Leveraging the Web Platform for Ambient Computing: An Experience

Fabio Mancinelli

Leveraging the Web Platform for Ambient Computing: An Experience

Fabio Mancinelli

INRIA Paris-Rocquencourt, Project ARLES
Domaine de Voluceau-Rocquencourt
B.P. 105, 78153 Le Chesnay Cedex - France
fabio.mancinelli@inria.fr

Abstract. This paper explores a very simple idea: what can be achieved by using the principles and the technologies of the Web Platform[1] when they are applied to Ambient Computing? To answer this question we will present an experience that addresses some of the problems of Ambient Computing by making use of the technologies and the common practices of today's Web Platform. The aim is to provide an architecture that tries to lower the deployment costs while guaranteeing accessibility, interoperability and extendibility.

Key words: Web Platform, Ambient Computing, Interoperability

1 Introduction

Ambient computing proposes a paradigm where devices present in a physical environment collaborate in order to support people in carrying out their daily tasks. Though many technologies have been introduced in order to implement this paradigm, there are still a lot of open issues that must be addressed and solved. In particular many of these issues concern the *"ubiquitousness"* of these solution and their interoperability with the multitude of heterogeneous devices that might interact with such a kind of systems. In this paper we explore a very simple idea: what can be achieved by using the principles and the technologies of the Web Platform when they are applied to Ambient Computing? This idea is motivated by the fact the by leveraging the Web Platform we can address interoperability issues by relying on a uniform platform that has a well defined set of protocols and and semantics. This is very important because different solutions must interoperate in order to provide a better experience to the end-users. By using the Web Platform as a common ground, and by building on top of it, many of the interoperability issues are solved forefront. In this paper, basically, we took the Web Platform as the reference platform, and built on top of it a simple system for Ambient Computing paradigm. Our aim was to describe the experience we did in building an actual system and to present what are the advantages that, in our opinion, derive from following such a kind of approach.

[1] With the term "Web Platform" we refer to the ensemble of the protocols and standards the World Wide Web is built upon.

2 The Web Platform

In this section we will briefly introduce the Web Platform and what it consists of. We used the term *platform* because nowadays developers are using the Web, its architecture and the technologies it is based on, as an actual platform for engineering, developing and deploying their applications[2]. The idea of the Web as a platform is also corroborated by all the companies that are producing advanced tools and solutions for easily building complex application without taking care of all the low-level details [2, 3]

The Web Platform has an architecture that is based on a well defined set of principles and constraints [4] that are implemented by a set of standard and widely deployed communication protocols [5], and makes use of commonly used data formats for exchanging information [6].

The key principles and constraints described in [4], that are relevant for the purpose of this paper are:

- *Addressability*: Every resource must addressable by a well defined resource identifier.
- *Uniform interface*: A single interface, with a well defined semantics, must be used for accessing and manipulating resources.
- *Hypermedia as the engine of application state*: By leveraging addressability, the execution of the application is obtained by following hypermedia links to resources that represent the "next state" of the application.

These principles are implemented in the Web Platform in the following way: *addressability* is given by using Uniform Resource Identifiers (`URI`s) that provide a *"simple and extensible way for identifying an abstract or physical resource"* [7]. `URI`s are used extensively in order to address any resource[3], and to operate on their representations. `URI`s also enable hypermedia features that are used as the primary mechanism to make applications change their state.

The *uniform interface* is implemented by the `HTTP` protocol that provides five standard methods with a well defined semantics[4] for interacting with resources.

The `HTTP` protocol is a very simple request-response protocol that exchanges messages consisting of a set of headers and a body. Headers contain meta-information for further describing the `HTTP` message; the body contains data to be associated to `HTTP` messages depending on their nature. `HTTP` headers are essential for making messages self descriptive and amenable to advanced pre/post processing.

[2] This is basically something similar to what happened with the Java Platform [1], with the introduction of the Java language, its standard libraries and all their extensions like the "Enterprise Edition" or the "Micro Edition".

[3] The term "resource" is highly generic but, as stated in [8], we can say that a resource is *"anything that is important enough to be referenced as a thing in itself"*.

[4] These methods are the well known `GET`, `HEAD`, `PUT`, `DELETE`, `POST`, and some other less known ones: `OPTIONS`, `TRACE` and `CONNECT`. Though the semantics for these methods is well defined, `POST` is an exception because its actual semantics depends on what is specified in the body of the corresponding `HTTP` request.

