A Comparison of Terminal Environments for Mobile Web Publishing Clients

Jose Manuel Lucea Anaya
Abstract

**A Comparison of Terminal Environments for Mobile Web Publishing Clients**

Jose Manuel Lucea Anaya

Human beings have always aspired to share their thoughts with each other. Historical inventions such as letters, writing and, later, the printing press have served to underscore this basic need. With the advent of the Internet, and the World Wide Web, people can now reach a global audience in a cost-effective manner, with an almost insignificant barrier of entry.

Services such as blogs and online photo galleries allow users to share their content with their acquaintances or other people located around the world. A characteristic of the Web 2.0 revolution is that people are moving from a passive reader role to a more active one. Previously, the desktop was the only platform that could be used to publish content to the web; however, that is no longer the case.

With the advent of mobile devices, the sharing, and thus publishing of content, is moving steadily from the desktop to the mobile.

This move creates some challenges due to the constraints placed by the mobile platform. There are multiple kinds of mobile devices, each of them with its own capabilities and limitations.

In this thesis, different architectures are compared in order to find a proper design capable of providing an application environment conductive for mobile blogging. Such an environment carries a number of requirements on what it should facilitate. Some of these requirements are essential to enable publishing from the mobile, whereas other enhances the overall functionality of the service.

Two different prototypes have been implemented in order to evaluate the advantages and drawbacks of the most useful architectures.
# Table of Contents

1. Introduction .......................................................................................... 9

2. Background .......................................................................................... 12
   2.1. Blogging .......................................................................................... 12
   2.2. Classification .................................................................................. 12
      2.2.1. Traditional Approach .............................................................. 13
      2.2.2. Folksonomy ........................................................................... 13
      2.2.3. GeoTagging .......................................................................... 15
   2.3. Mobile Blogging ............................................................................ 16

3. Hardware Platforms .......................................................................... 19
   3.1. Blogging from the Desktop ............................................................. 19
   3.2. Blogging from the Mobile ............................................................... 20
      3.2.1. Mobile classification .............................................................. 20
      3.2.2. Physical characteristics ......................................................... 21
      3.2.3. Radio Link ............................................................................. 22
   3.3. Contrasting Mobile and Desktop ................................................... 24
   3.4. Opportunities in the Mobile ............................................................ 24

4. Device Software Architecture Alternatives ..................................... 26
   4.1. Native ............................................................................................. 26
   4.2. Runtime Environment ................................................................... 27
      4.2.1. Java ME .................................................................................. 27
      4.2.2. Other environments .............................................................. 28
   4.3. Web Application .......................................................................... 28
      4.3.1. Browser-based ....................................................................... 29
      4.3.2. Web Runtime Engines ........................................................... 29

5. Client Architecture Selection ......................................................... 31
   5.1. General requirements on mobile application .................................. 31
      5.1.1. Responsiveness ..................................................................... 31
      5.1.2. Footprint ............................................................................... 33
      5.1.3. Minimizing power consumption .......................................... 34
   5.2. Web publishing requirement .......................................................... 35
      5.2.1. UI implementation ................................................................. 35
5.2.2. Networking ............................................................................................................. 35
5.2.3. Accessing camera .................................................................................................. 35
5.2.4. Accessing gallery in file system ............................................................................ 36
5.2.5. Accessing location ................................................................................................ 37
5.3. Desirable Features .................................................................................................. 37
  5.3.1. Client Mashups .................................................................................................... 37
  5.3.2. Offline Solution .................................................................................................. 38
  5.3.3. Portability ............................................................................................................. 39
  5.3.4. Minimal development difficulty ......................................................................... 40
  5.3.5. Ease of deployment ............................................................................................ 42
5.4. The two candidates .................................................................................................. 42

6. Implementation ........................................................................................................... 44
  6.1. Design overview .................................................................................................... 44
  6.2. Core Structure ....................................................................................................... 44
    6.2.1. RESTful Architecture ...................................................................................... 44
    6.2.2. Protocol ............................................................................................................. 46
  6.3. Prototypes ................................................................................................................ 48
    6.3.1. Java ME Client .................................................................................................. 48
    6.3.2. RIA Client ....................................................................................................... 48

7. Evaluation .................................................................................................................. 51
  7.1. Responsiveness ..................................................................................................... 51
  7.2. Reusability ............................................................................................................. 52

8. Future work ................................................................................................................. 53
  8.1. Usability ................................................................................................................ 53
  8.2. Security Framework .............................................................................................. 53

9. Conclusions ................................................................................................................ 55

10. Works Cited ............................................................................................................... 57

Appendix I Use cases .................................................................................................. 59
Appendix II Abbreviations ............................................................................................ 64
# Table of Figures

- Figure 1 – Publishing from a mobile platform .................................................. 11
- Figure 2 – Mashup on the client ........................................................................ 38
- Figure 3 – Overall Architecture ...................................................................... 44
- Figure 4 – Server Overview .......................................................................... 45
- Figure 5 – Publishing Sequence ...................................................................... 47
- Figure 6 – Java ME Prototype Architecture .................................................... 48
- Figure 7 – RIA Prototype Architecture .............................................................. 50
Acknowledgments

The present thesis was carried out at Ericsson Research during spring and summer 2008. I would like to express my gratitude to my supervisor, Göran AP Eriksson, who provided instructive comments and guidance through every stage of the thesis process. Special thanks to Vladimir Katardjiev who worked with me on this thesis. His thesis, Mobile Browsing of Classified Content, bring another view of the problem from a reader perspective. Finally, I would like to thank Ericsson Research for providing the necessary resources.

Finally yet importantly, I would also like to thank my family and my girlfriend for their support and encouragement.
1. Introduction

Publishing refers to the process through which information become available to the public. By public is understood anyone, including the own publisher. Before the advent of computing and the Internet, publishing was confined to the creation and distribution of physical media. Traditional examples of published media are books, newspapers, music CDs and VHS movies.

Personal computers together with specialized software made easier for large publishers but also individuals to create their own digital publications, automating tedious tasks as well as saving time.

With the advent of the Internet and the web, it was easy for everyone to publish some content and make it available worldwide. Making content such as text, images and other data available via Internet and by means of the HTTP protocol is known as Web Publishing. At the beginning only the savvier, early adopter kind-of people were able to carry out the task of publishing. These people usually designed static websites that were updated regularly to add new content, or delete previous one. This method of publishing characterized by static ‘read-only’ is part of what years later has been coined as Web 1.0.

Gradually, accessing Internet at home became more and more popular, mainly due to broadband connection and fixed-rate tariffs. Consequently, a much wider group of people became computer-literate. This fact together with the emergence of new tools that made it easier publishing to the web, encouraged a higher number of people to publish their own content. The user role was gradually switching from being a mere reader to fully participate in the creation of content. This is one of the main characteristics of the Web 2.0.

Since the beginning of the WWW, different applications have held the privilege of being the most popular way of publishing. First off, Usenet made its appearance, allowing users to participate in a distributed discussion system worldwide. This system was conceived to let users post messages in groups with different topics on scope. Later a new application came into scene, the Internet forums or bulletin boards. These forums gathered communities of people with similar interests. Advanced users could create their own forum and gather people that were willing to generate new content on that area.

Eventually a new application turned up, the blog, a web site run by a single or several users that regularly create content. At the beginning, this content consisted of text entries on the front page that allow commentaries to be made by readers. These days, all sort of blogs exist with different focuses, text, photos, music or video.

Although it is possible for a user to install blog publishing software like wordpress on their web-hosting provider, many prefer to use specialized blog service providers, such as blogger or livejournal. Well-known blog providers are flickr for photo or youtube for video blogging respectively.
Historically, blogging has been an activity performed from the desktop. However, it is becoming more and more common to blog from a mobile phone. In the old days, mobile phones were much simple devices used mainly to hold conversations due to the network capabilities at that time. Later on, when 2G networks were deployed, data services became possible. SMS became a popular service, allowing people to communicate using short messages. Internet services became available shortly after, making possible to access web services such as blogs from a mobile phone. The act of blogging from a mobile phone is also known as mobile blogging or moblogging.

Due to the fact that a mobile phone is a very personal device, it is possible to provide context-based services that, for instance, leverage the user’s current location. A possible scenario would be a user that takes a photo, using his camera phone, that includes the current location as metadata; thus, providing additional value over standard desktop photoblogging.

Yet another reason why moblogging is increasing in recent years, stems from the dramatic improvement on the network infrastructure. Cell-based networks are now ubiquitous, and less and less areas lack of connectivity. Therefore, blogging from almost everywhere around the globe, at least in most developed countries, have become possible. Furthermore, 3G networks are now here to stay, leading to higher bandwidth capacities. Together with this 3G deployment, affordable fixed-price rate tariffs are now offered. Consequently, a mobile user should not be cautious anymore about the cost of the data transmitted. Consequently, a mobile user should not worry as much as before about getting an astronomical bill by the end of the month.

At present, mobile users have a wide choice of methods that enable them to make content available. It is possible to publish to the web, but also by MMS, SMS or e-mail.

This paper seeks giving an insight into different software platforms that enable publishing to the web from a mobile platform, specifically to a blog.

In order to better understand how the system works, in Figure 1 is visually represented the overall architecture of the system. The scenario would be as follows. A mobile user creates a blog entry and uploads it through the wireless network to the Internet until it reaches the web blogging service. The database located on the server will store the entry and metadata associated to it. Finally, other users either from a mobile device or from the desktop can download the uploaded content.
The content of this paper is structured in the following way. Section 2 - *Background* provides a view about blogging and mobile blogging, including previous approaches and state-of-the-art implementations. Section 3 - *Hardware Platforms* introduce both desktop and mobile platforms from a hardware perspective, contrasting and comparing both. Section 4 - *Architecture Alternatives*, presents the reader the different architecture designs that could enable mobile blogging. Section 5 - *Client Architecture Selection*, introduces the required characteristics of the system and how each of the environments fulfils them. Section 6 - *Implementation*, offers a view of the prototype mobile blogging system, comprising the back-end system, and the two different device prototypes. Section 7 - *Evaluation*, discuss about the different approaches taken to evaluate the different architectures. Section 8 - *Future Work*, introduces other architecture aspects that due to time constraints have not been discussed in depth. Section 9 - *Conclusions*, discusses the overall results of this paper.
2. Background

2.1. Blogging

A blog is a website where new entries are shown in reverse-chronological order. Consequently, blogging is the activity of contributing to a blog. A blog entry usually is a composition of text and media, such as photos and videos, together with commentaries from other users on the entry.

People’s motivations behind blogging are varied. Some people write some sort of on-line diary, where they document their lives and express their feelings. Even though it is a journal, bloggers usually don’t mind other people reading it. Other kind of bloggers add new entries to their blog aiming at their social network as audience. The newly created content serves the author to convey some information or events that are of interest to his friends. Finally, there are other kinds of bloggers that provide commentaries and opinions to different topics, reaching a specific community audience.

Blogs entries use primarily text. However, the use of photographs, also known as photoblogging, and videos is widely spread.

There are two different methods for publishing; the first one involves installing a content management system, CMS, like Wordpress on your own server; the second using one of the multiple free-available blogging services. Due to the know-how and time availability that setting up a CMS requires, most users choose to create a free account in one of the available blogging services.

From the very beginning, several blogging services drew more attention than others did, namely, Flickr has become the most relevant photoblog on the Internet; Blogger and Livejournal are among the most used textual blogs services; and YouTube is the big party on the field of videoblogging. However, it’s worth noting that most previously textual-based blogs, now let users publish any kind of media as well, either videos, photos or audio.

At the beginning all this information was completely unorganized. As more and more entries were written, those that previously were on the front page were pushed into the background. Therefore, it became so difficult to retrieve old posts since they were only order chronologically.

Clearly, it was essential to organize this information to make it easily to retrieve in future look-ups.

