Widgets Mobility

Jean Le Feuvre Cyril Concolato Jean-Claude Dufourd
Telecom ParisTech; Institut Telecom; CNRS LTCI
46, rue Barrault
75634 PARIS CEDEX 13
{lefeuvre, concolato, dufourd}@telecom-paristech.fr

ABSTRACT
Widgets are becoming a natural way to design small, intuitive application, and their usage is becoming widespread on both the desktop and the mobile world. However, most widgets systems are web-centric and cannot address service mobility or application migration without heavy workarounds. In this paper, we describe a mechanism enabling the communication between widgets and their environment (other widgets, remote services). The proposed solution provides mechanisms to handle application and service mobility while retaining compatibility with existing Widget technologies. A description of an implementation is also provided.

Categories and Subject Descriptors
H.5.3 **[Information Interfaces And Presentation]**: Group and Organization Interfaces – *Asynchronous interaction, web-based interaction;* I.3.2 **[Computer Graphics]**: Graphics Systems – *Distributed/network graphics*

General Terms
Design, Experimentation, Standardization.

Keywords
Application Communication, Mobility, Widget.

1. INTRODUCTION
With the ever-growing deployment of mobile communicating devices, applications are carried across various environments as the user moves, either benefiting from the environment or enriching it with their own services. Additionally, users are constantly switching devices, using TV sets, mobile phones or desktop computers, and expect to have their applications on every device they use. In this paper, we address the mobility, in environments constantly modified by near-by devices, of a particular kind of applications: widgets.

Widgets are small interactive multimedia applications that can be found on desktop computers, mobile devices or, more recently, TV sets. Whether embedded in a web page or used as standalone applications, most if not all widgets systems are designed in the same way. A configuration file (sometimes called manifest) provides metadata about the widget and standard web technologies such as (X)HTML, SVG, JavaScript and CSS are used to display the widget. The standardization of these common practices is in progress by the W3C [1]. Such widgets are of particular interest in application mobility as they are designed with standard, platform-independent technologies.

Mobility 2009, Sep 2-4, Nice, France Copyright © 2009
ACM 978-1-60558-536-9/00/0009……$5.00

Typical widgets present information and media retrieved from the Internet, such as weather forecasts, news, stock quotes, pictures from community web sites, … Such widgets rely on mechanisms similar to the XMLHttpRequest API to query textual content or media from a server in pull scenarios, or through data streaming (video, …) for push scenarios. In both scenarios, the location of the server is known by the widget at authoring time.

Other widgets are used to present information about the device they run on, such as time, CPU, memory usage, battery, network or mailbox status. They rely heavily on proprietary JavaScript bindings with the system, even though some efforts are being made to achieve interoperability on these extensions.

As we can see, existing widgets systems such as Google Gadgets, Microsoft Gadgets, Opera Widgets, Apple Dashboard or Yahoo Widgets allow communication with external entities only in cases where the entity is known at the widget authoring time (remote server, multicast address, local service binding …). While sufficient for web-based scenario, this approach has some limitations when considering widget mobility, in particular in the home domain. Indeed, in such scenarios, devices - and therefore the associated services - may appear and disappear at any time, with changing IP addresses. There is therefore a need to define how widgets can be authored without knowing the exact address of data sources.

In this paper, we present a method for the creation of widgets compatible with mobility scenarios, based on the definition of a service abstraction model and on tools to dynamically associate widgets with network services. In particular, the proposed method supports asynchronous messages exchange between a widget and its sources. Finally, this paper shows how the proposed method allows widgets to be transferred between devices.

The rest of this paper is organized as follows. Section 2 presents the scenarios we want to address and derives some requirements. Section 3 describes related work in this area. Section 4 details our contribution, describing in particular service abstraction, widget discovery, interfaces between the widget and the services. Section 5 presents some results and gives some implementation details. We conclude this paper and present future works in Section 6.

2. SCENARIOS AND REQUIREMENTS
In this paper, we address the following scenarios. Imagine a user, Stanley, watching his pictures on his mobile phone with his favourite image gallery application, implemented as a widget. Stanley visits his girlfriend and finds a large LCD TV in his surrounding environment, on which the images would look better

and can be shared with his girlfriend; he therefore pushes his personal gallery widget to the TV. The widget on the TV then requests the images from the mobile phone and displays them. In this scenario, a fixed device (the TV) controls a service on a mobile device.

