Communications and Network, 2014, 6, 86-96
Published Online May 2014 in SciRes. http://www.scirp.org/journal/cn
http://dx.doi.org/10.4236/cn.2014.62011

Scientific Research

A Generic Platform for Sharing Functionalities among Devices

Remi Nguyen Van, Hideki Shimada, Kenya Sato

Department of Information Systems Design, Doshisha University, Kyoto, Japan
Email: hiroakigoto0624@gmail.com, hideki-s@is.naist.jp, ksato@mail.doshisha.ac.jp

Received 23 January 2014; revised 23 February 2014; accepted 8 March 2014

Copyright © 2014 by authors and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY).
http://creativecommons.org/licenses/by/4.0/

Open Access

Abstract

With so many potentially interconnected electronic devices in today's homes, manufacturers have to think of theirs as only one of the components involved in a general user experience, and not as an isolated device. Users are likely to be using several devices at the same time, either actively through immediate interaction, or passively by expecting devices to give them notifications when necessary, for example. Thus, having cross-devices functionalities is often necessary for a product to be really adapted to its usage situation. Moreover, just as we install software on computers, smartphones and tablets for additional functionalities to use their own hardware, it would be logical to install cross-devices software to use the combined hardware of several home devices for a better user experience. However, even though a number of technologies can be used to transmit data or commands between devices, UPnP being a widespread example, it is not possible to access the behavior of remote devices and add functionalities to them this way. Thus, when manufacturers design their products, there is no way for them to make full use of the other appliances at the user's home without developing and deploying specific software on each of them. In order to address this issue, this paper discusses a platform for generic development and on-the-fly deployment of applications on home devices. This system aims at letting device vendors deploy innovative features across devices in a home network, without requiring prior knowledge or control over devices already present in the user's environment. For this platform to be fit for consumer devices, it is designed to be cost-effective, use recent and widespread technologies, and be fast to implement and work with.

Keywords

Home Network, Sharing, Application, Deployment, Cross-Platform

1. Introduction

Let's imagine a phone manufacturer that wants his users to be able to receive notifications for incoming calls

How to cite this paper: Nguyen Van, R., Shimada, H. and Sato, K. (2014) A Generic Platform for Sharing Functionalities among Devices. *Communications and Network*, **6**, 86-96. http://dx.doi.org/10.4236/cn.2014.62011

and text messages on whatever device the user is currently using at home, like their TV or laptop. This vendor would need to develop a client application for the laptop, and have access to the TV's firmware, since displaying this kind of notification is not natively supported on those devices.

The objective of our research is to free the vendor from this requirement, by providing him with a platform that allows generic development and automatic deployment of his applications onto other home devices, eliminating the need to have access to those. In the first section, the required specifications of this system are explained; the proposed system designed to meet these specifications is described in the second section. The third section gives an evaluation of how well this system solves the original issues by studying a developed prototype. Finally, a comparison of the proposed system and other similar systems is made in the fourth section of this document.

2. Required Specifications

In order to solve this issue, a proposed system should:

2.1. Let Devices Communicate in a Generic Way

To be able to focus on the user experience provided by their devices without spending valuable time studying other potential devices in the user's home, manufacturers need to have a generic way to make their device communicate with other home appliances. This means that there must be one standard way to share functionalities that is compatible with any device supporting the platform.

2.2. Let Devices Share Any Kind of Feature

This system is intended to let vendors design innovative cross-devices features. Thus, devices supporting this platform should be able to receive any kind of content, not just standard messages. This genericity also means that no manual update should be needed on existing devices to be able to interact with a new device. Thus, manufacturers do not need to develop additional software to be installed on remote devices, as those are potentially already compatible with any functionality they want to make available.

2.3. Keep Devices Updated on Each Other's States

For devices to automatically interact with each other so as to create a user experience as a whole, they need to know which devices are active and can provide them with additional content at any time. This enables vendors to develop user experiences based on all the devices surrounding the user, without requiring any user interaction. The goal is for several devices to create a user experience as one system, rather than having the user manually connect each independent device. For example, if a user receives a call while watching TV, the TV should automatically be aware of the state of the phone, and could prompt the user for starting a video call; in this situation, the phone and the TV can be seen as one system automatically configured to receive calls.