Another important aspect of the HTTP protocol are their response code. Each HTTP request comes with a response containing a code used to communicate the result of the request. There are several response codes, grouped in classes: *meta*, *success*, *redirection*, *client-side error* and *server-side error* classes. For the purpose of this paper, the *redirection* class will have a particular relevance, as it will described below.

To conclude this section we would like to point out that the most important part of the Web Platform consists of all the standard format, languages and clients that are used to create leverage for the web applications. In particular the XML [9] and XHTML [10] formats, the *JavaScript* [11] language and the standard-compliant *Web browsers*. These elements, of course, will be central in the experience presented in this paper.

3 Architecture

In Ambient Computing, there are several problems that should be addressed in order to build an effective system. First of all there should be a way for defining, discovering and identifying what is available in the surroundings and, of course, there should be a way for interacting with the found entities.

These requirements entail several others that can be summarized as the following:

- *Networking:* How can the entities that are present in a given environment exchange information among them?
- *Layout:* What is exactly the *surrounding* environment and how could it be defined and identified?
- *Addressing:* How can we address entities that are in a given environment?
- *Management:* How can an entity providing a service declare itself, and how other entities can be aware of it?
- *Interaction:* How can entities co-operate, and what kind of interaction entities can have in order to fulfill a given task?

The proposed architecture is presented in Figure 1 and, by having in mind the previous list, we will describe in the following sections its structure and how these requirements are met.

3.1 Networking

For our architecture we need a flexible networking infrastructure that should be widely supported and easy to manage. The choice is to use the TCP/IP networking on top of WiFi connectivity. Basically the target infrastructure will consist of TCP/IP (private) networks that are pervasively available on a physical environment through WiFi hotspots. An hotspot defines an environment called an *environment* that logically regroups all the devices connected to the hotspot.

This choice might seem restrictive because it presupposes, in order to be compatible with the infrastructure, the availability of built-in WiFi support on the

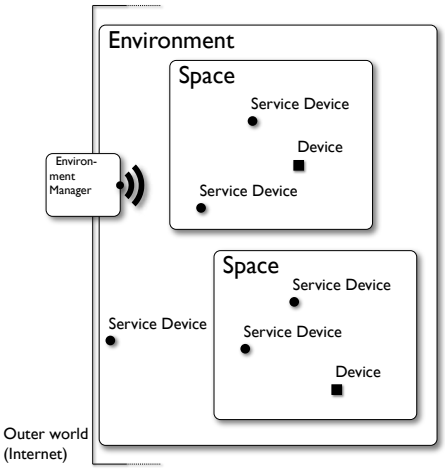


Fig. 1. The system architecture

devices. However many current-generation smart-phones, PDAs, and appliances have already this support built-in and in the future this support will become more and more widespread.

Since we want to deploy the architecture on top of the Web Platform, another assumption we are making is that the devices should be equipped at least with a *Web browser*, i.e., a component capable of understanding the `HTTP` protocol. This again should not be too restrictive: a WiFi-equipped device surely has a browser built-in (or installable as a third party software), so we can suppose that if a device is able to connect to the WiFi infrastructure, it will also able to *"browse"* what is available on that infrastructure as well.

Finally we don't make any assumption on the capabilities of the browser, unless for the fact that it should support basic XHTML capabilities like receiving and sending forms, and support *redirection*.

3.2 Layout and discovery

With the term layout, we refer to the definition of how a physical *environment*, i.e, the set of all the devices that are linked to a WiFi hotspot are organized. In order to do so we introduce the notion of *space*: a logical environment that is used to group devices that belong to the same "domain". The semantics of a domain is defined by the administrator and can, for example, reflect the organization of the physical environment (i.e., each *space* might correspond to a physical room), or to some administrative subdivision (i.e., a *space* for the Computer Science Department).

The entry-point of the system, from the point of view of a device, is the WiFi hotspot where it connects to. Once it is connected, an `IP` address is assigned to it and it can start to communicate with other entities present on the same network.

The problem now is how can this device discover the layout that is defined for a given *environment*? Discovery protocols such as `SLP` [12] or `SSDP` used in the *Universal Plug and Play* framework [13], use the underlying network topology for delivering multicast packets containing announcement information about the device present in the network. This announcement also contains information the actual *logical location* of a given resource (e.g., `printer-location: 'Room 3'`).

