development process. They do not require extensive IT skills and can be developed in a rapid-prototyping fashion improving time-to-market. Mashups are better tailored to the end users thanks to less resources required, letting developers focus on the user experience rather than building the basic components. Finally they can spur creativity and the development of new and innovative services as well as better interfaces for already existing services.

Finally, from a user's perspective Mobile Web Mashups allow the creation of new applications and services thanks to the long tail theory and the opening of new markets. Also, as pointed out previously, the applications developed are better tailored to their needs with a better user interface.

6.1.5 Advantages and disadvantages

The second objective was to describe advantages and disadvantages of Mobile Web Mashups.

Among the advantages are those described in the previous subsection while motivating Mobile Web Mashups and those described in Table 2.1. The most remarkable disadvantage is the service reliability while assuring the quality of service. Also it is important to note that most of the data sources on the Internet are not made as a service, i.e. do not provide their content via an API, which means unreliable screen scraping programs are needed. Finally, there is no standards for Web Mashups making it difficult to implement security mechanisms. However, some work has been done on securing private data with protocols such as OAuth as shown in Subsection 3.3.1.

6.1.6 Types

The third and final objective was to describe different types of Mobile Web Mashups.

As described in the Section 3.2, Mashups can be categorized in three types: Data Mashups, Consumer Mashups and Business Mashups. The most common category of Mashups is the Consumer Mashup, designed for the general public.

Consumer Mashups combine many sources of different types into a visual representation while Data Mashups combine many sources of similar types into a single representation. Business Mashups are meant for the industry and allow a better usage of the resources of the enterprises. Business Mashups are similar to Consumer Mashups but their aim is to solve a business problem.

6.2 Guidelines

As described in Section 1.4, the target audience of this thesis are designers and developers. The reference framework that has been built will, hopefully, help them put together all the different pieces of the mobile web service puzzle. After reading this work they should have a foundation on the topic, and the following guidelines, developed in Section 3.8, can help them to start building prototypes of Mobile Web Mashups.

The first step is to have a rough idea on what kind of service it will be and who will be using it. Then, decide what kind of Mashup it is—Data, Consumer or Business—based on the target audience. A rough estimation of the number of potential users and how they might be using the Mashup should also be performed when possible.

Subsequently, information about the services that provide the functionality required should be gathered. Things to take in consideration are if the data is private or public, if it is freely accessible or a fee is required and the limitations of the Terms of Service. Additionally it is needed to know what protocols these services understand and which data formats are available.

Based on all the previous information an architecture should be chosen. For example, if the Mashup has potentially a big number of users, a client-based architecture might be a good solution to reduce the server-load. On the other hand if the service providers only work with a SOAP protocol, a proxy will be needed and some kind of server-based architecture would be more appropriate.

Finally, what kind of information is going to be provided and what are the requirements and limitations of the interface should help to take decisions for the mobile interface. For example, depending on the type of data to visualize a Multi-page model or Single-page might be more appropriate. On the other hand, if the application requires the usage of some hardware features such as accelerometer or the camera, a pure browser-based application is not possible and a native web application should be developed.

As it was stated in Section 1.6, the aim of this work is not to provide a fully optimized framework ready for real-world deployment. However, they can be useful for the targeted audience in an initial stage and there is room to further development in future work.

6.3 Contribution

My major contribution and aim is to set a foundation on how to build Mobile Web Mashups. I have set up a reference framework in order to serve as a base for future work and help developers exploring Mobile Web Mashups. As shown in Table 3.12, in this reference framework I have:

- Categorized Mobile Web Mashups by type and architecture.
- Pointed out which are the best protocols and data formats to use in a mobile context.
- Analyzed different characteristics of Services providers.
- Listed advantages and disadvantages of a Web interface for Mobile Mashups in Table 3.10 and raised a number of issues to take in consideration such as the page model and which tool to choose.

6.4 Future work

6.4.1 Performance indicators

As we have said earlier, no performance indicators have been chosen neither benchmarking carried out. The main reason behind it was to look upon the entire system as a whole without digging into details. However, a possible way to further develop this work could be done choosing appropriate performance indicators

for each one of the different components described in the framework (see Table 3.12). Additionally, some kind of indicator could be develop in order to make comparisons between different Mashups.

6.4.2 Mashups categorization

One of the limitations of this work, as described in Section 1.6, was to develop solely the Consumer type of Mashups. However, since there is no clear definition of the requirements of this kind of Mashup, it could not be assessed whether they conformed to that category as we have seen in the results (Sections 5.1 and 5.2).