2.4. Be Easy to Implement for Vendors

Device manufacturers would never choose to implement a feature if the development cost is too high compared to the potential benefit. Thus, making devices compatible with this platform and developing applications for them must be easy and fast.

Finally, learning how to use the platform must be simple for developers, as the availability of qualified developers for a platform is directly reflected on development costs. This means that the use of recent and widespread technologies must be preferred.

3. Proposed System

3.1. Basic Process

In the proposed system, which basic process is illustrated in **Figure 1**, a generic UPnP platform (referred to as "the UPnP platform") is deployed on remote devices. The platform takes care of receiving and executing compatible applications; those applications are written in generic code and packaged inside the vendor's device.

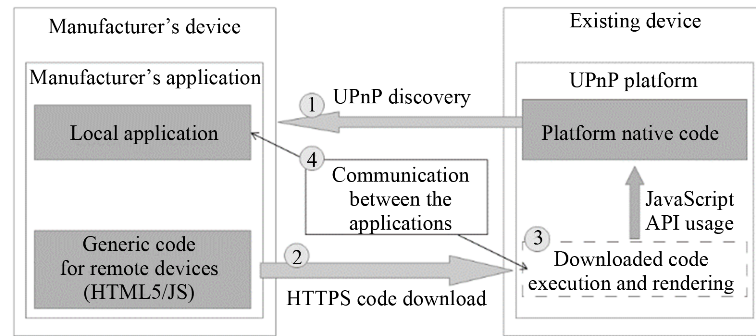

Figure 1. Basic operation of the system.

Once the platform finds the vendor's device through UPnP discovery, it downloads the application code from it; this way, client applications are deployed on-the-fly on compatible devices, and no manual update is required.

The language chosen for the generic applications to be downloaded and executed is HTML5/Javascript. A Javascript API supported by the UPnP platform on remote devices ensures that the JavaScript code can use lower-level functionalities not supported by HTML5, such as controlling the application window or sending UPnP commands.

3.2. Technology Choices

3.2.1. Device Discovery and Eventing: UPnP

UPnP is already very popular among device vendors, and is designed to enable devices to discover themselves and share their states in a generic way. Thus, it is very well suited for handling discovery. In the discovery phase, UPnP communicates a presentation URL for the service, which is used to specify the location of the client code to download. Additionally, once the discovery phase is completed and applications have been deployed, UPnP can be a convenient way for devices to communicate.

3.2.2. Code Download: HTTPS

Because the UPnP platform is going to download and execute code from other devices, it is necessary to have a safe method to check these devices are trusted. In our proposed system, we use HTTPS to download this code, and we validate the vendor's device's public key using a list of user-approved keys. This means that the user has to approve the vendor's device on remote devices the first time it is discovered. Once this is done, the generic client can be sure that every communication made with this public key originates from the approved device [1].

To let the user add trusted devices, the UPnP platform application can either use the UPnP Device Protection service [2] if the remote device supports it, or simply ask the user to press a hardware or software button to approve newly discovered devices.

Although this research mostly focuses on downloading the code from the remote device in the local network, it is noteworthy that it can also be downloaded from a remote server through the internet, as the SSL connection ensures it is retrieved from a trusted host. However, this option has not been studied into detail in our research in order to focus on a prototype with better reactivity and fully working on a local network only.

3.2.3. Code Execution and User Interface: HTML5/JS

HTML5 is getting more and more popular recently, and more and more devices—like smartphones, computers and tablets—have HTML5/JavaScript rendering capabilities. Applications are easy to develop and these technologies are well known among software developers, which make it easy for device vendors to use. Additionally, since the code is executed in a sandbox, it also ensures that even if the system were to be compromised, the potential damage would be limited.

3.2.4. Using Lower-Level Functionalities: JavaScript API

Since JavaScript does not provide ways to control the application's window or send UPnP commands for instance, a JavaScript API has to be available in the UPnP platform to let deployed applications access these functionalities. To develop our prototype, we defined a standard API for the client program entry point, window

management (fullsize and reduced mode, notifications) and UPnP commands. Other functionalities like letting client applications provide sound or video streams using the API can be defined (as very few browsers support WebRTC [3] for now), but most functionalities required to develop a rich application are already available in HTML5.