This approach has two consequences: first, a program listening for packets on a given multicast port should run on the mobile device; and second, the device should embed some logic in order to understand the messages and, for example, filtering only the messages that refers to devices in "Room 3". Since these messages are completely arbitrary, an ad-hoc application is needed for this.

In our architecture we would like to impose the least requirements on the device using the infrastructure (i.e., only they should be equipped only with a Web browser), so the previous approach is not suitable.

When a device wants to interact with the infrastructure it has to obtain an IP address from one of the available WiFi hotspots. This is usually done using the `DHCP` [14] protocol that configures the device's gateway, `DNS` and so on. So the idea is to configure the WiFi network for hosting a *transparent proxy* that will redirect all the traffic generated by the browser to a given server that can provide, despite of the actual address the user typed in her browser, an `XHTML` page that represent the entry point for all the *environments* registered at that hotspot.

This is basically the role of the *Environment Manager* pictured in Figure 1. What this page actually contains and how it is built will be detailed in Sections 3.4 and 3.5. Of course the content of this page will reflect the logical layout of the devices that are present.

This setup implements a transparent discovery mechanism that can be used without using any additional software. We would like to point out that what is discovered here is an entry point that will give directions on how to reach actual devices, while in `SLP` the target devices are directly discovered. Therefore, our approach implies that there is a central authority, i.e., the *Environment Manager* that is behind the WiFi hotspot, that should keep track, in a centralized way, of all the device that are present in a given environment. The procedure for doing so will be described in Section 3.4.

3.3 Addressing

When we reason about addressing the first thing to decide is what are the resources in our domain that needs to be addressed, and how these addresses are structured. In our architecture we will address two kinds of resources: *spaces* and *service devices*. The first ones are used for the logical organization of devices, in order to reflect their actual disposition in physical space or in administrative domains; the second ones are devices that provide the actual services. As we are

leveraging the Web Platform, we will use URIs for addressing resources, and the address template has the following form: http://environment/{space}/{device}[5].

From the address template it is clear that it is not possible to specify space nesting. In fact, in our architecture, an address like http://campus/building/-hall/room/printer would not be recognized.

However we allow the definition of a logical hierarchy of *spaces* by associating to each *space* a parent addressable using the special address http://place/{space}/parent.

The fact that *service devices* can be addressed using an URI, implies that every service device must have a web proxy that can be used in order to interact with it. This web proxy, that can be located on the devices itself or somewhere else, provides the actual interface, adapted to the Web Platform, and that will be used for controlling the device itself. Usually, as we will show in Section 3.5, this interface will consist of an XHTML page presenting a form with all the relevant attributes exposed by the device.

3.4 Management

In order to organize the layout of the *environment*, service devices need to be able to perform two sets of operations: logical *space* creation/removal/re-parenting, and device registration/unregistration/re-parenting in a given *environment*.

All these operation are carried out by leveraging the HTTP protocol.

Space operations are quite straightforward: a *space* is created or re-parented by PUTting the *space* name and the (optional) parent name to the *environment* entrypoint, whose location is given by http://environment. Re-parenting is done in the case where the *space* already exists.

A *space* is removed by DELETEing the *space* address.

The current mean by which they are performed are XHTML forms that are presented in documents that are served by the *environment manager*. These forms provide the interface for performing the previous operations, and are available as resources at well defined URI[6]. The attributes for identifying the *space* name and parent are arbitrary since they are provided through the form by the same entity (i.e., the *environment* entry point) that will re-interpret them when they will be received as POSTed data. This is fine is the POSTer is a human that fills out the form, however if we imagine to automatize this process, then these attributes should be well defined and immutable.

The attentive reader might notice that XHTML forms only supports the GET and POST methods[7]. However it is possible to overload the POST method by specifying the actual HTTP method in an hidden field of the form. In this way

[5] This is actually an URI template as defined in [15].