It might be necessary to develop further the categorization presented in Section 3.2 with detailed requirements and guidelines for each one of the types. A possible way to carry out this kind of work can be with a case-study approach. Taking a high number of already existing Mashups and classifying them by the target audience. Then, try to see which common factors and patterns are presented. For example, Business kind of Mashups potentially require a higher QoS in order to ensure the stability of the business while compromising the user experience in the Consumer kind of Mashups could be highly negative. Additionally, interviews with different stake-holders might be something worth doing.

6.4.3 Native applications

In the Introduction (Subsection 1.1.3) was said that this work would only focus on Mobile Mashups developed with Web technologies. However, there is a number of applications that cannot be done with these tools, for example background processes or applications that require an immersive experience such as augmented reality. Additionally, there are some issues presented in Mobile Web Mashups, such as the Same Origin Policy, that do not exist in native applications. The framework and guidelines developed in this work could be analysed to see whether they apply for native Mashups and still can help developers building Mobile Mashups.

6.4.4 OAuth aggregator

As it was discussed in Subsection 3.3.1, the OAuth protocol is a very web centric protocol from a user experience perspective. The normal work-flow implies the user being redirected from the Mashup to the service provider where the user has to authenticate, approve and then be redirected back to the Mashup. The problem

is presented when a number of different Web Services are mashed up and accessed from a mobile device. Having all these redirections with each one of the used services and the limitation in screen size presented in mobile devices can cause a bad user experience. On the other hand, the idea behind OAuth protocol is still very valid, and having a key to access your personal data without compromising your password is very valuable.

A possible way to overcome this problem might be to create an OAuth aggregator that works as a gate-keeper to other Web Services. With such a scheme, a Mashup would be required only to be granted access to the aggregator. When the access is required the Mashup should be notifying which third party Web Services needs access to. The user can then grant access to those services for a limited amount of time, for example a single day. This OAuth aggregator at the same time needs to have access to the third party Web Services. However, such an access can be provided only once and work for a long time.

Glossary

API An Application Programming Interface (API) is an interface implemented by a software program to enable its interaction with other software [24]. 2

Haiku A three-line poem in any language, with five syllables in the first and last lines and seven syllables in the second, usually with an emphasis on the season or a naturalistic theme [34]. 2

LBS A location-based service (LBS) is an information and entertainment service, accessible with mobile devices through the mobile network and utilizing the ability to make use of the geographical position of the mobile device [25]. 5

P2P A peer-to-peer, commonly abbreviated to P2P, is any distributed network architecture composed of participants that make a portion of their resources (such as processing power, disk storage or network bandwidth) directly available to other network participants, without the need for central coordination instances (such as servers or stable hosts) [29]. 4

QoS In the field of computer networking and other packet-switched telecommunication networks, the traffic engineering term quality of service (QoS) refers to resource reservation control mechanisms rather than the achieved service quality. Quality of service is the ability to provide different priority to different applications, users, or data flows, or to guarantee a certain level of performance to a data flow [30]. 5

SOA In computing, a service-oriented architecture (SOA) is a flexible set of design principles used during the phases of systems development and integration. A deployed SOA-based architecture will provide a loosely-integrated suite of services that can be used within multiple business domains [32]. 4

Web scraping Web scraping (also called Web harvesting or Web data extraction) is a computer software technique of extracting information from websites.[33]. 34