Later that day, Stanley decides to watch a movie and uses his phone to control the lights, DVD players and other fixed devices by means of widgets.

From these two scenarios, we set the following requirements:

- It should be possible to interface widgets with remote services not localizable at authoring time, to send control messages to them or to retrieve data or notifications from them.
- It should be possible to push a widget instance from one device to another.

In this paper, we propose a system that does not require modification to existing devices and network stacks, and does not modify the existing widget architecture either. We target a generic method working with most multimedia description languages (HTML, SVG) and describing most of the existing communication services (UPnP, Bonjour, Web Services …).

3. RELATED WORK

Application mobility is a topic of interest for many researchers, most of the time with a strong focus on client-server models (Web 2.0, web services). Regarding application mobility on CE devices, Cesar et al. [2] envisage the usages of a secondary screen in an interactive TV environment. The mobile device is used for example to display the electronic program guide and control the TV channel. However, they do not envisage a technical solution. We believe these usages are perfectly achievable with the mechanisms proposed in this paper.

From a commercial point of view, there are several solutions which try to address scenarios similar to those described above. For example, it is already possible to use an iPhone as a remote control for an iTunes application located in the same network. This is very similar to what we want to achieve. However, in this case and many others, the controlling application is a native application, not a widget; therefore, the mobility of that application is not possible.

We can also mention the work done in the Web4CE [3][4] standard. This standard defines how to combine Web technologies like HTML, CSS and JavaScript with UPnP and defines a UI service to access web applications on CE devices in a client-server model. We differ from this approach because we do not define new services for UI but rather map existing services to existing UI technologies. Furthermore, our approach is not server centric and we intend to be agnostic of the underlying protocol and of the presentation format.

4. PROPOSED APPROACH

This section describes our contribution. We first present an abstraction of services with which a widget may communicate. We then describe how service discovery is handled in our proposal, how messages are exchanged between the widget and the external entity using the service abstraction. Finally, we discuss how widgets mobility is achieved.

4.1 Service Abstraction

In our approach, a service is seen as a set of messages that may be sent by or received from the remote entity. Each message can additionally trigger and/or require a potential reply.

Web-based communications with remote entities are usually asynchronous processes where data packets are exchanged over the network, and can not rely on a threading model. This can be illustrated by the heavy use of AJAX programming in existing web sites. Supporting synchronous handling of messages within a widget would imply suspending its processing until the response is received. This may freeze user interactions or communication with other services, which is not acceptable in terms of user experience. Another solution could be to restrict messages to one-way only communications (no reply), but it would be very hard to map to existing service classes, e.g. UPnP, where most messages have both input and output parameters. We therefore only consider asynchronous messages and provide optional callbacks in the widget to handle replies.

Each message is defined as a set of input and output parameters representing the data being exchanged between the widget and the remote entity. In order to support as many service descriptions as possible while keeping the solution generic, the message syntax is kept simple and does not describe complex object structures, only supporting simple data types like string, Boolean or number. When complex parameters are exchanged (e.g. UPnP Browse request response), we consider the parameter as a string to be further interpreted by the widget (e.g. DIDL XML parsing).

In order to be able to describe a service, we assume that service messages and their parameters are labeled, either through existing description documents (such as SDCP in the UPnP environment or WSDL for Web Services), or through naming conventions (for example in the service technical specification). Each message and its related parameters are then identified by their name, and each service is identified by a URN, whether existing (such as "upnp:…"), or yet to be defined. Messages and parameters can be in input (from external entity to widget) or output (from widget to remote service) direction. An example of the syntax used is given in Section 5.

We should note here that this service abstraction is applicable to services provided by remote servers, local entities and even to widgets providing services. A Widget can indeed be viewed by another widget as a regular service, enabling them to communicate with each other or to act as proxies of other remote services.

4.2 Service Discovery and Binding

In our system, services can be discovered in several ways. Services exposed by other widgets are discovered when these widgets are loaded. Services exposed by other devices in the network are typically discovered using UPnP or ZeroConf protocols. However, in a mobile environment, devices (and therefore the services they provide) may appear and disappear at any time. In order to communicate efficiently with a remote entity, a widget shall therefore be aware that a service is becoming available or not.

For a widget to interface with a service, two solutions are possible:

- Wait for the indicated service to be discovered by the widget manager before running the widget, and shut down the widget when the service disappears;
- Run the widget and define a mechanism to inform the presentation when a service appears or disappears.