Depending on the generic client application, some functionalities of the API may not be supported; some devices may not be able to use UPnP or display notifications for instance. Thus, client applications have to adapt their behavior to the platform they are running on; our API provides an easy way to test whether a functionality is available on the guest UPnP platform, and react accordingly.

3.3. Architecture Summary

The general architecture of the UPnP platform is illustrated on **Figure 2**. This system is designed so that it is also easy to develop and embed the UPnP platform client on many devices. On devices where an HTML5 engine is included in the system and can be used for development (such as Blackberry devices, iOS or Android-based devices), this component is used by the UPnP platform to execute the downloaded code. This helps make it lightweight and easy to develop.

In case there is no HTML5 engine available on the platform, a library can also be used; in our prototype, we used the WebView component of JavaFX to develop such a client on a personal computer.

3.4. Requirements Summary for the Platform

Given our technology choices, devices must meet the following requirements to be able to run the UPnP platform client:

- Have local networking capabilities.
- Be able to fully support the UPnP protocol. In particular, the device must be able to send multicast messages. There is a number of libraries that developers can use in many languages.
- Be able to download files through HTTPS on the local network, and carry out custom certificates chec; several methods are discussed in the next section.
- Be able to run JavaScript code. An engine may be already available on the device, or support can be bundled with the application if the developer chooses to use a library.
- Be able to provide a custom JavaScript API in the JavaScript engine; several methods are discussed in the next section.

As for vendor's devices that use this platform, the following requirements must be met:

- The device must have local network capabilities and be able to fully support the UPnP protocol, just like the for UPnP platform.

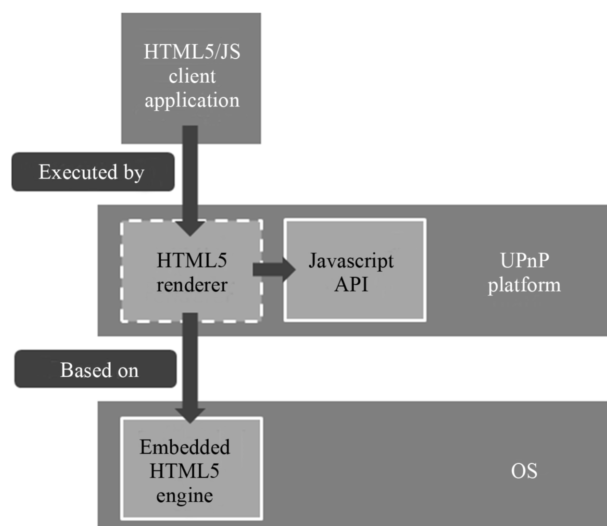

Figure 2. Architecture of the UPnP platform.

- The device must be able to serve files through HTTPS; many libraries can generally be used by the developer for this task.

3.5. Implementation Options

To easily develop the UPnP platform client, an existing HTML5/Javascript engine will generally be used. However, some special functionalities must be added to this engine: the Javascript API must be supported, and the SSL certificates checking used in HTTPS must be customized to use a list of user-approved public keys. A number of implementation options are possible for these two problems.

3.5.1. Custom Certificates Checking

1) Managing the engine's requests

With some engines, it can be possible to have direct access to the SSL certificate validation procedure used when loading a document through HTTPS. In this case, the engine can directly load the client application code from the remote device using a customized validation procedure as shown in **Figure 3**; this is the most simple implementation option. However, on many engines, modifying the SSL certificates validation procedure is not possible.

2) Using a local proxy

If the SSL certificates validation procedure cannot be customized, it is possible to embed a lightweight HTTP/HTTPS proxy in the UPnP platform client, so that requests can be handled manually. This process is illustrated in **Figure 4**.

In this case, the web engine loads the document from a local address using regular HTTP. As the proxy receives HTTP requests from the application, it downloads the requested resource from the remote device using HTTPS, verifying the certificates in the process. The resource is then delivered to the HTML engine. This method can be used on Android or iOS devices for example, and is the one we used in our JavaFX-based implementation.