2.2. Classification

Due to the vast amounts of data that are published on the web every day, it is necessary a method to sort all this information out in a convenient way. It actually boils down to putting the different pieces of data into different overlapped groups or categories, with a retrieval purpose in mind. The actual process of classifying the data
can be made either automatically, in case of a machine applying the categories, or manually, if some people (experts) are involved in the classification.

2.2.1. Traditional Approach

The classical categorization, also known as taxonomy, groups different objects or entities based on the properties they all share.

In a taxonomy system, the elements are organized in a hierarchical way. First off, a bunch of different general categories are defined at the higher level. These top categories may hold other subcategories, with which they share their own constraints or properties. By and large, these (sub)categories are defined by a board of experts.

An element or entity may be a child of several different and unrelated categories. For instance, apple will at least belong both to the fruit category and the computer manufacturers’. It is worth noting that these categories are mutually exclusive and every new object should be a member of either one or several categories in the hierarchy.

2.2.2. Folksonomy

Folksonomy is the practice of annotating and, therefore, categorizing content in a collaborative way. It has emerged as a superior way for classifying content than taxonomy. It delegates the task of annotating content to the very same publishers and consumers of content, instead of having a board of experts classifying the content in a hierarchical structure. In a folksonomy the keyword applied to every piece of content, is also known as tag. Each one of these tags is a category in itself. All tags are at the same level, and thus, a hierarchy does not exist.

In a folksonomy-based classification system, users are allowed to tag their own media, but also other people media, with words chosen at their own will. Some of these systems aggregate these tags or keywords in order to build a tag recommendation system. In a way, they leverage the collective intelligence of the very same users to improve the system. The recommendation engine will rank the tags, and provide a sort of controlled vocabulary for new users tagging, what the system infers is, related content. The approach used by these systems or web services is suggesting the more relevant tags to the user's specific media, in order to make it more likely to everyone to find this content if those tags are applied.

Most folksonomy systems allow users to apply the tags provided by the recommendation engine but also user-created tags. The latter entail some risks stemming from the freely chosen tags policy (2). Some tags may convey two different concepts, for instance, turkey can mean either the country or the animal. A related challenge is caused by those words with different but related meanings, such as milk, which could mean either the liquid or the act of getting the milk from the cow. Yet another challenge come from different words that imply the same concept such as apple and Mac; this entails that a user will only get those results associated to the
word he looked for and not the other. Lately, new systems have tried to federate those words with similar semantics \(^1\).

Finally, since a controlled vocabulary is not enforced, taggers may misspell words, use the plural of a word, or tags not related to the content. There are other situations where tags are so specific for a user in a context that bears no meaning for other people to use it. No harm is done, since the systems will promote those tags more often used or relevant tags in their tag recommendation system. Thus, those irrelevant tags will fall into oblivion.

Folksonomy or collaborative tagging has been the topic of numerous papers. Their focus ranges from user motivations for tagging to tag recommendation algorithms

Sigurbörnsson and van Zwol on their paper (3) introduce an algorithm that suggests tags to annotate a photo based on previously added user-defined tags. The algorithm first finds a set of tags that frequently appear at the same time as the user-defined tags, that is, it takes into account the number of times where both the user-defined tag and the candidate tag co-occur. Second, it merges the candidate tags that co-occur with every user-defined tag in a single ranking. Finally, dampening measures are taken in order to promote those tags that are frequently used, but are not too generic such as wedding or the number of the year (2008).

In recent years, several web-based applications have started to support user-defined tagging to enhance the overall functionality of the service. Mainly, the primary goal for this kind of tagging has been the benefits it provides when searching for already uploaded media on the site. Previously, this media was organized either by using a known set of categories, author or date. This approach was not very optimal and people found difficulties in retrieving data they were interested in. Folksonomy or collaborative tagging has become the preferred way to organize all this data.

Among the pioneers mainstream services using folksonomy are Flickr, del.icio.us and Technorati. Every one of them implements collaborative tagging in a unique distinct way (4). In del.icio.us users can tag their own bookmarks as well as other users’ bookmarks, and the user actually get some suggestions to make the tagging more meaningful. Meanwhile in Flickr the users cannot tag global images, only those they or someone in the social circle owns. This occurs as a result of photos being contributed by users in contrast to bookmarks, which are global resources.

The motivations behind this social tagging activity are not evident. People annotate content both for personal and social purposes (5). They mainly tag for personal reasons, for instance, to be able recall the context of the content later on. Tagging also helps to organize the content and therefore its retrieval. Searching for a keyword that has been used for tagging some media, make the retrieval of the content a trivial task.

As a side effect, by tagging your own content with relevant keywords, other people would be able to reach it. This way you may help them to find your content.

\(^1\) Del.icio.us – http://delicious.com
but, at the same time, communicate the context of a photo or blog entry too. Motivations differ from service to service. In del.icio.us users tag mainly for retrieval or to keep bookmarks organized. Flickr users besides retrieval are usually eager to communicate the context or their own views.

### 2.2.3. GeoTagging

The range of opportunities derived from having geographical data attached to media, has supposed a boost to the development of web services that make use of this data. Geotags are metadata representing the geographical coordinates of the content, expressed as latitude and longitude. However, other metadata such as bearing, altitude, zip codes or place names may be used as geotags. This data, which usually is automatically gathered, must be fetched in the very moment a publisher create the media to be consistent with the location.

In order to tag a photo with its geographical position an external system is required. At present, GPS and network-based location systems are the most ubiquitous ones. Especially now, built-in GPS mobile devices are making their way into the market. Due to the pervasiveness of wireless networks, geotagging and media publishing at the right spot, becomes possible. Furthermore, geotagging can be performed transparently, without user interaction.

Geotagging shows one challenge in itself. For instance, if you find yourself taking a photo of a summit in The Pyrenees, it is likely you will be some kilometres away. This implies the location coordinates attached to the media will not be strictly accurate. The solution proposed by most geotag-enabled services involves retrieving media within a distance from the actual coordinates sought by the media seeker.

Bearing in mind this limitation, geotagging becomes a great tool that leverages the retrieval of media within an area, given some geographical coordinates.

As for data formats used for geotagging, a group of them have made their way into the scene. These days some high-end digital cameras come with a built-in gps receiver. As soon as a user takes a photo, the geographical data is written into the image file. Actually, it is stored in the Exif\(^2\) header as geotags. Concerning HTML documents, a microformat has been developed that use html tags to include the position data. The geo microformat is a stable specification that make easy for tools to extract the geographic information. It uses the value geo in an attribute class of a span or div element to point out where geodata is located. For instance, the following line enclosed in other data, could indicate the information is related to those coordinates.

```
<span class="geo">41.657523; -0.876589</span>
```