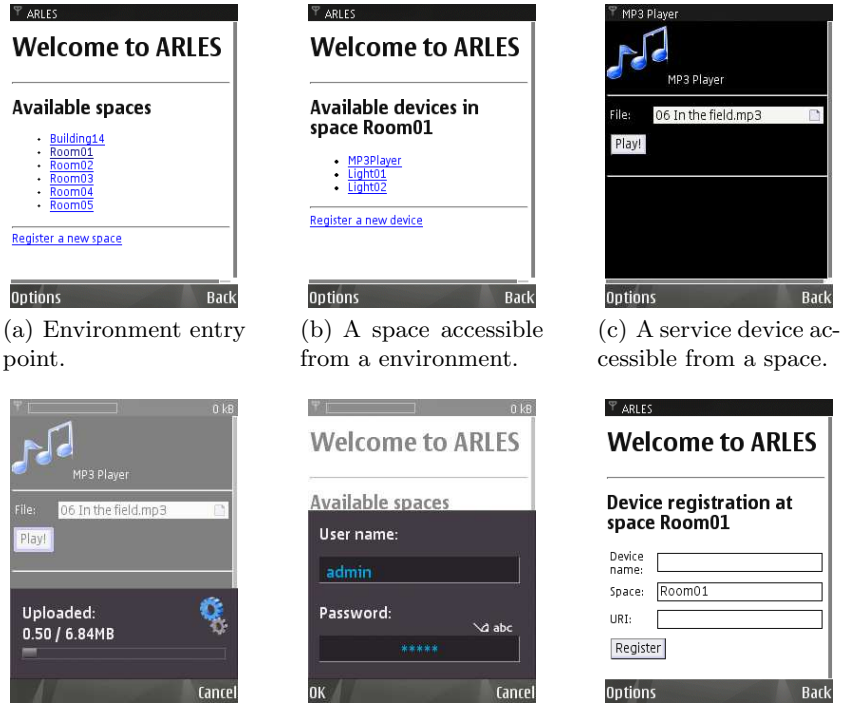
[6] In our system we publish the forms for registering *spaces* at http://environment/registration, and the forms for registering *service devices* at http://environment/{space}/registration

[7] This limitation will be overcome in XHTML5, but it will take some time before this standard will be widely adopted.

the POST request will be treated as if it were a request that used the HTTP method specified in the POSTed data.

Device registration, unregistration and re-parenting are done in a similar way. What changes is the set of attributes that are contained in the POSTed data. The device relevant attributes are: its name, the optional logical *space* where it is located, and the URI for its representation.

3.5 Interaction



(a) Environment entry point.

(b) A space accessible from a environment.

(c) A service device accessible from a space.

(d) Device communication.

(e) Authentication.

(f) Service device registration.

Fig. 2. Interaction with the system.

In this section we will present a classic interaction of a device with the system. This description is based on a real case study that we have implemented as a proof of concept, and that is available at [16]

A user enters a space and with its mobile phones looks for WiFi hotspots that are available. Once she has found a suitable one, she connects to it and opens her browser.

At this point she doesn't know what address to type in the address bar, however no matter what she will type the *Environment Manager* will intercept

all her requests and reply as if they were addressed to itself. Since many browser have a predefined home page that is displayed when the browser is opened, this operation will be totally transparent for the user and what she will see after connecting to the hotspot is just the *environment* entry point as presented in Figure 2(a). Of course this means that all the user will not be able to reach the "outer" internet, however this interception mechanism could be put in place only for the first user request. In this way the user will be able to bookmark the environment entry point page, and will be free to use the internet as usual afterwards.

From the environment entry point, the user can navigate the structure of the *environment* by following hyperlinks; for example she might go to the room she is currently. Once the *space* page is displayed, the user has a view of all the devices that are present in the given *space* (Figure 2(b)). A *service device* called "MP3-player" is available for playing music in the current room. Clicking on the *service device* will present the user an interface for operating the player. This interface is served by the *web proxy* associated to the actual MP3-player.

We point out that the actual *web proxy* could be located everywhere in the network. For example, it could be running in the data centers of the campus (i.e., in the "outer world" pictured in Figure 1). What happens when a device requests a `http://environment/{space}/{device}` address is that the *Environment Manager* redirects the device to a new address that is the one of the actual web proxy for the *service device*. This is possible because of the `HTTP` protocol that supports, as described in Section 2, redirection response codes.

Once the user has the device interface displayed on her mobile phone, she can select an MP3 file from her collection and send it to the *service device* for playing it. Again, this is done by leveraging the `HTTP` protocol that allows the user to `POST` data to a given address (i.e., the one of the *service device*). Figures 2(c) and 2(d) show this interaction whose effect is that of having the (local) MP3 file streamed to the player in the room: the user can then enjoy her favorite music on the HiFi system's speakers in the room.

We would like to point out also the fact that since all the resources are directly addressable, the user has the possibility of bookmarking them for later usage. So, instead of following the link-chain from the *environment* entry point to the current *space* and then to the *service device*, she can directly jump to one or the other.