Bibliography

- [1] C. Anderson, “The long tail,” *Wired*, October 2004. [Online]. Available: <http://www.wired.com/wired/archive/12.10/tail.html>
- [2] Anerousis, Nikos, and M. Ajay, “The software-as-a-service model for mobile and ubiquitous computing environments,” in *Mobile and Ubiquitous Systems: Networking Services, 2006 Third Annual International Conference on*, July 2006, pp. 1–6.
- [3] Apple. (2009, November) Apple announces over 100,000 apps now available on the app store. [Online]. Available: <http://www.apple.com/pr/library/2009/11/04appstore.html>
- [4] A. Brodt and D. Nicklas, “The telar mobile mashup platform for nokia internet tablets,” in *EDBT 08: Proceedings of the 11th international conference on Extending database technology*. New York, NY, USA: ACM, 2008, pp. 700–704.
- [5] P. Chaisatien and T. Tokuda, “A web-based mashup tool for information integration and delivery to mobile devices,” in *Web Engineering*, ser. Lecture Notes in Computer Science, M. Gaedke, M. Grossniklaus, and O. Diaz, Eds. Springer Berlin / Heidelberg, 2009, vol. 5648, pp. 489–492. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02818-2_45
- [6] Citizendium. (2010, April) Mashup. [Online]. Available: <http://en.citizendium.org/wiki/Mashup>
- [7] H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin, “Mashup advisor: A recommendation tool for mashup development,” pp. 337–344, Sept. 2008.
- [8] D. Esposito, “Single-page interface and ajax patterns,” *MSDN Magazine*, 2008. [Online]. Available: <http://msdn.microsoft.com/en-us/magazine/cc507641.aspx>
- [9] B. Fling, *Mobile design and development*. O’Reilly, 2009.
- [10] I. R. Floyd, M. C. Jones, D. Rathi, and M. B. Twidale, “Web mash-ups and patchwork prototyping: User-driven technological innovation with web 2.0 and open source software,” *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pp. 86–86, Jan. 2007.
- [11] P. A. Gunderson, “Danger mouse’s grey album, mash-ups, and the age of composition.” *Postmodern Culture & the Johns Hopkins University Press*, September 2004.
- [12] H. Hamad, M. Saad, and R. Abed, “Performance evaluation of restful web services for mobile devices,” *International Arab Journal of e-Technology*, vol. 1, no. 3, 2010.
- [13] I. Jorstad, E. Bakken, and T. A. Johansen, “Performance evaluation of json and xml for data exchange in mobile services,” pp. 237–240, 2008.

- [14] X. Luo, H. Xu, M. Song, and J. Song, "Research on soa-based platform to enable mobile mashups," *Communication Technology, 2008. ICCT 2008. 11th IEEE International Conference on*, pp. 607–610, nov. 2008.
- [15] E. Maximilien, "Mobile mashups: Thoughts, directions, and challenges," *Semantic Computing, 2008 IEEE International Conference on*, pp. 597–600, aug. 2008.
- [16] E. Ort, S. Brydon, and M. Basler. (2007, May) Mashup styles, part 1: Server-side mashups. [Online]. Available: http://java.sun.com/developer/technicalArticles/J2EE/mashup_1/index.html
- [17] ——. (2007, August) Mashup styles, part 2: Client-side mashups. [Online]. Available: http://java.sun.com/developer/technicalArticles/J2EE/mashup_2/index.html
- [18] S. Peenikal. (2009) Mashups and the enterprise. Mphasis - HP. [Online]. Available: http://www.mphasis.com/pdfs/Mashups_and_the_Enterprise.pdf
- [19] M. Stanley, "The mobile internet report," December 2009. [Online]. Available: http://www.morganstanley.com/institutional/techresearch/pdfs/2SETUP_12142009_RI.pdf
- [20] J. Stark, *Building iPhone Apps with HTML, CSS, and JavaScript*. O'Reilly, 2010.
- [21] M. Suster. (2010, February) App is crap (why apple is bad for your health). [Online]. Available: <http://www.bothsidesofthetable.com/2010/02/17/app-is-crap-why-apple-is-bad-for-your-health/>
- [22] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller, "Performance considerations for mobile web services," *Elsevier Computer Communications Journal*, vol. 27, pp. 1097–1105, 2003.
- [23] A. Voulodimos and C. Patrikakis, "Using personalized mashups for mobile location based services," pp. 321–325, aug. 2008.
- [24] Wikipedia. (2010, April) Application programming interface. [Online]. Available: http://en.wikipedia.org/wiki/Application_programming_interface
- [25] ——. (2010, April) Location-based service. [Online]. Available: http://en.wikipedia.org/wiki/Location-based_service
- [26] ——. (2010, April) Long tail. [Online]. Available: http://en.wikipedia.org/wiki/Long_Tail
- [27] ——. (2010, April) Mashup (web application hybrid). [Online]. Available: [http://en.wikipedia.org/wiki/Mashup_\(web_application_hybrid\)](http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid))
- [28] ——. (2010, April) Mobile operating system. [Online]. Available: http://en.wikipedia.org/wiki/Mobile_operating_system
- [29] ——. (2010, April) Peer-to-peer. [Online]. Available: <http://en.wikipedia.org/wiki/Peer-to-peer>
- [30] ——. (2010, April) Quality of service. [Online]. Available: http://en.wikipedia.org/wiki/Quality_of_service
- [31] ——. (2010, August) Representational state transfer. [Online]. Available: http://en.wikipedia.org/wiki/Representational_State_Transfer
- [32] ——. (2010, April) Service oriented archit. [Online]. Available: http://en.wikipedia.org/wiki/Service-oriented_architecture