\(^2\) Specification that allows adding metadata to an image file format
There are systems that are built upon an ontology that uses RDF to describe the concepts and relations between them. The W3C made public a basic set of vocabulary that provides a namespace, called “geo”, for geographic information. The same data as above but in RDF will look like the following.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"><geo:Point>
  <geo:lat>41.657523</geo:lat>
  <geo:long>-0.876589</geo:long>
</geo:Point></rdf:RDF>
```

Some applications have started to use geotagging data to improve their services. *Flickr* and *Panoramio* are good examples of this usage. Both services feature a parser that extract the geo coordinates from the Exif header, and use them to tag the photo. *Flickr* is a photoblog, that is, it shows the photo information, and allows other users to comment on them. In contrast, *Panoramio* is focused on showing geotagged pictures placed on a map. Both web applications also allow users to add the metadata later on, once the photo is already on their server.

### 2.3. Mobile Blogging

Mobile blogging is the activity that involves blogging from a mobile device. The word “mobile” is ambiguous enough to be understood as any kind of portable device. However, for this paper, by mobile I will refer to those devices that have the capability of connecting to the public wireless network.

Mobile blogging or moblog has seen its use increase steadily during the last couple of years. Two aspects have contributed to this spread:

- The proliferation of camera phones (more than 90% of the total mobile phones sold).³
- The availability of flat rate pricing for UMTS data access to the Internet.

In the whole world, there are more than 3.3 billion mobile phones⁴ by the end of 2008; and 1.4 billion has a camera built-in. Mobile phones outnumber the PCs 3 to 1

---
³ Cellular News - Camera Phones to Represent 90% of Western European Mobile Phone Shipments http://www.cellular-news.com/story/14320.php
worldwide. In developed countries, a PC can be found in fifty percent of the households. However, in most developing countries, people don’t have a personal computer available to access the Internet, and instead the mobile becomes the prevalent method to access the Internet. For instance, in India\textsuperscript{5} 3 out of 4 people access the web from a mobile.

Since tags are a core feature in blogging, several mobile applications have come up with different solutions for this purpose. Tagging is a tricky task in a mobile phone due to the small size of both screen and keys. Therefore, automated tagging becomes necessary, if people are to be encouraged to annotate their content. A suggestion-based tagging system is a challenge in itself. Since a mobile screen size is generally small, few tags can be shown at the same time. Thus, one of the main goals of these systems is to define an algorithm that presents the most relevant tags to the user in a context-aware scenario.

In order to be able to publish from a mobile device, two approaches have been in use so far. Flickr allows users to send photos in the shape of an e-mail, using the subject as the title of the photo and the body as the description. It even let you attach tags by using special codes.

The other way to send tagged content is by means of a rich mobile client. Rich mobile clients are much more powerful than just plain e-mail. ZoneTag (6) is an application that among other things let you upload a photo in just two key presses. Location data are automatically included. User-defined tags may be added easily. A search component in the interface provides auto-complete functionality that uses those tags previously selected by the user.

ZoneTag focuses are making input as easy as possible (especially cumbersome in the mobile), support for tag suggestions, and providing a way for users to avoid tagging all along.

Tags are prefetched in advance based on the current context of the user and cached on the mobile. Whenever the user wants to upload a photo, a tag option is shown on the menu. If the user chooses to do so, a list of tags is shown on the screen divided by categories. Tags in each one of the categories reflect the source of the tag, e.g. Recent, to indicate tags used during the last 24 hours or Local, to show tags used by user’s social network members.

It is worth mentioning that ZoneTag build up part of the user context knowledge such as the social network from the user profile data in Flickr.

 ZoneTag is a custom-made approach to upload and tag new photos to Flickr. A user is not able to upload other kinds of media, such as blog entries or even choose where to, other than Flickr. Another approach is needed to offer the users these functionalities. Later in the challenges section it will be shown how to let other developers to enhance the publishing experience atop our platform.

\textsuperscript{5} The Economic Indian Times - Indians prefer to surf Net on the go
http://economictimes.indiatimes.com/Indians_prefer_to_surf_Net_on_the_go/articleshow/2183516.cms
ShoZu\(^6\) also use two different approaches to upload media to the web. ShoZu MMS make use of the MMS service to upload content to a specific e-mail address. Of course, this method limits the functionality of the service to just the upload of the image, neither tagging nor location metadata is added to the photo.

Nevertheless, ShoZu also provides a rich client quite similar to ZoneTag in functionality, but among other things has fewer restrictions in terms of supported services. It actually allows one to send new content to other online services such as Facebook or Blogger. Anyhow, every new service supported must be hard coded into the client. Thus, as it happens with ZoneTag does not provide an easy way for adding new services or any other functionality that has not been imagined yet.

One must point out that this is the state-of-the-art at the time this paper was written, and so it should be expected for both clients to add more functionality as time goes by.

Apart from these third party solutions, mobile phone manufacturers are developing their own publishing solutions. Sony Ericsson recently released two mobile phones\(^7\) with this functionality built-in. It allows users that own an account in the Blogger service to upload photos to their blog. In a few clicks, a user can take the photo, write some text and publish it immediately to the web. These mobile phones feature an A-GPS device built-in that makes it possible to geotag the pictures taken. It adds the geo data to the image file as part of the Exif header. This functionality enables it to show the photos in the location where they were taken over a Google map.

\(^6\) http://www.shozu.com

\(^7\) C702 and C709 models
3. Hardware Platforms

3.1. Blogging from the Desktop

By desktop platform is understood a personal computer that usually stands still at one place. Due to its dimensions, it is not designed to be portable. For convenience, a laptop is considered as a desktop computer due to having the same constraints and providing the same user experience as a desktop computer. A laptop can be easily moved between locations, but it is not as portable as a mobile handset. A laptop requires the user to be still while interacting with the computer, whereas a mobile user can be on the move. In addition, both laptop and desktop computers feature the same user interaction devices, that is, keyboard and mouse/touchpad for inputting data, and a big screen for output.

The desktop was the first platform for blogging, meaning that most users of mobile phones are already used to the desktop blogging experience. This obviously doesn't hold in most developing countries, where, as stated previously, the mobile phones are prevalent over the PCs.

There are different characteristics that help us define what a desktop hardware platform looks like. These are the input/output methods, the capabilities and the network link.

Historically, two input devices have been the standard on the desktop: the keyboard and a pointing device. The keyboard provides a fast way to input different letters or symbols in the computer, just by pressing one or several keys. Most common keys, such as the alphabet letters, the numbers or punctuation characters are inputted by pressing exactly one key. Likewise, other keys are mapped to common actions such as scrolling or editing text. Having around 100 different keys on the keyboard, make it possible to input most of the common symbols in one stroke, thus, saving time. However, typing fast requires training the finger muscles to remember the position of the keys. Once this training is done, typing becomes a much faster task than handwriting.

The pointing device enables to input spatial information into the computer. In a graphical user interface, different windows, icon or menus are shown in the screen, along with a cursor. The pointing device controls the position of the cursor. The user can move the cursor over some of the visual widgets to interact with the desktop environment. Different actions are mapped to the different widget elements. The events associated to these elements are triggered whenever a user clicks on them. The pointing device is usually a mouse on the PC, and a touchpad on the laptop. Both share the same interaction paradigm.

As important as inputting data into the computer is getting the response. The output device is the screen. Once is processed, data is shown on the screen as more or less bright pixels of different colours. Current popular screen resolutions are 1024x768 and 1280x1024.
According to Moore’s Law the transistors in a CPU chip double every two years. Current desktop computers come with 2 GB of RAM and 500 GB of hard disk. Blogging is not a very demanding resources task, and by means of a browser, it can be easily performed. A browser like Firefox \(^8\) recommends having a Pentium 500 MHz processor, 64 MB RAM and 52 MB of hard drive space. This actually means that most computers sold during the last eight years can fully perform blogging. In other words, most of the installed desktop base can be used for web publishing.

Desktop computers have a power supply that provides constantly energy to the system. Then high power consumption is not a problem beyond the electric bill getting higher.

The last characteristic is the network link. The desktop is characterized by having fixed connectivity. A desktop computer usually accesses the web via ADSL or Cable Modem. Regardless of the actual connection to the modem being via Ethernet or WIFI, this type of connection has some particular attributes. It has more than an acceptable latency for browsing, usually less than what a user tolerates. The throughput is usually acceptable, with an average of 1MB/s or over. Still, blogging itself doesn’t require that much bandwidth. The last attribute of the fixed connectivity is its reliability. An ADSL or Cable connection is up most of the time.

Due to all previous characteristics, one can perfectly assume that a desktop user would be most of the time comfortably sitting in front of the computer. Therefore, he can perfectly carry out some other task on the computer while waiting for a previous task to complete.

These were the characteristics of the first platform used for blogging, that is, the desktop. However, the mobile is at present another, but different platform, that can enable blogging.

### 3.2. Blogging from the Mobile

#### 3.2.1. Mobile classification

By mobile is understood any portable device that provide both data and voice communication through a wireless network of base stations.

There are actually different types of mobile phones, ranging from high-end smartphones to mid-range and low-end devices. Smartphones are characterized for having more PC-like features than mid and low-end handsets. Advanced software applications such as word processing are common. Furthermore, a fully-fledged operating system is commonly embedded on these phones. These operating systems allow developers to build applications on top of it that can access low-level device capabilities contrary to some mid and more specifically low-end phones. In these other phones, applications are only allowed running inside a runtime environment.

---

However, the line that divides smartphones from the rest of the phones is blurred. Devices that some years ago were known as “smartphones”\(^9\) could be standard mid-range at present.

As with the desktop, on the one hand a mobile platform is defined by the physical features of the handset. On the other, the connectivity, in this case provided by the radio link is also a major characteristic of the platform.

### 3.2.2. Physical characteristics

Several input methods help the user to interact with the mobile device. In the mid-range and low-end phones, the twelve-button keypad with several other soft keys is the standard. A soft key is a button that provides no fixed functionality by default, and then several different functions can be mapped onto it. The keypad consists of the 12 numerical keys that are also found in a regular telephone. From number two to number 9, three or four letters are assigned to each key according to the ITU-T E.161\(^{10}\) recommendation. This actually means that mobile users must press 1 to 4 times the same key in order to input the desired letter. To be fair, most phones incorporate a predictive text technology system, thus allowing entering words using a single stroke for each letter. Apart from the keypad, every phone features five different soft keys that have different mapped function depending on the application. The central one is usually a cursor key that can be seen as a metaphor of the arrow keys of a standard keyboard. These arrow keys allow the mobile user to move the focus up and down, left and right.

Although the cursor key allows the user to move between the different menu options, is impossible to point to a specific part of the screen. Another issue is the tiny size of the keys. One key on a mobile keypad is roughly one quarter of the size of an actual keyboard key size. Pressing one key instead of the desired one happen more than often, preventing the user from fast inputting data. Yet another characteristic of the keypad is that keys are pressed using one or both thumbs, since the rest of the fingers are holding the phone itself.

Smartphones usually a bit more handy input interfaces. Some of them have small QWERTY keyboards that allow users to input letters pressing just one key. Yet, keys are so tiny that users cannot use the typing technique to input data rapidly on their mobile.

Some smartphones also feature another type of input device, the touch screen. It let developers provide a different user interaction experience. In this case, no cursor is located on the screen. However, the fingertip is used as a pointing device, and then a much closer to desktop user experience is possible.

---


\(^{10}\) E.161 : Arrangement of digits, letters and symbols on telephones and other devices that can be used for gaining access to a telephone network. http://www.itu.int/rec/T-REC-E.161-200102-I/en
The output interface in a mobile phone is the LCD screen. What characterizes a mobile phone screen is its miniature size. Many different mobile phone manufacturers exist, every one of them having a vast line of mobile phone models. Therefore, there are multiple screen sizes ranging from 176 x 220 to the 480 x 320 pixels size of the Apple iPhone, the average size being 320 x 240 pixels. In the tiniest screens, no more than five lines of text are recommended to show at the same time.

The processing power of mobile phones greatly varies between models, ranging from less than 100MHz to 620MHz (iPhone) ARM processors. Clearly low-end and mid-range devices don’t run at speeds above 200MHz. Therefore, processing power may easily come as a constraint when several processes are running in the background.

Memory capacity is also a variable that differ quite a lot between handsets. It ranges from a few MB of shared memory to 128MB (iPhone). Low-end and mid-range phones usually don’t have more than 50 MB of memory to share among all running processes.

A common feature of every mobile phone is having a battery that must be recharged every now and then. Due to the miniature sizes of the mobile devices, batteries must be also kept small. Battery capacities are improving with every new model, but still it doesn’t keep pace with the power needs of the applications that make full use of the device resources. Several are the factors that cause this battery drain in applications that enable blogging. Namely, these are the transmission of data packets through the radio link, the camera and the GPS device.

New devices capable of connecting to 3G networks are suddenly getting less hours of battery than those using GSM networks. 3G networks provide better bandwidth thanks to the W-CDMA channel access method, but at a cost. In CDMA, the entire signal must be processed, leading to 3G networks having higher power demands.