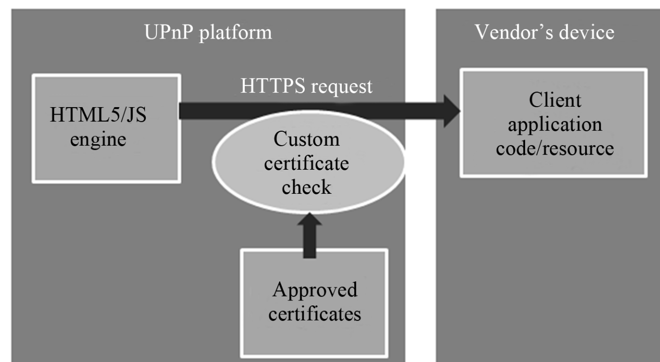

Figure 3. Checking certicates by directly managing the engine's requests.

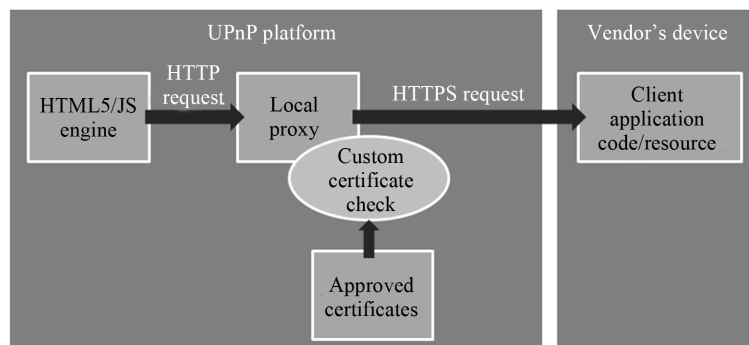

Figure 4. Checking certicates by using a local proxy.

3.5.2. Implementation of the Javascript API

1) Direct access to the Javascript engine

Some engines allow the programmer to inject native objects into the Javascript environment, so native code functions can be accessed through Javascript. In this case, the Javascript API can be easily created by implementing the required functionalities in native code, and then giving access to them by this technique. This is illustrated in **Figure 5**.

This method can be used on Android [4] and Blackberry [5] devices, as well as on PCs through JavaFX [6]; this is the one we used in our prototype.

2) Intercepting AJAX requests

Some engines do not give any way to call back native code from Javascript, although it is generally possible to run Javascript code from native code. In order to make the API available when using such engines, an additional Javascript file can be inserted in the document by native code, so that calls to the API functionalities will result in corresponding AJAX requests. Such requests can then be intercepted in the native code, either directly if the engine allows it or by using a local proxy (which can also be used for certificates checking).

This method can be used on iOS devices for example, as the iOS SDK does not provide a way to access the Javascript engine directly. It is illustrated in **Figure 6**.

4. Prototype and Evaluation

To evaluate the efficiency of such a system, we developed a UPnP platform client to use on a PC, and then developed a smartphone application that uses this generic client to:

- Let the user use his PC to browse through the phone contacts and send text messages (SMS) to them, see **Figure 7**.
- Display a notification on the PC when the phone receives a call, see **Figure 8**.

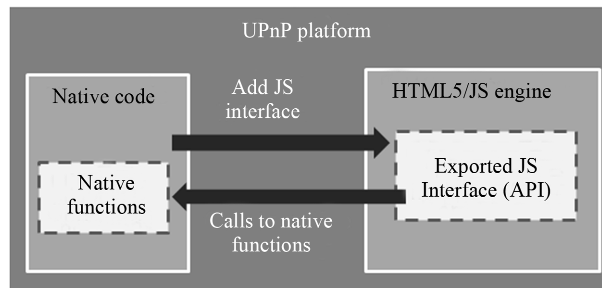

Figure 5. Implementing the Javascript API by direct access to the Javascript engine.

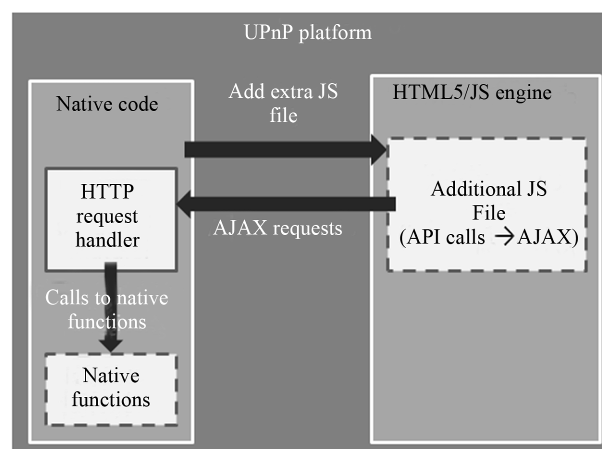

Figure 6. Implementing the Javascript API by intercepting AJAX requests.