- [33] ——. (2010, August) Web scraping. [Online]. Available: http://en.wikipedia.org/wiki/Web_scraping
- [34] Wiktionary. (2010, April) Haiku. [Online]. Available: <http://en.wiktionary.org/wiki/haiku>
- [35] J. Willemsen, “Improving user workflows with single-page user interfaces,” *UX-matters*, 2006. [Online]. Available: <http://www.uxmatters.com/mt/archives/2006/11/improving-user-workflows-with-single-page-user-interfaces.php>
- [36] H. Xu, M. Song, and X. Luo, “A qos-oriented management framework for reconfigurable mobile mashup services,” *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, vol. 03, pp. 2001–2005, feb. 2009.
- [37] K. Xu, X. Zhang, M. Song, and J. Song, “Mobile mashup: Architecture, challenges and suggestions,” pp. 1–4, sept. 2009.
- [38] X. Zhang, M. Song, K. Xu, and J. Song, “Mobile mashup based on p2p: Architecture, design and realization,” in *IT in Medicine Education, 2009. ITIME '09. IEEE International Symposium on*, vol. 1, aug. 2009, pp. 416–420.

Index

- 3G, 1, 10
- Akismet, 24
- Amazon, 2
- API, 24
- APIs, 2
- App Store, 11
- AppAccelerator Titanium, 41
- Apple, 1, 11
- Architectures, 5
- ATOM, 31

- Basic HTTP Authentication, 25
- Beatles, 2
- Boarding, 16

- CD-XHR, *see* Cross-Domain XMLHttpRequest
- Chris Anderson, 2
- Chrome, 58
- Cross-Domain XMLHttpRequest, 36
- CRUD, 29
- CSV, 31

- eCommSource, 14
- eXtensible Mark-up Language, *see* XML

- Facebook, 27
- Firebug, 58
- Firefox, 58
- Flickr, 2

- Geo Location Javascript Library, 62
- GeoRSS, 31
- Google Health, 17
- Groupon, 16

- Haiku, 2
- HTML, 31
- HTTP, *see* HyperText Transfer Protocol

- IBM Mashup Center, 4
- Inspector, 58

- Intel Mash Maker, 4
- iPhone, 1, 11
- iPod Touch, 1, 11

- Javascript, 59
- Javascript Object Notation, *see* JSON
- JQuery, 64
- JSON, 31, 32
- JSON Padding, *see* JSONP
- JSONP, 36

- KML, 31

- LBS, 5, *see* Location based services
- LinkedIn, 26
- Location based services, 5
- Long tail theory, 2, 13

- Microsoft HealthVault, 17
- Mobile Mashups, 3
- Mobile Web Tool, 39
- Multi-page Interface, 39

- Native Web Application, 40
- Nintendo Wii, 14

- OAuth, 34
- OAuth 1.0, 26
- OAuth 2.0, 27

- P2P, 5
- Page Model, 39
- PhoneGap, 41
- ProgrammableWeb, 2, 12
- Protocol, 28
- Prototype, 64

- QoS, 5, *see* Quality of Service
- Quality of Service, 5

- Representational State Transfer, 29
- REST, *see* Representational State Transfer

Rhodes, 41
RSS, 31
Ruby, 61
Ruby on Rails, 61

Safari, 1, 11
Same Origin Policy, 35
Simple Object Access Protocol, *see* SOAP
Single-page Interface, 39
SOA, 5
SOAP, 28
Social network, 1, 10

Techcrunch, 4
Terms of Service, 25
ToS, *see* Terms of Service
Twitter, 2
TXT, 31

Uniform Resource Locator, 29
URL, *see* Uniform Resource Locator
User Generated Content, 12
User-driven development, 7, 13

VoIP, 1, 10

W3C, 28
Web scraping, 24, 37
Web Service Description Language, 60
Webkit, 11, 38, 58
Wired magazine, 2
Withings, 17
Woices, 12
WSDL, *see* Web Service Description Language

xAuth, 60
XHR, *see* XMLHttpRequest
XML, 30, 31
XMLHttpRequest, 36

Yahoo Pipes, 4
Yahoo Query Language, 63, 68
YAML, 31
YQL, *see* Yahoo Query Language