Camera built-in on phones is also a source of high power consumption. Most modern ones feature an extra flash unit that drains the battery even more. Yet, is not as problematic as the other power drainers since can be off and be used just when needed.

Finally, GPS devices must keep permanent communication with the satellites, causing the battery to drain fast. Otherwise, it will take several minutes to align itself with the satellite. GPS receiver consumption can be expected to decrease over time. However, reaching levels where the impact on battery time is unnoticeable for the user (~1mA) is not expected to be reached on the near future.

\subsection*{3.2.3. Radio Link}

The last characteristic of the mobile platform is its network link, that is, the wireless radio link.

Even though UMTS networks have been deployed over the last few years, the availability of the network is different in every country and even in different areas of the same country. Big cities have been among the first to get UMTS access while vast
areas are still just covered by GSM cell towers. Even more constrained city areas have at this moment access to HSDPA-enabled base stations that deliver a maximum theoretical downlink speed of 14.4 Mbps.

This leads to a situation where those people that are owners of a brand new 3G mobile phone, are getting currently a more realistic downlink speed of up to 384kbps at most. The event of mobile broadband drives fixed rate charging schemas, where user can send data at fixed cost (up to a certain limit). All those users that are not willing to spend their money on a 3G-enabled phone, still can send and receive traffic from the Internet using a GPRS network. GPRS provides packet-switched service on top of the GSM network. The theoretical bandwidth that GPRS networks may provide is between 56 and 114 kbps. Usually it translates into a maximum 48 kbps. Just like the throughput old analogical modems deliver, which was the usual way for individuals to connect to Internet one decade ago.

The average low bandwidth available constrains the amount of data that should be sent, and thus one should be cautious when deciding which data is strictly necessary to transmit and in which format. To ensure that no information is duplicated, a policy involving cache should be taking into account.

In a ubiquitous and changing context like this, coverage becomes one fact that may force you to redesign from the ground a whole application. The coverage a mobile terminal has at a specific moment is inversely proportional to the distance that separates it from the cell mast. This translates into higher latency and lower bandwidth the further we are located from the nearest mast.

Besides, there are some situations where there is no coverage whatsoever. It often happens in sparsely populated areas, in the countryside, or inside some buildings, where the waves cannot go through beyond some walls. There are areas that due to the mountainous geography, are difficult to cover. Furthermore, sometimes the carriers for economic reasons do not deploy their networks in low-density areas, unless they are given some incentive.

Some measures should be taken to allow publishing from places where the coverage is almost not existent.

Due to the limited capacity of Base Stations, only a maximum number of phone calls can be established at once. If one of these base stations gets overwhelmed, it is more than possible for the traffic to be blocked for a while. This can easily occur in a massively attended event such as a football match or a concert or even in a situation where many people want to establish a connection at the same time like in New Year’s Eve or in a traffic jam. Since there are a finite number of channels available in every mast, some users will realize that the phone shows a strong signal coming from the mast; nevertheless, no traffic will be transmitted until one of the channels, also called slot, is released. This is a characteristic of the Time-Division Multiplexing (TDM) used in the GSM telephony system.

Carriers carry out studies that show how much is the average working load of every one of the cells. They provide enough resources to satisfy the demand for most of the time. This doesn’t imply that it shouldn’t be taking into account, since there are sometimes the blocking happens. The user will first complain about the application,
because it wasn’t able to publish a photo of their friends, rather than about the carrier, for instance, in case the cell was flooded. A blocking situation yields the same effects as no coverage at all, and some similar measures should be provided.

 Transmitting data by means of either a GPS or UTMS cellular network come with a significant cost. Eventually more and more people will have a flat rate UMTS tariff, but currently the cost that the use of a specific service may yield, is an entry barrier that few people may afford to overcome.

 For instance, currently in Spain the average UMTS flat rate costs 39€ a month, and when the bandwidth used exceed 1 GB the speed is limited to a maximum of 128 kbps. Another common non-flat rate, is 1 euro per day used, limited to 10MB. The speed delivered will depend on the coverage of the area. If one happened to be in an area where HDSPA has been deployed, a theoretical maximum of 3.6Mbps is delivered, 384Kbps for UMTS cells, and 64 Kbps for GPRS accesses.\textsuperscript{11}

 The cost associated with the traffic of data should be an incentive to reduce the bulk of data passed to and from the network. Similarly, a cache policy that takes into account the actual resources of the mobile would allow keeping data often used by the application. Otherwise, the user could face a huge bill at the end of the month.

 A typical mobile user is an individual that may have time constraints or being in motion. The mobile user is not a patient user willing to wait for a response. For instance, a person that takes out her mobile during the trip between two bus stops to make a photoblog entry.

### 3.3. Contrasting Mobile and Desktop

 Every aspect of the hardware platforms that support mobile and desktop blogging has being described. In this section, both platforms will be compared.

 The desktop interaction paradigm differs in very many ways from the mobile. To input data into the mobile just the tiny keypad is used in contrast to the keyboard and the mouse or pointing device. Clearly, the input method is much more limited in the mobile. The lack of a pointing device in the mobile means that is not possible to point to any area of the screen the user wish. In order to trigger an action, the user will have to move the focus to the visual element responsible to perform the action.

 The user interface design in the mobile should reduce the user movements to a minimum.

### 3.4. Opportunities in the Mobile

 Then, applications on mobile phones may be tailored to take into account the user data and context. An application sensitive to context could show information to the user according to his present location. In blogging scenarios, it can mean that location

\textsuperscript{11} http://docs.google.com/View?docid=dfmf2xcn_9ws3mwc. Up-to-date list of mobile data rates.
metadata could be automatically included, adding value to the service. In the desktop, this is simply impossible. Clearly, context represents an edge over the desktop.
4. Device Software Architecture Alternatives

There are hundreds of different mobile devices in the market, every one of them allow developers to create applications in one or other way. There are devices that make it possible to create applications directly on top of the operating system, whereas others only allow running applications inside a runtime environment. Most of them provide access to the web, and therefore applications can be download and executed on the browser.

4.1. Native

Native applications are those that run directly over the operating system, often implemented in C or C++ using a proprietary real-time operating system. These kind of applications make use of the API the OS exposes to access its natives resources.

In the mobile operating system market, several competitors allow the deployment of native apps on top of them. There are no less than four major OS in the market including Symbian, Windows Mobile, iPhone OS, and the most recent Linux-based Android.

Symbian is an OS currently owned by Nokia, which has several platforms based on it, the main ones being UIQ and S60.

Windows Mobile, which is the Windows OS tailored for PDAs and smartphones.

Android, is an open platform based on the Linux OS that includes also middleware that makes easier to develop apps that run on a broad range of devices. Apart from Android, there are other Linux-based OS platforms already on the market.

In the desktop, this challenge also exists but it is mainly constrained to three platforms, Linux, Mac OS X, and Windows. This last OS have more than 90% of the market share. Thousands of developers release their software just in this platform, and still reach most of their audience. However, in the mobile platform a more balanced distribution of the market. Symbian has the greatest percentage of smartphones sales in Europe. In China, Linux and Symbian share most of the market. On the contrary, Symbian has no relevance in the US, whereas Microsoft, Apple and Blackberry are the major players. Therefore, one should bear in mind, which market and audience are on the scope when developing a mobile app.

---

12 According to the market share survey conducted by Net Applications


4.2. Runtime Environment

The second approach involves developing an app that is executed in a certain runtime engine. A runtime environment is a normal application that runs inside an OS as a single process, but provides software services to other programs, therefore behaving as a “virtual machine”. Its purpose is creating a platform independent programming environment. The underlying operating system and hardware is abstracted, allowing executing the same application in every platform where the runtime environment is available. Instead of issuing commands to the operating system, these are sent to the virtual machine, using the runtime engine API.

4.2.1. Java ME

Java ME is a runtime engine or environment based on the original Java platform tailored for small and resource-limited devices such as mobile phones or PDAs. It comprises the virtual machine, known as KVM, where Java programs are run and a set of API libraries. It doesn't offer the complete library set of the Java Standard Platform, but just a small subset of classes. Due to the limited memory available on handsets, just those key APIs are included.

In the mobile platform, the vast majority of mobile device ranging from high-end to low-end handsets includes Java ME by default. Some manufacturers in the so-called smartphones niche also include it. However, the set of APIs installed differ from device to device. Sony Ericsson's devices are classified according to the libraries included in order to help developers. These range from JP-1 to JP-8 now. A specific device must include the libraries that correspond to that label to be classified in that category.

One of the benefits of developing an application in the Java ME platform is that it can run in the largest set of devices. The drawback comes from the fact that this is far from reality. Not every single device has the same characteristics. Some have a bigger screen, some a tiny screen. Some of them use a touch screen as the input device, whereas most of them use the keypad and soft keys to interact with the handset. Bluetooth and camera may be built-in in most devices, but GPS is included in just a few of them. Finally, as stated above, some of them have a more complete and advanced set of libraries and others just the basic ones.

A program that runs over the Java runtime environment is also known as a MIDlet. Bearing in mind that developers that are skilful in Java outnumber by far those that program in C/C++, most of them will prefer moving to Java ME than programming an application that runs directly on top of the OS. Developers can take advantage of their previous knowledge of Java on the desktop and apply the same patterns on the mobile. As stated previously, the portability could be one of the main benefits of a runtime environment, but also the additional layer of security provided by the JVM can be seen as a benefit. The environment handles the access to the hardware and networking resources, thus reducing risks. On the other hand, as a trade-off of having an additional component, the program execution is slower than in native applications. In contrast to
a browser-based web application, a MIDlet can have the business logic built-in and work in a situation where the network resource is not available.

**4.2.2. Other environments**

Another example of a runtime environment approach is Flash Lite, Adobe’s lightweight version of Flash Player to mobile devices. It allows developers to create programs quickly using Adobe’s proprietary authoring tools. Like Java ME, there is a wide group of developers already making Flash content on the desktop that willingly migrate their applications to the mobile. In a similar way, developers will have to focus on a specific Flash Lite plug-in or make their applications backward compatible with previous versions. One drawback is that by far much less devices support Flash in contrast to Java ME. This is steadily changing since the major mobile manufacturers are committing themselves to bring more Flash-enabled handsets into the market. However, by using Flash developers surrender their ability to access some device capabilities, which are accessible to Java ME.

A new system that combines Java ME and Flash to create a better experience has been unveiled by Sony Ericsson. It aims at encouraging developers to create the user interface and multimedia content in Flash, whereas the business logic will remain in Java ME. The Capuchin API will be responsible for exchanging data from the presentation layer to the backend.

**4.3. Web Application**

With the advent of the Web 2.0, the World Wide Web has turned into an application platform in itself. The programs that are built into this web platform are called web applications. They have in common that are coded using open well-known web technologies. However, there are two main types of web applications, those rendered by a standard web browser and those running on top of other runtime environments.

The web technologies the applications use are HTML, CSS, JavaScript and the DOM API.

HTML is a markup language that is used to describe the overall structure of a web document. CSS, on the other hand, is a style sheet language used to describe the presentation details of the HTML document. The content of the document is then separated from its presentation.

JavaScript is a script language that is widely used in web environments. It is mainly used to interact with the DOM API.

---

14Capuchin project
https://developer.sonyericsson.com/site/global/newsandevents/latestnews/newsapr08/p_project_capuchin_announcement.jsp
A Rich Internet Application or RIA (7) is a web application that relies on an underlying engine to render and execute it. They are based on technologies that provide asynchronous client/server communication, improved usability and portability. These technologies differ from RIA to RIA. Some of them rely on open web standards to achieve its goal. Others rely on proprietary or a mix between proprietary and web standards. The most popular ones, on the last set, being Adobe Flex and Silverlight. The processing of the user interface is done on the client in contrast to traditional web applications. This is actually translated into greater user responsiveness. The user experiences an interaction with the web application closer to that of native desktop applications. To achieve this effect, asynchronous communication with the server is used. Ajax (8) is the name that groups the different technologies that allow sending or retrieving data from the web server on the background. The user will be able to interact with the user interface in the meanwhile. Most browsers use the XMLHttpRequest (XHR) object with this purpose.

RIA applications can run either inside a browser or inside a web runtime engine.

### 4.3.1. Browser-based

A browser is a piece of software that communicates with a web server using the HTTP protocol to fetch some web pages and display them on the device’s screen. These web pages are built over a combination of web standards such as HTML, CSS or JavaScript, and other content files such as images.

Current web browsers support RIA applications using Ajax. In the mobile niche, previous browsers only supported formats specially tailored for mobiles, such as WAP 2.0 or WML. However, newer browsers such Opera Mobile or those based on the WebKit layout engine are capable of rendering the “full” web. However, these mobile browsers, including Webkit-based browsers for Android, iPhone and Symbian S60, are just available for smartphones.

Opera offers another browser, Opera Mini, which is built upon the Java ME platform. The web pages are first processed by a proxy server in order to compress them and make browsing faster.

