A Support to Multi-devices Web Application

Xavier Le Pallec, Raphaël Marvie, José Rouillard, Jean-Claude Tarby

Laboratoire d'Informatique Fondamental de Lille (LIFL)
Université Lille 1
59655 Villeneuve d'Ascq Cedex, France
tel. : (33)3 20 43 32 70 fax : (33)3 20 43 32 01
{xavier.le-pallec, raphael.marvie, jose.rouillard, jean-claude.tarby}@univ-lille1.fr

ABSTRACT

Programming an application which uses interactive devices located on different terminals is not easy. Programming such applications with standard Web technologies (HTTP, Javascript, Web browser) is even more difficult. However, Web applications have interesting properties like running on very different terminals, the lack of a a specific installation step, the ability to evolve the application code at runtime. Our demonstration presents a support for designing multi-devices Web applications. After introducing the context of this work, we briefly describe some problems related to the design of multi-devices web application. Then, we present the toolkit we have implemented to help the development of applications based upon distant interactive devices.

ACM Classification: H.5.2 [Information interfaces and presentation]: User Interfaces. - User interface management systems.

General Terms: Design, Human Factors

Keywords: Interactive devices, toolkit, web application

1. Context

Nowadays we can find a lot of devices to interact with using computer systems. Each week, web sites like *engadget* or *gizmodo* unveil new types of devices, as well as variation of existing ones. Often, these devices are low-cost and come with an API. Their use for every day life applications currently increases. This is a trend that have been amplified by the Wii console and smartphones, as well as their extended interactive abilities (speech recognition, GPS, compass, webcam, accelerometer). Such abilities represent major features for pervasive computing.

A few months ago we have started the MINY project (*Multimodality Is Nice for You*). It mainly aims at studying the benefits of applying Model Driven Engineering (MDE) to the design and the development of multimodal application. We focus on platform that allow the construction of multimodal web applications. The reason of this choice is that web applications can run on a lot of terminals, do not need heavy installation steps and can be modified at runtime thanks to reflexive properties of Javascript. We also focus on platforms which deal with affordable devices (<500€ and stable drivers) and which are easy to setup. With such a multi-devices support, programmers could combine devices to create modalities. From now, we will use term «multi-devices» rather than multimodal in order to be more precise.

Part 2 presents the constraints that a «software support» has to respect in order to match our requirements. Part 3 presents the platforms we have studied based on these criteria. Part 4 presents the lightweight toolkit we are developing to help developer create multi-device web applications, as no existing platform does so matching our constraints. Finally, we conclude by presenting a web application for home automation.

2. Designing a multi-devices web application

We have identified six constraints that a «platform» (toolkit, middleware, support) has to respect in order to be an efficient software support for designing multi-devices Web applications.

(C1) Multi-languages support. Programming a Web application (in Javascript) that uses a Wiimote, RFiD reader and X10 adapters requires to handle several programming languages such as Java for Wiimote, and C# for mir:ror.

(C2) Devices access programming. In addition to the diversity of programming languages, developers also have to tackle different programming patterns like active waiting loop (mir:ror RFiD reader), file managment (Interactive Vocal Server), HTTP request (Roovio robot). Often developers implement an additional layer to ease devices access (to gain in readability).

(C3) Network heterogeneity. If an application is distributed on several kinds of terminals (like computers and smartphones), it may involve several kinds of networks (wifi, ethernet, 3G) as well as firewalls. This implies some restrictions like avoiding sockets.

(C4) Web browsers. Using a remote interactive device from Javascript within a Web application is currently not easy. This requires a bijective communication with remote processes (driving the device). Such communication is a current major limitation of Web browsers.

UIST'10, October 3–6, 2010, New York, New York, USA.
ACM 978-1-4503-0271-5/10/10.

(C5) Traces. A main functionality for an interactive system is the support for «undo». Trace management is an efficient way to address this concern as well as providing feedback.

