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 run on a lot of terminals, do not need installation, and can be modified at runtime thanks to reflexive properties of applications. Such abilities represent a major feature of pervasive computing.

Few months ago we started a project named MINY (Multimodality Is Nice for You). It mainly aims at studying the benefits of applying Model Driven Engineering (MDE) to multimodal application design. We are particularly interested in two aspects: 1) applying techniques related to model transformation and aspect oriented modelling to improve participation of end users in interaction design processes through specific/simple metamodels 2) using model templates to enhance crystallisation of good practices concerning the use modalities.

Usual MDE processes may be seen as two-steps: defining models and mapping them to technical platform(s). Concerning the use of platforms to construct multimodal application, we focus on those that allow the construction of web applications because such applications can run on a lot of terminals, do not need installation 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 install. With 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 constraints (that we have defined) a «software support» has to respect in order to match our context. We have further studied software supports to detect «multi-devices platforms» and we have not found one which installation was easy, integrating access to a lot of device and addressing web applications (part 3). For this reason, we are currently developing a light platform which respects previous constraints. This toolkit allow developers to easily program a multi-devices web application (part 4). Finally, we conclude by a little web application of home automation (part 5).

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 application.

(C1) Multi-languages support. Programming a Web application that uses a Wiimote, RFID reader and X10 adapters requires to handle several programming languages. It can be for example Java for Wiimote, C# for mir:ror (RFID reader), Javascript for Web application.

(C2) Devices access programming. In addition to the diversity of programming languages, developers also have to tackle different programming patterns : active waiting loop (mir:ror RFID reader), file changes (Interactive Vocal Server), HTTP request (Roovio robot)… Generally developers end implementing an additional layer to homogenise devices access (to gain in readability).

(C3) Network heterogeneity. If an application is distributed on several terminals (like computers, smartphones), this may involve several kinds of networks (wifi, ethernet, 3G) and firewalls. This implies some restrictions: for example, using sockets is not efficient (mainly because of firewalls).

(C4) Web browsers. Using interactive devices from Web application is currently not easy (Javascript code cannot access to all devices). Future versions of Web browsers will allow JS code to access devices like GPS or accelerometer. However this does not concern distant devices (i.e. on other terminal). A non-bijective communication with other process is a major problem of Web browser. Server-based events, currently present in Opera, or HTML5 Web sockets address this problem. In our context, this is an additional technical aspect that developer has to deal with.

(C5) Traces. A main functionality for an interactive system is the support for «undo». Traces management is an efficient way to address this concern. Moreover, traces is also useful for debugging.

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

3. Current solutions

There are different kinds of solution that we could use to address our problem. We compare some of them to our previous concerns.

Dedicated environments. Projects like OpenInterface [1] or WComp [2] provide powerful platforms to allow the design of new modality (for the first one) and to compose interactive components (for the second one). The main drawback of such platforms is...
related to their laborious parameterisation during installation and maintenance. For that reason, these platforms remain inaccessible to many developers. 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. This refers to (C3). It is still possible to set up HTTP tunnel or install some plug-ins addressing this problem. But once again, environment installation gets more 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 [3]) support several communication protocols (socket, HTTP) and different kinds of middleware (object, message, service). However, as they are not targeting interactive software, they do not provide a library of interactive components (C2-C6). Traceability (C5) and access from web pages (C4) are generally missing too.

4. Our proposition

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:

1. A http-based message bus (WSE) and associated API in javascript, java and C#.
2. Session-oriented communications with traces persistence.
3. A message format targeting to interactive devices.
5. Adapters for interactive devices.

**WSE = http-based message bus.** We have implemented a COMET-like bus [4] that we call WSE (Web Server Event). Such bus are generally dedicated to web pages. As we focus on interactive devices whose drivers are generally not accessible in javascript, we also provide an API in Java and C#. We chose long polling among possible COMET methods. A messages is a JSON object (JavaScript Object Notation).

Only a web server and PHP support are required to install WSE. Installation is reduced to copying a directory on the web server. We choose not to create a standalone WSE server in order to avoid conflict on port 80 with a possible existing web server. We adopt PHP support because a lot of web hosts propose it.

The following javascript code shows how to send a message or to listen to upcoming ones.

```javascript
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);
```

Equivalent code in Java and C# may be found on our web site.

**Session-based.** As shown in the above code, using WSE begins with joining a session (or creating if necessary). All messages within a session are stored in a file which is named as the session (http://server_url/WSE/traces_files/name).

**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 such kinds of messages.

- **action:** the expected action (ex: `switchOn`) or name of the event (layDown for a RFiD reader).
- **actionParams:** arguments of action or event.
- **object:** type of device (ex: X10, RFiDReader).
- **objectParams:** optional details about the device (ex for X10: lamp, fan).

Rather than automatic identifiers, we choose to use explicit identifier which indicate where is the device.

- **location:** points to the place where the device is (ex: ‘Office 328’).
- **locationParam:** details the location in the previous place (ex: ‘Desk of Xavier’).

**Stub/skeleton generator.** To address concerns related to «devices access programming», requesting a device to do an action is done through invoking a method on an object. 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 is done by using 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.

```javascript
{ 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.

5. Conclusion

Our toolkit allows developers to program applications that use interactive devices. Applications can be coded in Java, Javascript (web browser) and C#. The main goal of this toolkit is to facilitate development of multimodal web applications.

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. It can switch on (and off) the lamp, the radio or the fan. It also shows how to change background color according to RFID reader events.

References