What all these browsers have in common is that they are unable to access the underlying hardware capabilities. The lack of accessibility makes impossible to take a photo from the camera for instance, and thus photoblogging.

### 4.3.2. Web Runtime Engines

The other environments where web applications can be executed are dedicated runtime engines. There are several types of runtime engines or virtual machines, as the Java ME platform represents. What is understood by web runtime engines are those that leverages open web technologies to build applications. The basis of this kind of applications is HTML+CSS to structure and define the style of the user interface, and

---

JavaScript for the logic. In contrast to web browsers, the Web Runtime Engine (WRT) allow web applications to access the hardware abstraction layer.
5. Client Architecture Selection

The application architecture in the user device can be structure in different ways depending on the requirements. In the case of publishing content to web services such as blogging, the requirements can be divided into two categories; one general to mobile application and another specific publishing to Web services like geo-tagged enabled blogging.

One architectural question is how much of the application that should be put into the user device and how much should be implemented in the network in a back-end server.

On the client side, the key architectural questions are whether to implement all in Java or parts in JavaScript/HTML, and if so, how much of the application. As stated before, both Java ME and JavaScript technologies have some pros and cons. In this section the different requirements that a mobile blogging application must fulfil are presented. The goal is to find a trade-off between the performance that a more close-to-the-machine JavaME approach will deliver, and the ease of development and portability inherently of the web application approach.

5.1. General requirements on mobile application

5.1.1. Responsiveness

Responsiveness is a property of a system that defines how quickly it responds to user input. The problem with a non-responsive system is that the user usually gets frustrated after waiting a specific amount of time for a response. A long delay from the input action to the response may convey the wrong impression that the application has crashed or that the command was not handled.

User’s first instinct will be stopping the application or sending new commands, thus, worsen the situation. Even if the user sense that the application is behaving correctly, this delay can be enough reason to stop using the application permanently. Around 100ms is the limit of time a user sense the application response is immediate. If the delay is at most one second, the user feels the system is working on the task, although the response it is not instantaneous. Although the user can focus her attention for as much as 10 seconds, usually gets frustrated (8). Usually half a second delay is considered a great for web applications.

Furthermore, as opposed to a desktop user, a mobile user may not have that much time to wait for the response. There is a high chance the user is on the move, and his attention cannot be focused on the mobile screen.
Due to the latency and in some cases poor throughput of the radio link, this lack of responsiveness is inherent to the platform. In the context of tagging an image with the position or a user-defined tag, one of the events that could trigger this poor responsiveness is the tag recommendation system. Tags are a vital part of the blogging service. In order to provide the blogger with the best tags according to his context, the system should search and fetch the recommended ones from the server and show them to the user. This procedure also includes fetching the position and the map that surrounds that position, so the user can check how accurate that position is. These actions include interactions with a network server over the wireless access.

Now, as described above, the characteristics of the wireless access may impose delay, leading to requirements on the speed of the client execution environment, in this case either the JVM or the DOM/HTML and JavaScript engines. In the later evaluation, it is assumed that the overall time for tagging an image and uploading the metadata tagged image should stay within 1 second in order to make the user sense the response is immediate.

On the one hand, is important to know if the UI rendering is good enough to provide a good responsiveness in both the Java ME and the RIA solution. On the other, if the latencies and therefore, the poor responsiveness created by the network link, can be overcome using both technologies.

In the RIA approach, the decoupling of the user input commands from the HTTP synchronous connections, helps to reduce the delay before the user get the response. The use of JavaScript to modify the DOM model enable a more responsive interface, since only some of the graphics element are rendered.

In order to overcome the network link delay, some common solutions could provide better responsiveness: asynchronous communication, prefetching and caching data.

In general, to get a good UI responsiveness is necessary to give the higher running privileges to the user interaction controls. If a process must do some computation, the user should still be allowed to interact with the user interface. Performance issues like these ones may arise since mobile devices have a low processing power. Tasks like parsing XML data can be computationally expensive and could potentially block the UI if it is not done in a different thread.

Most of the needed communication with the back-end server should be handled asynchronously. Those triggered by a user command should not block the UI, but instead give back the control to the user until the response is received and shown.

As for data such as tag, location and maps, there’s no need to force the users to issue a command to fetch them. Since these data depend on a specific context that takes into account the user position, this data can be retrieved as soon as the application notice the user wants to publish something or even better at certain intervals. These intervals could be based on a period or be triggered as soon as the user moves from one base station to another. Then, one solution to solve this problem is asynchronously prefetching the tags during those intervals.
Those tags must be not only prefetched but also stored for future uses. Bearing in mind the limited amount of memory in regular mobile phones, the available cache space will only allow saving some of the tags. A highly optimized tag recommendation algorithm is needed to sort and rank them, in order to send the user the most relevant ones at a specific moment.

In order to accomplish asynchronous communication with the backend server, a multithreading-enabled execution environment is required to allow parallelism. If the execution environment supports this, the different activities can be done in parallel, providing for a better perception from a user perspective.

More precisely, one can for instance retrieve map data or tags from network servers while the user takes a photo or searches for an image in the file system of the device.

The Java ME platform provides HTTP connectivity. Thanks to the multi-threading support of the platform, HTTP connections can be spawned on different threads. A new HTTP connection can be created to send data to the server, while the user on the main thread continues interacting with the application. As long as the exceptions are caught, a thread will not crash the application.

The Generic Connection Framework in the MIDP supports the creation HTTP connections. However, it is important noticing that also other kind of connections such as SMS or Bluetooth are available through the Java ME APIs. This may be of vital importance if a BT-based GPS device should be accessed.

As opposed to the Java ME platform, asynchronous connections are possible in a RIA application through the triple XmlHttpRequest + JavaScript + DOM. In this case, the XHR object is a thread, spawned by the JavaScript logic, responsible for creating the network connection, transferring the data and waiting for the response. When the reply has arrived, it calls a function declared on the JavaScript logic, which will handle the response. By using this object, a synchronous HTTP request appears to be asynchronous. It is easier to create an XHR object than managing threads. Multithreading programming is a highly skill demanding task prone to errors.

### 5.1.2. Footprint

In order to interact with the REST-based services, the application needs the logic that allows it to create and parse the data format used by the service. Data is usually encoded in XML or JSON formats.

The Java ME platform has a good support for the XML data serialization formats. The JSR 135 provides a good implementation of a SAX XML parser. In case the preferred data format is JSON, json.org provides a lightweight library tailored for memory-constrained devices. Since the parsing is executed inside the Java Runtime, it is clear that the performance will be better than if the logic was written in JavaScript. Some overhead is bound to exist in case of interpreted JavaScript code.

However, this functionality could be easily abstracted to be used by the RIA logic, simplifying the development.
As for the actual footprint of the runtime environments, the Java Virtual Machine for mobile devices, also known as KVM, has a footprint of only 128KB\textsuperscript{16}. The device used for the evaluation, the SonyEricsson K850i has a total of 6MB available. Therefore, the overhead added by the KVM is not that huge. On the other hand, the RIA environment used properly obfuscated, takes up roughly 1 MB. However, since it has a prototype status, and then it is not optimized at all nor fully featured, its footprint cannot be used as a reference for this study. A commercial mobile browser such as Opera Mobile takes up around 1 to 2 MB of space. Bearing in mind that mobile devices bring more memory each day, the footprint is not a factor that could tip the scales in any of the environments favour.

\subsection*{5.1.3. Minimizing power consumption}

As stated in the previous chapter, several different factors can drain the battery in just a few hours. The use of a built-in GPS device, a camera or transmitting packets to the network are among the causes.

Minimizing the waste of the battery resource is a big challenge in itself. Nobody is willing to use an application if it means that the mobile must be charged frequently. People will be even less willing to use a program if the battery doesn't last for at least one day.

Mobile phones have more features than ever, allowing a user to execute several of them at the same time. Some of the applications, such as multimedia applications are quite power consumers. Developers must do their best to prevent battery draining since batteries capacity has not increased as dramatically as the need of power. Worst case must be taken into account, that is, a user that uses the mobile extensively. For instance, a user that uses the phone to call, sends SMSs, listens to music and browses the Web.

A blogging enabler platform should be able to minimize this consumption. Several solutions must be readily available in the platform.

Certainly using the radio link extensively the battery capacity drop quickly. Thus, data should be transferred efficiently, that is, only when is strictly required. Compacting data is a solution. Other approach is making use of the same HTTP connection to transfer a set of different data. This is also known as piggybacking. The data that is has been collected but is not necessary to send immediately to the server is stored on the mobile phone temporarily. When the user trigger an event that involves sending data straight to the back-end server, a connection is created and the bulk of the data stored is transferred to the server.

This way, power is saved because the period the radio component is on is minimized.

There are some times when data is essential to be fetched even when the user don’t issue any command. In the case of tag recommendations, these must be

\textsuperscript{16} http://java.sun.com/products/cldc/ds/
gathered at certain time intervals. Fetching data available on the server periodically from a client is also called polling. The platform should support polling driven by timers but also from external sources (e.g. moving to other base station), in order to minimize the requests. Since polling translates into creating a new battery expensive HTTP connection and transferring data, it also should be reduced to a minimum.

5.2. Web publishing requirement

These requirements are those that allow blogging from a mobile platform, or rather the lack of them will prevent users from publishing in an appropriate way and will hinder the development of the blogging solution.

5.2.1. UI implementation

The potential users of mobile blogging stem from users that already blog from the desktop. The blogging services they used all are web-based and thus use the same web technologies, sharing the same user interface experience. A RIA application brings the web UI to the mobile. By the use of web technologies such as HTML, CSS and JavaScript, the users experience an interaction with the mobile blogging application that is familiar to them.

Not only for users but also for developers is a familiar paradigm they can use to make the most of the user interface.

On the other hand, by using the web layer for the presentation, while the Java ME for the logic, the different parts are loosely coupled. Therefore, the business logic will be isolated from the user interface, being easier to modify either the appearance or the business rules without affecting the other respectively.

5.2.2. Networking

5.2.3. Accessing camera

Due to the requirements of blogging, having access to the camera phone device seems reasonable. As opposed to a desktop user that stays still on a room, a mobile user may be located almost everywhere. The mobile user could potentially change the context that surrounds him every now and then. This fact brings more opportunities to the user to be inspired and make him be willing to take a photo and use it in a blog post.

The JME platform grants access to the camera device through two different APIs. Old mobiles control the digital camera by means of the JSR-135. On the other hand, mobiles that are labelled as JP-7 or superior Java platform make use of the JSR-234, also known as the AMMS API.
This library allows using every capability of the camera such as the flash, zoom and of course capturing photos. Different sizes of the photo as well as photo rotation are available.

A RIA application should also have accessed to the camera in order to fulfil the blogging expectations. This functionality should be exposed from the underlying system somehow. Exposing this functionality exposes a risk since third party code download from a specific domain may have some malicious purpose. The user must be at least warned of the risks it entails given permission to this resource. Different policies could be applied such as asking for permission every time a site asks for access to the camera. On the other hand, this can be very annoying, so a good trade-off would be asking the user for permission in a per domain base. This way the user will decide if it trust the site and let it access the camera in future occasions or just this time.

One way of exposing the functionality could be through a wrapper API, also known as shim, which will convert the Java method signatures into JavaScript calls. This way most of the functionality would be exposed as it is. However, since current standard web technologies don’t support video, there’s no way for the web application layer to show a preview window of the camera, so that the user knows what actually is on the focus. The only possible preview would be after the user has taken the shot in the form of an image rendered in the HTML page. Since the preview functionality is not possible to be shown by means of the web layer, the user will be shown the actual JME user interface to perform this task. It follows the same paradigm found in browsers when a user wants to upload a file to the web. In this case, the native interface of the operating system is actually shown to the user. Whenever the user clicks on the camera button on the web page, the control will be handed to the JME layer.

As a blogging application doesn’t require advanced camera features, the API functionality exposed should be limited. This would also permit the JS developers to avoid writing the code that controls the camera, reducing the development time. A simple entry point exposed as a hosted object to the JavaScript should be enough. This object will have a method that would accept basic parameters, such as the size of the photo, and will deliver the reference to the photo. The photo itself could be stored on the JME layer, unless a preview is required to be shown on the HTML page.