(C6) Drivers. When a developer aims to exploit devices, she has to use associated drivers. Drivers discovery is generally time-consuming.

3. Current solutions

There are different kinds of solution that we could have used to address our concerns.

Dedicated environments. Projects like OpenInterface (http://www.openinterface.org) or WComp (http://rainbow.essi.fr/wikiwcomp) provide powerful platforms to allow the design of new modalities and to compose interactive components. The main drawback of such platforms is their laborious parameterisation during installation and maintenance. Finally, access from web pages is generally missing (C4). **Middleware.** Message oriented middleware (ex: JMS, IVY) or object oriented one (RMI) generally rely on the use of sockets (C3). It is still possible to set up HTTP tunnel or install some plug-ins to address this problem. But installation is complex. In addition, using such middleware within a web page is only possible through rare plug-ins (C4). Being unable to subscribe to events also prevent the use of service oriented middleware. **Component platforms.** Component platforms may be an interesting solution for our context. Some existing platforms (like Frascati, http://frascati.ow2.org) support several communication protocols (socket, HTTP) and different kinds of middleware (object, message, service). However, as they are not targeting interactive software, existing ones do not provide a library of interactive components (C2-C6), traceability (C5), and access from web pages (C4) are generally missing.

4. Our proposal

To design multi-devices web applications as easily as common applications based on mouse-keyboard-screen interactions, we propose a toolkit (http://www.lifl.fr/miny). It is made of five parts: an http-based message bus (WSE) and associated API in javascript, java and C#; session-oriented communications with traces persistence; a message format targeting to interactive devices; a stub/skeleton generator for devices; and adapters for interactive devices.

WSE = http-based message bus. We have implemented a COMET-like (http://cometdaily.com/) bus that we call WSE (Web Server Event). We chose *long polling* among possible COMET methods. Messages are JSON objects (JavaScript Object Notation). This message-oriented bus can be accessed from JavaScript, Java and C#. Finally, a web server and PHP support is all one needs to install WSE. Installation is limited to copying a directory on the web server. The following javascript code shows how to send a message or to listen to upcoming ones.

```
wse.joinSession("UIST2010");
wse.sendMessage({day : "Sunday ",
   object : "Doctoral Symposium"});
a = {};
a.newMessageReceive = function (message)
{ alert("A message has been received "
         +message); };
wse.addListener(a);
```

Message format. WSE is only a message bus and is not specific to interactive device access. We have defined a format for messages «coming from or going to» interactive devices in order to easily classify and process them : the location of the device (location/locationParams), its type (object/objectParams) and the action/event concerned by the message (action/actionParams).

Stub/skeleton generator. Device interactions are performed through method invocation and event subscription (to address constraint C2). The method is associated with only one action, and the object corresponds to only one device (i.e. not a type of device). Subscribing for events from a device follows the observer/observable pattern. Relying on the message format and this pattern mapping, a generator takes the description of devices (methods, events) as inputs and produces corresponding classes that encapsulate (un)marshalling of messages on the WSE bus. The following Javascript code shows X10 description file and the use of generated code.

```
{ name : "X10",
  constants : { ... },
  methods : { switchOn : {}, switchOff : {} }
}
-------------------------------------------------
manager = new Manager("uist2010");
X10 = manager.getX10 ("328", "Xavier", "Lamp");
X10.switchOn();
```

Adapters for interactive devices. Our platform currently include WSE adapters for 4 interactive devices : RFiD reader, X10, WebCam IP, BCI Epoc headset. Multitouch tablets (under Windows), Wiimote and Android mobiles (compass and accelerometer events) are to be included soon. QRCode reader and Speech recognition will come later.

In the video, we show a short example using 3 terminals: a computer connected to a X10 adapter (a lamp and a fan are plugged), a computer connected to a RFiDReader and a smartphone (Android). The video explains how to develop a web application to be operated from the smartphone.