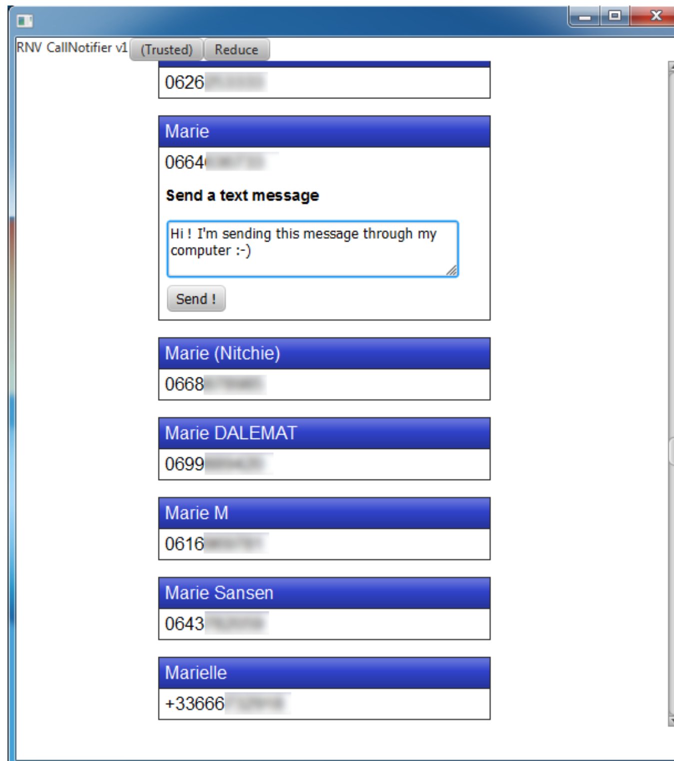

Figure 7. Using the computer client to send a text message (SMS) through the phone network.

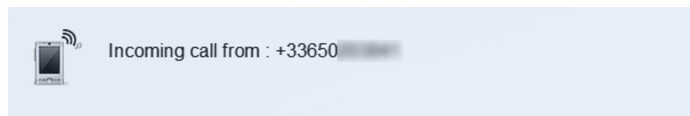

Figure 8. On-call notification shown in the top-left corner of the screen.

4.1. Generic UPnP Platform Development

The JavaScript API supported by the generic UPnP platform program of our prototype only includes methods for displaying notifications, and UPnP communication. These features should be supported by any implementation of the platform, regardless of the device on which it runs. However, the end result of those methods may differ depending on devices; for example, showing a notification on a PC can be done by displaying a message in a corner of the screen, while there often is a dedicated mechanism for notifications on smartphones. These different behaviors on different devices are obtained by executing the same JavaScript code: this ensures that applications written in this generic way are compatible with a wide range of devices.

In the future, we plan to define more functionalities in the API that may not be supported by every device: standard methods to display streamed videos for example. Thus, there is a set of API methods that cannot be supported in some implementations of the platform, depending on which device it is designed for. However, the API provides a way to check for the availability of those functionalities so that deployed applications can adapt their behavior to the device they are deployed on.

4.2. Development of a Smartphone Application Using the Platform

It is important to note that developing the smartphone application required no modification to the UPnP platform client installed on the PC, as it is a generic platform designed to receive and run any kind of application. Only HTML5/JavaScript code had to be included in the smartphone application, and is automatically uploaded to the PC. For example, to develop the on-call notification functionality, an application to send UPnP events

when receiving a phone call must be developed and deployed on the smartphone. Once this is done, showing a notification on the PC can be achieved by only including the code shown in **Figure 9** in the smartphone application.

One may notice that the subscribe To Events method of the API uses the name of a callback function for event subscription, and not a reference to it. While slightly less efficient, this is to ensure that this API method can be implemented on any system, as passing a reference to an object from JavaScript to native code is not possible when using some JavaScript engines.