Only one line of code would be required to take a photo in JavaScript.

### 5.2.4. Accessing gallery in file system

In the same way that the camera is exposed, the folder on the file system where the photos are stored should be accessible for the application. The place where photos are stored in the mobile phone file system is also known as the gallery. In the case of a pure JME-based application, the file system is accessible by means of the JSR 75. This API allows reading and writing files to the file system of the phone. It also permits JME developers accessing to the personal user information that includes the address book. The JSR 75 allows filtering which folders and type of files are shown to the user, so that these are restricted to images.
Just as the camera, the gallery should be exposed to the RIA application somehow. Although in this case, it would be possible to show this information in the form of a tree in the HTML file. This will effectively increase the logic necessary in the JavaScript layer, when it could be easily shown using the JME UI just as desktop browsers do. The underlying Java logic can be wrapper into a hosted object to expose a method that will enable the JavaScript developer to bring the load image dialog. A simple method that accepts basic parameters, such as filters in size or type of image, would suffice. This is equivalent to a unique line of code.

As well as with the camera exposing the file system to the JavaScript code could compromise the security of the system.

5.2.5. Accessing location

Thanks to the location data provided by either the phone or an external server provider, the context of the phone user can be inferred.

Due to the location data coming from both an external and internal device (GPS), an abstraction layer is required so that it becomes transparent to the JavaScript developer.

5.3. Desirable Features

5.3.1. Client Mashups

Data blending is the process of mixing data from one or more sources to create a new service or mashup. This service provides a distinct functionality not provided by any of the previous services. Mashup is a buzzword that involves the concept of data blending.

In recent years, the web is emerging as major application platform. Applications that previously were only found on the desktop now come in the form of web applications that are executed inside a browser environment. More and more parties like Yahoo, Amazon or Google, expose their computing services or data via public APIs. These APIs, also known as web services, are easily accessible using the HTTP web protocol. As the APIs are public interfaces, thousands of new web applications are created by mixing data from these several web services. One of the most popular mashups involves using cartographic data from a map provider to show other kind of information located on top of it.

Mashups can be placed on either the server or the client. A third party can create a new web service and deliver the new functionality to the user via the browser. The other way is mixing the data directly on the client platform.

A blogging platform should deliver a common framework to build the necessary publishing functionality. As it is impossible to foresee every single feature that a blogger may want to be included, it’s interesting to provide a way to make the system easily extensible.
One way of extending functionality is giving developers the opportunity to mix data easily from other services with the data that is already provided by the blogging platform. For instance, a developer could add the necessary functionality to allow the user publishing to several blogs services. The backend server and database provides specific tags and location data, but a developer could include other tag recommendation or location providers that complement or have data that is more accurate.

The mashup is properly done by using current web technologies such as the HTTP protocol to access the exposed web services. Due to the interoperability that the use of standard web technologies enables, mixing data from different sources is at everyone’s hand. Web Services using REST principles expose their resources via URLs. Applications can fetch and modify these resources using the GET and POST HTTP methods, exchanging data payloads using serialization formats such as XML or JSON.

Just by using the JavaScript and the XHR object, the data can be easily fetched. Adding new data sources could only mean modifying some JavaScript code from the HTML files. Taking into account that these files can be easily retrieved and copy into the mobile file system, new services could be easily deployed.

![Mashup on the client](image)

**5.3.2. Offline Solution**

An application that can be used offline is one that is fully or partial functional when no connectivity is available. Considering the unreliable network link, an offline mode is a highly desirable feature that would enhance the overall user experience.

Unlike a desktop user and thanks to the little weight of mobile phones, they can be carried everywhere with ease. This represents an edge in most situations. The ubiquitous presence of cell stations allows to send information to the network from plenty locations. However, there are situations where this is not possible, because no cell tower is found around or the signal strength is so low. It may also happen that any
new connection attempt to the cell station is blocked, as was shown previously, because all transmitting channels are busy.

It may happen that a person is in this situation, but still wants to publish some content to the web. One possible user scenario is a hiker reaching the peak of the largest peak in the ridge wants to take a photo of the view but finds that there’s no coverage in that location. The application is unable to send the data to the server at that precise moment. It’s not acceptable to show a disclaimer stating that it’s impossible to use the service at that time. Neither is good to store the photo in the file system and do the blogging later on. This could mean that data such as title, description text or tags is not available anymore. Maybe the user comes up with some words to tag the photo when he was there, but doesn’t remember them anymore.

A transparent solution is needed. One solution that leads the user to believe the data has been published. Even though, actually the publishing is delayed until coverage becomes available again. Whenever the signal is available again, the application should attempt to retransmit the content with the associated data.

To fulfil this requirement, two already commented features must be provided by the platform: prefetching and caching. Since it can happen at any time that the connectivity is low or none, tags should be prefetched at certain intervals. Not only by prefetching, but also by storing locally on the mobile, these get available for later uses. In an offline situation, the latest cached tags will be probably the more relevant ones to present to the user. In case a GPS device is available and since each tag will have its location metadata, those tags located next to the actual user location will be more relevant.

As well as with tags, the blog entry data, which was created by the user when the network was offline, must be stored internally in the mobile until the network is up.

\textbf{5.3.3. Portability}

Portability is a feature of software that allows the reuse of the same code base or components among different platforms.

A system is portable if the cost of adapt it to another environment is less than the cost of redevelopment*. Depending on the degree of effort and money, an application will be more or less portable. A highly portable application is cost-effective, and involves a lower time to market.

The point in portability is reaching the largest audience as possible. The more portable a component is the less the time and therefore money a developer will have to put on the porting, and more it can use to actual development of applications or services.

In order to aim at the mass market, is a must covering the widest range of mobile devices as possible. The share of market smartphones represent is 10% of the total. In 2008, 100 million smartphone units are going to be sold worldwide. Even though is a big number, there’s no point in constraining to this niche market.
Obviously, one of the requirements of the various architecture designs is providing a solution for most low-end and mid-range mobile devices.

Portability has been one of the Java platform characteristics. The Java platform seeks abstracting the operating system, providing a common interface to programs for interacting with the underlying architecture. An application developed on top of the Java platform is cross-platform, as long as a Java Virtual Machine exists for the target operating system. The motto “Write Once, Run Anywhere” actually holds for the desktop and server platforms. In the case of the Java ME platform several issues prevents it from being portable (7). Unlike Java SE, in Java ME does not exist a major reference implementation, but every manufacturer provides its own implementation of the platform. Apart from this, every device bring a different set of Java APIs. Consequently, a developer either has to focus on a set of devices that have the same APIs, use only those features found in most devices, or port the application.

Web applications that run inside a web runtime engine can run on other environments as long as they also support the same web standards. Ten years ago when the browser war between Netscape and Internet Explorer was at its peak point, both browsers have their own extensions to the standards. Therefore, web applications were incompatible among browsers. However, nowadays most browsers on the desktop and on the mobile respect and implement new standards. A web application that uses HTML, CSS, JavaScript and AJAX will be able to execute on a desktop, on a mobile and on a widget environment.

The main issue is that building new standards is a process that takes a long time. As a result, there is a demand of new features on these environments that are not fulfilled by the standards, and every party involved create their own set of APIs. There is a high risk of API fragmentation that will hinder the portability of web applications.

### 5.3.4. Minimal development difficulty

This feature refers both to the time it takes to create the blogging service from scratch and the level of programming skills needed to do so. Not every single language takes the same amount of time for developers to learn before they could start to be productive.

Developing for one platform or another involves using different APIs and programming languages. Some of these languages have internal characteristics that make them more productive or less prone to errors than others. Languages running on top of a virtual machine usually benefit from features such as garbage collection. Thanks to these handy features, programmers may spend more time on the application logic and less time debugging.

In the same way, using some APIs rather than others can improve the overall productivity of the programmer. This can happen because one API further abstracts the underlying system than other leading to fewer lines of codes to write the same functionality. The more complex functionality is written in the API the less functionality a developer has to provide himself. How well are the API documented can also be a decisive factor in the programmer overall productivity.
It is obvious that different languages and platforms don’t require the same level of skills. Some have a steeper learning curve than others do. Those with a fast or short curve are more likely to be used by the greatest number of developers. A given architecture should leverage these low-medium skilled developers in order to boost the development of new functionalities on top of the platform.

The Java language has some intrinsic characteristics that make it more challenging for new developers.

JavaScript is a script language that enables developers to create their own logic separated from the user interface code. Having the document structure in HTML and the presentation in the CSS files, the logic in JavaScript is not intertwined with further issues.

Even though JavaScript took the name after the Java programming language for marketing reasons, both barely resemble each other in some syntax characteristics but they are completely different languages. Unlike Java, JavaScript is a weak-typed dynamic scripting language that allows rapid prototyping. Bearing in mind that the blogging application logic is not that complex, it will definitely reduce the time a developer need to build the service.

New developers that are not familiar with the Java environment find more convenient having such features. The JavaScript language also features an object-oriented like system, called prototyping. It is not based in classes and inheritance the way Java is, but in cloning existing objects that serve as prototypes. Anyhow, reusability of objects is enabled in this way.

Regarding Java experienced developers that are not that experience in JavaScript, solutions such as Google Web Toolkit (GWT) enable them to write the web application in Java, and get it compiled to a JavaScript Ajax application that abstract most of the different browser incompatibilities. The overhead of this approach is that this application uses a GWT runtime or set of libraries whose footprint is roughly 100Kb. It is logic that in the near future these toolkits will have a better support for mobile browsers, bearing in mind that some major players use the same rendering engine as other desktop browsers already supported, as is the case of WebKit.

For non-experienced JavaScript developers different toolkits or libraries that ease the development of cross platform Ajax web applications are also available. These are usually open source and their footprint is small, varying from around 20 to 100 Kb depending on the overall functionality the toolkit provides. These libraries usually help the developer deal with the XHR object and provide advanced widgets that are consistent among browsers. Some of these toolkits are Dojo, JQuery or Prototype.

On the other hand, there is a lack of fully-fledged development environments and debuggers for JavaScript programming. The good point is that developers do not need to learn how an IDE such as Eclipse works. They don’t need to compile, they don’t need to link libraries, or dealing with dependencies and so on. The browser can be its only tool. Some browsers such as Firefox have plugins to help developers on their task. In the case of Firefox, the Firebug plug-in permits editing and inspecting the html or JavaScript code and well as the DOM model. It also brings a debugger and profiler built-in, and it has a much less steep learning curve than Eclipse for instance.
The only thing they need to know is XHTML, CSS, JavaScript, XML and AJAX. The good thing is that there are many resources out there that a developer can use to learn these technologies, and most of them are free. Furthermore,

Before there was some portability issues, above all at the peak of the 1st browsers wars between Netscape and Internet Explorer, and the following years of total dominance of Internet Explorer. Developers had to check whether the web app was going to be rendered in a browser or another, and have different chunks of JavaScript code coping with each browser.

After seen how every browser was adding its proprietary extensions to the web technologies, the W3C consortium was created to get browser vendors to agree on current and future web technologies standards. At present, most web browsers follow the standards. This is even more notorious on the mobile platform. Since Windows Mobile is just a small player in this field, the use of Internet Explorer is not widespread. On the contrary, most browsers are based on the open source WebKit engine. This engine is part of the browsers: Safari for iPhone, Google Chrome for Android and Web Browser for Symbian S60. All this three platforms (iPhone, Android and Symbian) accounts for more than two-thirds of the smartphones worldwide. Therefore, most users actually access the web from an open source standard-compliant browser. If things continue going this way, WebKit will be the de-facto standard on the mobile phone browser market. As WebKit is aiming at being fully standard-compliant, the same application could be delivered across the different platforms. Summing up, for the developer it means less time-to-market, and with the same cost reaching all different platforms, not just the most popular one.

5.3.5. Ease of deployment

5.4. The two candidates

Among all the different architectures that were possible two compare, only two of them were chosen for a thorough contrast, the Java ME-based and the RIA designs. As stated above, a web application running on top of a browser would have been a bad solution because of the absence of access to the device resources. Therefore, users would have been prevented from performing photoblogging. The native approach was discarded due to being available only in mid-range to high-end phones, and having a great variety of OS. Thus, an extra effort to port the application would have been required.