Finally, the user might also register a *service device* by using the registration interface provided by the *environment manager*. This interface, that is presented in Figure 2(f) allows the user to bind, in a given *space* a *service device* to a given web-proxy, by giving its address. `HTTP` already provides authentication mechanisms that can be used to restrict this "administrative" task only to the users that are allowed to perform it (Figure 2(e)).

4 Conclusion

In this paper we have presented an experience in building a system for Ambient Computing that made use of the Web Platform as its reference architecture. What we presented in this paper is related to previous work presented in [17] and [18]. In fact there is a big overlap with CoolTown which proposes an infrastructure that is based on the Web Platform, exposes devices through HTML interfaces, and uses `URI` as the mean for identifying them.

In our system we wanted to stick to basic features available in the Web Platform, without relying on adding any-logic to devices. For example, in CoolTown, extra hardware is used for "sensing" device addresses provided by beacons, and to input them in the web browser (though these addresses might also be directly typed)

The principle we wanted to follow was to stick to the semantics of the `HTTP` protocol, that provide the uniform interface for accessing every device in the environment, and to build on top of it all the necessary infrastructure for manipulating and accessing the resources provided by these devices.

The experience has proven that this approach is very effective: our system has been built on top of reusable components (i.e., the Web Browser and the transparent proxy[8]), an *Environment Manager* that consists of less than 500 lines of code and a simple 10 lines Ruby script for implementing the MP3 Player proxy.

Of course our system is very limited at the moment, even when compared to the more advanced CoolTown one, however it is already good enough for a simple ambient computing environment.

The future work will consist in experimenting in the same direction, by integrating other Web technologies and standards in this "base system". For example a direction to explore is that of using syndication formats for handling what is available, and how it can be accessed. There is a lot of work in the Web community around those standard and we think that they can be fruitfully exploited for handling in a seamless and interoperable way device interactions in a pervasive system.

References

1. Sun Microsystems: Java SE technologies at a glance `http://java.sun.com/javase/technologies/`.
2. Google Inc.: Google Web Toolkit `http://code.google.com/webtoolkit`.
3. Ruby, S., Thomas, D., Hansson, D.H.: Agile Web Development with Rails, Third Edition. Pragmatic Programmers (2008)
4. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Interet Technol. **2**(2) (2002) 115–150

[8] Implemented using by a Squid web caching proxy, and a 10 lines script for URI redirection

5. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol – HTTP/1.1 (1999) http://www.ietf.org/rfc/rfc2616.txt.
6. IANA, Iternet Assigned Numbers Authority: MIME media types http://www.iana.org/assignments/media-types/.
7. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform resource identifiers (URI): Generic syntax (1998) http://www.ietf.org/rfc/rfc2396.txt.
8. Richardson, L., Ruby, S.: RestFul Web Services. O'Reilly (2007)
9. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (XML) 1.0 (fourth edition) (2006) http://www.w3.org/TR/2006/REC-xml-20060816/.
10. W3C HTML Working Group: XHTML 1.0 the extensible hypertext markup language (second edition) (2002) http://www.w3.org/TR/2002/REC-xhtml1-20020801/.
11. ECMA International: ECMAScript language specification (1999) http://www.ecma-international.org/publications/standards/Ecma-262.htm.
12. Guttman, E., Perkins, C., Veizades, J., Day, M.: Service location protocol, version 2 (1999) http://www.ietf.org/rfc/rfc2165.txt.
13. Goland, Y.Y., Cai, T., Leach, P., Gu, Y., Albright, S.: Simple service discovery protocol/1.0: Operating without an arbiter. (1999) http://www.upnp.org/draft_cai_ssdp_v1_03.txt.
14. Droms, R.: Dynamic host configuration protocol (1997) http://www.ietf.org/rfc/rfc2131.txt.
15. Gregorio, J., Hadley, M., Nottingham, M., Orchard, D.: Uri template (2008)
16. Mancinelli, F.: Leveraging the web platform for ambient computing: An experience - case study (2008) http://www.fabiomancinelli.org/web_ambient_computing.tar.gz.
17. Kindberg, T., Barton, J.: A web-based nomadic computing system. Computer Networks **35**(4) (2001) 443–456
18. Edwards, W.K., Newman, M.W., Sedivy, J.Z.: The case for recombinant computing. Technical report (2001) http://www.parc.com/research/projects/obje/.