When associated with the corresponding static HTML page, this code alone is fully functional and enables the PC to receive UPnP events from the smartphone and display notifications.

4.3. Testing Environment

Testing environments for the UPnP platform and the smartphone application are summarized in **Table 1** and **Table 2** respectively. The smartphone used for testing is much less powerful than most of the smartphones and tablets that are being manufactured today; this enables us to check that this system can be used on a wider range of embedded devices.

Although we used Java to develop our prototype, it is important to note that this is not a requirement, and any programming language or platform can be used as long as the required functionalities detailed previously can be implemented (see 2.4).

4.4. Testing Results

4.4.1. Reactivity

Table 3 shows the measurements made on the reactivity of the system.

After the UPnP discovery and HTTPS code download are completed, the communication between the two devices is responsive enough to seem like realtime to the user.

```
// Program entry point
function clientInit(api) {
   // Globally save the api object
   window.api = api;
   // Subscribe to UPnP events
   api.subscribeToEvents('callback');
}

// Called on UPnP events
function callback(state) {
   if(state.CallerNumber != '') {
      // Display incoming call phone number
      document.getElementById('caller_num')
         .innerHTML = state.CallerNumber;

      /* Show the contents of #notif
        * as a notification
        * using the specified stylesheet */
      api.showNotification('notif',
         'style.css');
   }
   else {
      // No call: hide notification
      api.hideNotification();
   }
}
```

Figure 9. JavaScript code sample for displaying on-call notifications.

Table 1. Testing environment for the UPnP platform.

eciveD	retupmoc potpaL
metsyS gnitarepO	64x 7swodniW
rossecorP	Core i7-2630QM @ 2GHz
yromem elbaliavA	36RDD MG
mroftalp tnempoleveD	2.2XFavaJ
tpircSavaJ/5LMTH	XFavaJ ni)enigne tikbeW)
UPnP library	2.0gnilC

Table 2. Testing environment for the smartphone application.

Device	HTC Desire Z smartphone
Operating System	Android 4.0.4 (ICS)
Processor	MSM7230 @ 800MHz
Available memory	512 MB
Development platform	Java (Android SDK)
HTTP server library	Jetty 8.1.9
UPnP library	Cling 2.0

Table 3. Time measurements on the system's reactivity.

UPnP discovery, code download	2 - 10 seconds
UPnP commands (list contacts)	<1 second
Eventing (incoming phone call	<1 second

Although UPnP discovery takes some time, it is made as soon as the device is powered on, so when the user introduces a new device into the local network, it is immediately synchronized with the others. This is necessary to ensure devices are aware of each other's states at any time, as discussed earlier. Thus, when the user starts interacting with an appliance, UPnP discovery most likely will already be complete and this device can immediately communicate with the others.

4.4.2. Ease to Implement

Table 4 shows the compressed source code size for all the programs. The compression method used to create an archive is LZMA. As it results in a very small archive, these measurements show that very little information needs to be provided to build an application making use of this platform. In other words, very little development work is needed. Moreover, in this case, the phone manufacturer only has to develop the smartphone application, and the HTML5/JS application code that is transferred to remote devices: the UPnP platform is supposedly already installed on remote devices.

5. Comparison with Existing Systems and Related Works

Table 5 summarizes the differences between our proposed system and existing technologies.

5.1. UPnP

UPnP-enabled devices can already send content to other devices and receive commands from them, as well as keeping them updated on their state through the discovery and eventing mechanisms [7]. However, these devices

Table 4. Compressed size of source code.

UPnP platform client (Java, PC)	6.57 kB
Smartphone application (Java, Android)	5 kB
HTML5/JS + CSS client code	2.45 kB
JavaScript client code only	1.7 kB

Table 5. Comparative chart for related technologies.

	UPnP alone	Web-based interfaces	Adaptive JINI	OSGi	Proposed system
Allows delivering any kind of functionality	×	o	o	o	o
Automatic discovery and initialization	o	×	o	o	o
Applications are very easy to develop	N/A	o	×	×	o
Does not require Java	o	o	×	×	o
Uses popular technologies	o	o	×	o	o

cannot handle every kind of content and commands: standard profiles for a fixed set of commands are defined, and devices are supposed to support a pre-defined set of profiles. Thus, they would need to be updated to support new functionalities, and UPnP alone does not address this issue, as manufacturers have no control over the software installed on existing UPnP devices in the home.