Both the Java ME and the RIA architectures have been compared in the same areas. Summing up, the Web-based RIA seems easier to develop, enforce reusability of components and web services (mashups) and provides portability. As stated above, full portability could be hinder if the different APIs that provide access to the local resources become fragmented.
The Java ME approach requires less memory footprint than the RIA, since an HTML renderer and JS interpreter is needed for the web application. It is deployed with most mid-range and low-end devices. Consequently and bearing in mind the effort of porting the application, it is still a good approach to bring mobile blogging to most of the mobile subscribers.
6. Implementation

6.1. Design overview

In order to build and test both the RIA and JME prototypes, a backend infrastructure is needed. The system goal is providing a blogging service that consists of a distributed system of client and server that communicate to each other over the HTTP protocol using a specific data format.

![Figure 3 – Overall Architecture](image)

6.2. Core Structure

6.2.1. RESTful Architecture

As stated before, one of the desirable features of the publishing service was making it reusable, that is made it accessible over the network so that it can be used and combined with other services. By embracing the World Wide Web technology as the platform for the publishing service, other people can use the exposed data and merge it with other sources, creating new interesting mashups.

To make it possible the creation of mashups as well as the interoperability with different devices and applications, it was necessary to expose the APIs in the form of a web service. Among the different architectures styles, REST was chosen.

REST (10) stands for a collection of architecture principles that make use of the Web, usually the HTTP protocol, to transmit data to and from a web service. Strictly speaking, REST just state how to define resources and how to access them.
In REST, a **resource** is a source of information, regardless its representation, that is distinctly identified by a URI. An application that wants to manipulate a resource would need the URI identifier and the **action**. In the HTTP protocol, these actions will translate into the four different **verbs** defined in the protocol itself. These verbs are POST, GET, PUT and DELETE. They match the four basic functions we found in a relational database: Create, read, update and delete, respectively.

Knowing both the identifier and the action to be carried out, is trivial for any new developer to build an application that interact with the publishing service, leaving the decision of choosing a platform and/or programming language to the developers themselves. Nevertheless, an application must understand the representation used by a service when returning a specific resource. Usually this representation is either XML or JSON, but may be any kind of format. In fact, as it shown below, the data format used in our prototype is a different one. Just another benefit that should be stated is that every state will be modelled as a resource on the server, since the HTTP is **stateless**. Thus, no session is required, and the use of cookies is avoided.

The platform chosen to build these services on is the Java Enterprise Edition. The application server used is GlassFish, which is open source. In order to expose the services, the Jersey framework was used. Jersey is an open source reference implementation of the JAX-RS (JSR 311), that is, the Java API for RESTful Web Services.
The application exposes several different web services. The ones that are related to the publishing service are location, blogging and tag recommendation. The location service allows the application to get the position a user in any specific moment. It exposes the service as the following URL that can be accessed using the GET method of the HTTP protocol.

\[http://\{servername\}/\{username\}/location/current\]

In the same way the blogging service lets applications upload new blog entries to the system in this case using the POST or PUT methods if it’s a new entry or the entry is going to be modify respectively. In addition, if there is a photo associated to the blog entry, the related resource can be uploaded to the server. It is worth noting that the location.

The tag recommendation service will be in charge of delivering the recommended tags to the application in relation to the current location of the device or the first letter input by the user in the corresponding text field. The recommendation algorithm that lies behind the service has not been actually developed, but is easy to figure out which factors could influence the result. Tags that have been used in or near the user current location, those that are more or less recent in time, and those used by the user or a member of his social network could rank high on the list.

All this data must be stored somehow on a server database. The backend RDBMS used in this case is MySQL, which is open source as well. On the publishing side, the blog entries, images and associated metadata such as location, tags, and time are stored on the database. Tags related information such as the number of times a tag has been used by the recommendation system or times is used, are also profiled.

**6.2.2. Protocol**

RDF parsing and creation is not supported natively by the Java ME platform. Due to this inconvenience, a third party open source solution was necessary to provide this functionality. A semantic web framework for Java called Jena had been ported to the JME platform. The RDF graphs are represented in a model that is updated whenever more data is generated from the server or the client. Therefore, at any time, the model is on the memory and data can be added or extracted. It seemed a good idea having the same framework in the server and the client, so that the code base dealing with RDF was the same in both parts. However, it was necessary to port the code minimally to Java ME, due to different Java language features not founded in this platform. In spite of that, the JME-based Jena or µJena framework was not completely ported so some constraints were set on the platform. The serialization format used for RDF was N3. It is worth mentioning that this format it was not the most optimized one, since namespace prefixes were not allowed. Therefore, the same long namespace strings were included several times in the file. However, a pure RDF serialization based on
XML would have been even more verbose, and also was not provided in the ported framework.

By leveraging RDF, semantics are applied to the data in form of triplets that define a relationship between two different entities. There are a bunch of standardize RDF namespaces that can already be used to apply semantics to the data. Some of the most well known ones are the Dublin Core or FOAF. The good thing is that these namespaces can be mixed with newly created ones to create other kind of relationships or semantics that are relevant for our application. Even if the new vocabulary brought by our namespace is not published so that other can understand the semantics, the other data could be extracted and still be use by others. This is a key point in the creation of new mashups.

RDF is used in the prototype to send data to and from the web service. The RDF file is made up of triples in the way of subject – predicate – object URLs. The publishing sequence diagram shows how this exchange is done. First off, the client retrieves the location data from the server that is formatted in RDF/N3. Second, the client sends an RDF file to the server with the post-related information. Since the post and the possible image don’t have a resource URI set yet, the RDF file is send to a “new post” service. This service will send back the URIs created by the server for the post and image entities. Finally, the client will upload the image to the corresponding URI and update the post data.
6.3. Prototypes

In order to compare the two different software environments, a Java ME and a Web-based RIA were developed.

6.3.1. Java ME Client

The first of the prototyped clients is completely based on Java ME technologies. All the business as well as the user interface logic is coded in the MIDlet, which is packed in a .jar file. In this case, along with the classes provided by the Java platform, the system will provide its own APIs. The location, camera, gallery, network stack, and necessary logic to publish to the web, are created as Java classes on the MIDlet. Since the data format used is RDF, it is necessary to have an RDF parser on the client. \(\mu\)Jena is the chosen one, since it was lightweight and a ported version of the parser that it is used on the server. However, it is far from being feature-complete since it just supports the serialization of RDF using the N3 format without prefix supporting. Consequently, the RDF data is more verbose affecting the overall time the publishing process takes.

![Java ME Prototype Architecture](image)

6.3.2. RIA Client
The Java ME runtime environment offers user interface rendering, local data storage and network communications. However, it doesn’t provide a framework to build rich web applications on top of it. For the purpose of this thesis, a runtime environment in the form of a Java ME MIDlet must be designed to provide the rest of functionality lacked. It’s worth noting that the actual runtime environment was not a complete system but rather was in alpha stage.

The actual architecture of the MIDlet runtime engine that enables the RIA application comprises several components:

- **Renderer.** It uses Java ME graphical components to draw the various HTML widgets on the screen. It also takes into account the style elements embedded in the HTML when drawing the web page. The use of Java ME widgets makes it possible to deliver a user experience similar to a native application.

- **XML/HTML Parser.** It parses the HTML documents and builds the DOM\(^{17}\) tree structure from this data.

- **uJena Parser.** It parses the RDF chunks of data coming from the server and builds a memory model of the triplets. From this model, the data can be extracted easily by the application. It also allows the creation of RDF documents to be passed to the server.

- **JavaScript Interpreter.** It interprets the JavaScript embedded in the HTML documents, executing the code and returning the results of the execution. This JavaScript code allows the programmer to modify the DOM and use hosted objects that interact with the underlying APIs.

- **Network stack.** It provides the necessary functionality to send HTML requests and receive the responses from the different web services. Following the REST principles, it allows sending GET, POST, but also PUT and DELETE requests.

- **DOM API.** It allows modifying dynamically the HTML elements or the style of the web documents. Through a specific hosted object, the DOM API is exposed. Therefore, JavaScript developers could easily call this API to modify the document structure.

These components provide the key features to enable RIA apps, but no support is added to aid blogging publishing. JavaScript developers that want to build services over this platform would have to access some native resources through the Java ME runtime engine. JavaScript developers can make use of these Java APIs by creating host objects. These objects are adapter classes that wrap the logic defined in the Java ME layer, enabling access from the JavaScript environment.

The logic needed to access the camera, file system, as well as the location abstraction and blogging functionality is placed in this layer. Via well-defined entry

---

\(^{17}\) Document Object Model
points, the JS developer can make use of this functionality. HTTP requests are enabled via the XHR object.

On the other hand, the user interface logic is located on the web layer.

As it was advanced previously, the user will have the chance of looking in a map her current position. This is clearly a mashup among the location data provided by the system and the maps retrieved from a third map provider. The maps can be fetched by means of a public API that returns the map image or via AJAX requests, as long as the developer has the corresponding API key.

In the same way, the JavaScript code could have been modified to retrieve the same maps from other provider in a short time.

![RIA Prototype Architecture](image-url)
7. Evaluation

In order to evaluate the prototypes, a mid-range mobile phone was necessary. A phone that had a camera built-in and could run Java ME applications since both prototypes are based on this technology. The chosen one was the SonyEricsson K850i. This mobile phone has 3G connectivity as well, which enable to test the application on the newly deployed networks, what could theoretically provide better throughput. Finally, the heap size is dynamically allocated by the system depending on the availability of space in the internal memory. In any case, it will be less than 40MB for every application running on the system providing there is no other data stored in memory. In case of Java ME sandboxed applications, the size of the jar file is actually unlimited but since this doesn’t apply to other mobiles, the functionality of the Web Runtime Environment was constrained to 1MB. It’s also worth noting that even though there are maximum 40 MB available for the heap, a Java ME application is limited to 5 MB tops.

7.1. Responsiveness

As assumed above, the overall time to tag an image and publish it should be in the range of 1 second. In the RIA realization the measurements done is depicted in the table below.

<table>
<thead>
<tr>
<th>Create Post w/ Image</th>
<th>Average Times (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>22392</td>
</tr>
<tr>
<td>Java</td>
<td>648</td>
</tr>
<tr>
<td>JavaScript</td>
<td>1824</td>
</tr>
<tr>
<td>RDF Parsing</td>
<td>12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>24876</strong></td>
</tr>
</tbody>
</table>

The measurements were done in a setup with a Sony Ericsson K850i using a WCDMA network supporting 64 KBps. Measurements were done in different times of the day to get an average at peak as well off-peak hours.

As stated in Table 1, the average delay caused by the MIDlet (Java + JS), was in the range of 2 to 3 seconds. The JavaScript interpreter used was implemented on JME, and performance from comparable engines like Webkit can be assumed to be faster, as is later shown. Clearly, the logic contained on the JME layer, in charge of dealing with the image, the RDF parsing and sending both through the network performance was good and expected. However, the use of the JavaScript interpreter, which was in alpha development, resulted in a big performance penalty. The renderization of a single page takes 2.5 seconds on average.

All this figures take into account that the 3G radio on the mobile is up and running. Otherwise, it took around 6 seconds to warmup before sending the packets.
Some parts of the DOM API were still not exposed to the JavaScript layer. Only those parts that were strictly necessary were hacked to deliver the functionality required. Due to these constraints, the whole HTML page needs to be rendered every time the JavaScript modifies the DOM tree. The time that it takes to parse, execute and create the new HTML document is far from being optimal due to the alpha status of the runtime. Still, is not far from 1 second, the time were the user realizes the response is not immediate, but his flow of thought is not interrupted. No wonder, other more mature engines will provide a response rather close to 100ms.

In a previous thesis project (9) that made use of the same web runtime, Erik Hedin measured the difference in execution speed between the JavaScript engine embedded in the Netfront browser and the interpreter we used. After benchmarking both using the Sunspider JavaScript benchmark, it become plain that the JS engine use in our MIDlet perform as much as 8x times faster than that included in NetFront. However when compare with state-of-the-art engines included in mainstream browsers, it is evident that there is much room for improvement in the JS interpreter performance. A recent benchmark\textsuperscript{18} using the same SunSpider benchmark tool, shed light on this issue. SquirrelFish, the JS engine that comes with WebKit, it is 120x times faster than our engine. However, in this case, the test machine is a not a mobile but a fairly old desktop computer (1GB of RAM and 1.8GHz AMD Turion). Yet, it is clear that with faster and faster JS interpreters and more powerful hardware on the upcoming mobile phones, the responsiveness of these widgets can only increase.

The conclusion is that for this operation, even the RIA prototype parser was sufficiently fast to get in the range of what is assumed to be acceptable. Commercial engines can be assumed to be considerably quicker. In the end, the responsiveness is mainly dependent on the radio technology used and the load in the cell from other devices using the shared radio media.

As the table shows, the mean time to upload an image was 56Kbits per second at most. A regular image taken with the phone camera at 1Megapixels that occupies 125Kb took on average 22 seconds to get to the server. A speed that resembles that of legacy modems. This is really what makes responsiveness an issue on mobile phones. AJAX techniques allow uploading and downloading the data to the server, letting the user being in control of the user interface. In this kind of applications, that is, those that enable publishing to the web, the network delay doesn’t hinder the overall use experience. When publishing with the Shozu Java ME publishing client, these estimations are corroborated.

Another factor that makes responsiveness a bigger deal is the time it takes for the phone to power on the radio. In case the radio is down, roughly 5 seconds are to be added to the previous timings.

7.2. \textbf{Reusability}

In terms of reusability, it is clear that the RIA approach is the way to go. The JME application needs to be installed by SMS, cable or BT every time a new characteristic is

\textsuperscript{18} JavaScript speed race \url{http://ariya.blogspot.com/2008/06/javascript-speed-race.html}
added. If one were to add new functionality, this would have to be placed into a new or modified class in the MIDlet jar file. Then this new jar file should be deployed by one of the above-cited means. This is process could be both troublesome and costly.

The RIA let developers create new functionality to pre-existing services just by copying the new html, css or JavaScript files into the corresponding folder. It’s worth mentioning that adding code from a 3rd party could compromise the security of the system, since these programs may have access to the underlying platform resources.

Due to the intrinsic characteristics of web technologies, it gets easier for developers to get other people JavaScript snippets to create their own. These days, every browser has a View Source feature built-in. This means that JavaScript code from AJAX-enabled web applications become available for users to look into, providing that the code be not obfuscated. Furthermore, many free resources on JavaScript and web technologies are available on the web.

Nevertheless, what is even more important is the availability of libraries and frameworks in JavaScript that make it possible to reuse common functionality. As it occurs right now on the desktop, in a near future

8. Future work

8.1. Usability

One of the issues not evaluated in this paper is whether a web-based UI is more usable than a Java ME counterpart is.

Since mid-range and low-end mobile phones lack of a pointer device, users cannot point directly to the area of the screen they want but rather scroll through the screen using the keypad buttons. It is likely that a Java ME application could perform the same tasks in fewer steps since it not only has access to the graphical widgets shown on the screen but also to the soft buttons. This buttons presented in every device can trigger a menu that shows different commands, which could potentially save some button clicks.

Other issue could involve the different interaction paradigm that a desktop application provides in contrast to a web application. Although web applications have come a long way to mimic desktop application behaviour, it may also be that most bloggers are comfortable with the web paradigm and then the learning curve is quick. However, neither mobile nor mobile web applications are quite the same as desktop or fully-fledged desktop web applications, among other things due to the physical device constraints.

8.2. Security Framework
In the Java ME platform exists a security framework that prevents any untrusted application from accessing the local resources such as the camera or the network stack. The MIDlet should be signed either by a third party authority that has their certificate installed on the mobile or the user would be prompted to allow the application.

In the case of a widget application, there are some concerns about how it is going to access the phone capabilities. A widget that needs to be signed and installed on the system would defeat the ease of deployment characteristic. However, it is less of a problem to prompt the user to allow access in a per-widget basis than risking a malicious program accessing the user personal information.
9. Conclusions

The scope of the research was analyzing different software environments that could possibly enable publishing to the web from a mobile platform. The work has focused not in publishing to the web in general, but specifically to a blog or photoblog.

Mobile handsets due to their physical capabilities and limited connectivity impose some constraints in the system, in contrast to the desktop platform. However, the same physical characteristics of the handset also allow users to carry them everywhere, and then, context-aware services such as those using location data become possible.

As mobile phones become more and more capable, both in memory and processing power, the applications will be more functional too. Fewer limitations in memory use translate into more data that can be stored on the device. Responsiveness will be improved by caching more data likely to be needed by the user in the short term. Other processes like the parsing of RDF-encoded data that uses a relative high amount of memory will benefit from these more advanced handsets.

Two are the main issues that mobile limitations entail. On the one hand, the interaction with a handset by means of a keypad is a poor experience. Due to the reduced size of the screen, not much information can be shown to the user. Thus, the creation of user interfaces tailored to mobile devices becomes a challenging task. On the other the lack of connectivity involves building a solution that supports the use of the application in such situations.

One of the main requirements was focusing on the widest audience as possible. Bearing in mind that most people own a mid/low-end device, and that the application should be as portable as possible, the alternatives were rapidly narrowed down to two different approaches. These were building a fully-fledged Java ME client or a RIA client.

Two different prototypes were built. One was entirely developed in Java ME, including the user interface. The other one is a RIA client built upon a runtime engine that provides the necessary functionality to make publishing possible. This runtime engine is a Java ME MIDlet that makes use of the underlying environment to access those native resources needed by the RIA, such as the camera phone or the GPS device. Furthermore, it provides the necessary logic to parse and render the HTML and RDF data, as well as create the HTTP connections to access the RESTful web service interfaces. Other libraries were also added to the engine enabling basic blogging functionality.

Unlike its counterparts, Java SE and Java EE, Java ME portability is far from being a reality in mobile devices. The sentence “Write once, run anywhere” doesn’t apply since every manufacturer include their own Java ME implementation of the specification, which can differ.

By enabling developers to create their own applications by using well-known web technologies over the enabler runtime engine, we’re killing two birds with one stone. On one hand different tailored versions of the runtime engine could bring portability
closer to reality. On the other by means of standard web technologies third party developers can create their own applications. They may even come up with new mashups by merging data from one or more web services, with the data provided by the Marco system. Other service APIs that use REST, and XML or RDF as their data transferred format may enhance the application. These services could include map, location or better tag recommendation providers.

The responsiveness of the user interface was notably better for the Java ME client than for the RIA one. It is worth pointing out that the runtime engine used was not stable and the performance was far from the desired. Thus, no evaluations were possible on this issue.

As for the easiness of development, building the user interface took less lines of code in the HTML/JavaScript approach than in the Java ME. Overall, many more developers have the necessary skills to build the web app using web technologies than Java itself. In addition, the learning curve is steeper for Java. Therefore, the targeted developer audience is greater in the RIA approach.

Summing up, a platform consisting of an enabler runtime engine supporting the creation of user interfaces and additional functionality using web technologies seem the best approach. Better support for JavaScript development could reduce the time novel developers take to create web applications, and then boost the development of new services. In addition, further optimizations in the JavaScript engine could improve the overall performance in a near future.

Bearing in mind the ubiquity of mobile devices, and that their users are plenty of times in motion, developing an offline solution that enable publishing data in situation with no connectivity seems the next big challenge.
10. Works Cited


## Appendix I  Use cases

**Person – Contacts – Global community**

<table>
<thead>
<tr>
<th>User scenarios</th>
<th>Challenges</th>
<th>Technical needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>User retrieval</td>
<td>• Provide tags already used in the past.</td>
<td>• Store user profile and track the context of any published item.</td>
</tr>
<tr>
<td>Anna wants to be able</td>
<td></td>
<td>• Define an algorithm to suggest or attach relevant tags depending on his current context.</td>
</tr>
<tr>
<td>to retrieve the photos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>of her children easily.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global retrieval</td>
<td>• Provide the most relevant tags for that piece of work depending on other users’ tags in the same location and/or context.</td>
<td>• Define an algorithm to suggest or attach relevant tags depending on his current context and frequent tags used at that point.</td>
</tr>
<tr>
<td>Miriam is a professional photographer interested in reaching many people to show them her work.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context recall</td>
<td>• Provide tags that communicate context to the content.</td>
<td>• Define an algorithm to suggest or attach relevant tags depending on his current context.</td>
</tr>
<tr>
<td>Anders wants to be able to recall details of a photo when he looks at it after a while.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Larry wants to communicate the details of a photo to an interested user that finds it.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context awareness</td>
<td>Security layer</td>
<td>Metadata election</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
<td>-------------------</td>
</tr>
</tbody>
</table>
| **Ashley arrives in New York for the first time in her life. Thus, she expects tag suggestions for a tourist, that is, different from those for actual New Yorkers.** | **John only wants to show the photos of his wedding to his friends and family.** | **Paula likes to blog about things she does in her everyday life, but she doesn’t want people to know exactly where she is.**

*Gina invites her friends to spend the night in a cabin in a forest. She uploads a photo of the actual cabin, tagging it with the real location of the cabin, not her current location.* |

| **Recognize habit patterns (e.g. work/home location or being new in a place) and store them in the user profile.** | **Allow different kind of accesses to the content.** | **Let the user uncheck the associated tag metadata (e.g. location) of the content.**

**Let the user modify the value of the metadata.** |

<p>| <strong>Track the location and time when a user tags his content, to be able to infer the context.</strong> | <strong>Create categories (me, friends, family and rest) of people who can access a specific piece of content.</strong> | <strong>Let the user choose the security the audience of the content or infer it automatically depending on the context (e.g. tags used).</strong> |</p>
<table>
<thead>
<tr>
<th>Tag pool</th>
<th>People presence</th>
<th>Erroneous metadata</th>
<th>Location-aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richard attends the presentation of a new player of his favourite soccer team. He would like to tag his photos with the same keywords of most people, so he can retrieve all of them later.</td>
<td>Bob is making a photo of him and his friends. He wants the name of his friends to be added as tags.</td>
<td>Frank is watching a live show, but he erroneously tags the artist with a different name. Once he uploads the content, he realizes the mistake and wants to modify the tag.</td>
<td>Matt takes a photo of the Eiffel Tower from the top of the Montparnasse tower in Paris, which is a few miles away.</td>
</tr>
<tr>
<td>• Suggest recent tags to users as soon as they become relevant.</td>
<td>• Automatically tag the content with people’s name in your social network, present at that moment.</td>
<td></td>
<td>• The system should search for tags not only in the current location, but also on the vicinity.</td>
</tr>
<tr>
<td></td>
<td>• Look for friends or acquaintances on you social network and check if the Bluetooth IDs around you, match some of them.</td>
<td>• Let the user modify tags associated with previously uploaded content, or even delete the content itself.</td>
<td>• Prefetch data of a wider area in advance every time the user gets a different cell ID or after a predetermined time interval.</td>
</tr>
</tbody>
</table>
### Offline solution

*Claire is in a small town of the Pyrenees, where there’s no network coverage in the area. Nevertheless, he wants to write a blog entry, and tag it with her actual location*

- Send a tagged content to the server when there is not network coverage now.
- Let the user tag the content with the undefined *location* tag.
- Store the tagged content, to send it immediately after coverage is back.
- Activate GPS device in the background when needed, to get the current coordinates.

### Location accuracy

*Carl takes a photo where MXE location is not accurate enough. He disagrees with the location suggested by the system*

- Provide a way for the user to tag the current location, when the one the server provides is not good enough, and not GPS device is available
- Show the user a map of the current area, so they can indicate where they think they really are. Tag the content with that location.

### Database retrieval

*Mark stops the car in a little town of Spain where a local festival seems to go on. He takes a photo and he would like to be able to tag it accordingly with the event.*

- Provide external context information as tags for the content.
- Fetch extra data related to the user current location from online databases (e.g. weather reports or events). Send them along with the prefetched data.
### Screen constraints

Sophie would like to tag her content before uploading it. Since it’s cumbersome to input user-defined tags from a mobile keypad, she expects the system to make easy to tag the content with the tags she expects.

- Show the user only the most relevant tags and make shortcuts to them.
- Show just a number of tags that fits on the screen and are the most relevant. Associate a key number with each of them.
- Create tabs in the interface. Each one of them would group those tags that are under the same category. These categories could be for instance, recent tags, user-made tags, friend tags or tags at the same location.

### Resource scarcity

Mathias would like to save as much battery as possible, so he can also use the phone to listen to music or making videos.

Laura doesn’t have a flat rate and so she wants to be able to spend as low as possible.

- Save as much battery as possible without reducing application functionality.
- Let the user choose different degrees of functionality.
- Activate GPS device just when needed.
- Piggyback data when prefetching data from the server.
- Let the user choose if they want the suggestions feature to be enabling or not.
Appendix II  Abbreviations

Ajax – Asynchronous JavaScript and XML
CSS – Cascading Style Sheets
DOM – Document Object Model
LOC – Lines of Code
HTML – Hypertext Markup Language
HTTP – Hypertext Transfer Protocol
IP – Internet Protocol
Java ME – Java Platform, Micro Edition
Java SE – Java Platform, Standard Edition
Java EE – Java Platform, Enterprise Edition
JSON – JavaScript Object Notation
RDF – Resource Description Framework
REST – Representational State Transfer
RIA – Rich Internet Application
SOA – Service Oriented Architecture
SMS – Short Message Service
URI – Uniform Resource Identifier
W3C – World Wide Consortium
XML – Extensible Markup Language