5.2. Web-Based Interfaces

When they want a wide range of devices to be able to access their product's functionalities, manufacturers sometimes rely on web-based interfaces. This is the case for web cameras or router configuration tools for instance. Although this method enables the product to display any kind of content, and is generic enough to be supported on all devices with a web browser, communication is only done in one way: a webpage has to be manually opened to send commands from a browser to the device. Thus, communication between devices cannot be done automatically without any user interaction.

5.3. Adaptive Jini

Jini (also called Apache River [8]) enables devices to execute applications provided by other devices, so that any Jini-compatible device can use all the applications deployed on the local network. In order to achieve generic application development, and thus avoid having to update the devices every time a new application is deployed, adaptive Jini has been proposed [9].

However, Jini is more intended as a way for one device to provide a service for other devices to use, rather than keeping the devices mutually connected. Additionally, Jini requires Java capabilities, which is not always available such as on TVs or smartphones. Finally, since it was introduced in 1998, Jini has not gained much popularity among multimedia device vendors, contrary to UPnP. Thus, it is unlikely for these vendors to decide to use a Jini-based system.

5.4. OSGi

In order to deploy applications on devices and communicate with them in a generic way, using UPnP with OSGi has been proposed [10], especially for home automation controllers such as to manage home lights. While this technique can be a good solution in many cases, it can have a number of drawbacks compared to our proposed system. Firstly, when dealing with multimedia devices like smartphones or tablets, HTML5 applications are often better suited and easier to develop than a java package. Secondly, devices do not always have java capabili-

ties, and cannot always use the OSGi framework. Finally, deploying an HTML5 application on these devices is simpler, since it only means rendering it; thus, it is easier for them to embed our HTML5-based UPnP platform rather than an OSGi client.

6. Conclusions

In this research, we have proposed a generic UPnP platform to be deployed on home devices so that device manufacturers can develop innovative cross-device functionalities using this platform by only working on their own device.

We have developed such a client on a computer, and demonstrated its efficiency by making use of a smartphone's functionalities through it. Even though the functionalities shared here are limited and only for demonstration purposes, this platform can already be used in many consumer products applications. As HTML5 enables displaying almost any kind of content, any situation when additional content needs to be displayed on another device's screen can be realized using our platform: IP cameras used for security or to monitor infants can automatically display notifications and even provide live video feeds in case of unusual activity. Multimedia devices can share advanced playback controls, display the current track information or suggest related artists for example, on any device compatible with the platform, even if they do not have a touch panel or even a screen themselves. All these features, among many others, can be built very quickly based on our proposed system.

References

[1] IETF Network Working Group (2013) Rfc2818:Http over tls. https://tools.ietf.org/html/rfc2818

[2] UPnP Forum (2013) Device Protection: 1 Service. http://upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service.pdf

[3] W3CWebRTC Working Group (2013) Webrtc 1.0: Real-Time Communication between Browsers. http://www.w3.org/TR/2012/WD-webrtc-20120821/

[4] Android Open Source Project (2014) Android API Reference, Package Android. webkit.WebView http://developer.android.com/reference/android/webkit/WebView.html

[5] BlackBerry (2014) BlackBerry Java Application Development, Class Browser net.rim.blackberry.api.browser.Browser http://www.blackberry.com/developers/docs/6.0.0api/net/rim/blackberry/api/browser/Browser.html

[6] Oracle (2014) Java Platform JavaFX, Class JSObject, java.lang.Object http://docs.oracle.com/javafx/2/api/netscape/javascript/JSObject.html

[7] UPnP Forum, (2013) Upnp Device Architecture 1.1. http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[8] Apache Software Foundation (2013) Apache Jini Specificationsv2.1.2. http://river.apache.org/doc/spec-index.html

[9] Kadowaki, K., Koita, T. and Sato, K. (2008) Design and Implementation of Adaptive Jini System to Support Undefined Services. Master's thesis, Department of Information Systems Design, Doshisha University,

[10] Donsez, D. (2007) On-Demand Component Deployment in the UPnP Device Architecture. Consumer Communications andNetworking Conference, Las Vegas, 920-924.