The first solution is not elegant, since a widget could disappear from the screen without the user consent. It is not practical either since a widget could have interfaces with different services not all present at the same time. We therefore choose the second approach, allowing a widget to be running even when none of its interfaced services have been discovered. The signaling mechanism between the service discovery and the multimedia scene is called the "binding" of an interface.

In our proposal, the service discovery and binding is handled by the widget manager. Once a service is discovered, the widget manager checks all widgets interfaces. A widget interface matches the service if the following conditions are validated:

- The type (URN) of the interface is the same as the one of the service;
- Each message declared in the widget interface is supported by the service;
- Each parameter of each message is understood by the service. Note that there may be parameters from the service not used by the widget interface.

For each validated interface, the widget manager sends a bound event to the widget when a new service of the matching type becomes available, and sends an unbound event when the service disappears. This solution is flexible enough to let the widget author decide how (un)connected services should be handled within the widget. This allows for instance disabling a button in the widget if a service is not available and enabling the button upon binding of the service. It also provides the ability for the author to handle multiple services of the same type, typically to control with a single widget multiple devices such as lights.

4.3 Message Exchange

For each message and parameter of the service interface, we associate a widget presentation construct to be triggered when an input message is received or to be monitored in order to send a message. With scene representations such as HTML or MPEG-4 BIFS, this scene construct can either be a JavaScript function name (foo), an attribute of an element (image.width) or an event on a particular element ("image.click").

In order to notify the presentation of service availability, we also defined two pseudo messages in the interface declaration called "bindAction" and "unbindAction". These messages have no associated parameters, and are called whenever the associated service becomes available or unavailable (see example 1).

For each output message and parameter of a newly bound interface, the manager creates listeners for the associated widget presentation constructs. When such construct triggers an output (message or reply to an input message), the manager collects all values of the output parameters listed in the interface and calls the associated service. When an input message or a message reply is received, the manager modifies the scene construct associated to each input parameter of the message and triggers the associated scene construct.

Our solution relies on the manager implementing the inner workings of the service protocol used (e.g. UPnP Device Architecture) but having no special knowledge on the service itself (e.g. UPnP Media Renderer). The main advantage is that once a service protocol is supported in a widget manager, any new service can be controlled provided there is a widget for this task.

4.4 Widget Migration

As stated in our requirements, the ability to move an application from device to device is fundamental in mobility scenario. The choice of platform-independent languages in widget systems provides the bases of the application mobility.

Our solution provides mechanisms to migrate widgets. This is achieved by making all target devices discoverable, exposing a URL exchange service very similar to the UPnP AVTransport service. The device initiating the transfer of the widget integrates a file delivery mechanism (typically a very simple HTTP server) and is able to discover these services. Upon decision to migrate a widget, the user is proposed a list of compatible devices. The widget manager simply instructs the selected device service about the location of the widget to be migrated.

Our solution also provides a mechanism for a device to announce itself along with a widget which can be used to control its own services. It uses the underlying service discovery protocols to embed a URL pointing to the widget.

5. RESULTS

5.1 Example

Our proposal relies on [1] for the description of the widget configuration file. We extended the syntax to declare the services supported by the widget. Each supported service is declared in an <interface> element, and each service message is declared in a <messageIn> or <messageOut> element.

The following example shows the part of the configuration document that declares the communication capabilities of a widget designed to display and control the status of a simple UPnP SwitchPower service.

```
<interfaces>
  <interface type="urn:schemas-upnp-org:service:SwitchPower:1"
bindAction="button1.boolean" unbindAction="button2.boolean">
    <messageOut name="SetTarget"
                outputTrigger="elt_1.boolean">
      <output name="newTargetValue"
              attributeModified=" elt_2.boolean"/>
    </messageOut>
    <messageIn name="Status" inputAction=" elt_3.activate">
      <input name="Status"
             setAttribute=" elt_4.boolean"/>
    </messageIn>
  </interface>
</interfaces>
```

Example 1: Usage of the service abstraction syntax for interfacing a widget with a UPnP Light device.

The interface element declares that the widget is capable to communicate with a service of the specified type. The `name` attribute of all elements in this fragment comes from the SCPD XML description of the UPnP SwitchPower service. The `messageOut` element declares when and how to generate

messages going from the widget to the device. The message is triggered when the attribute `boolean` of the element `elt_1` is modified. The value of the output parameter `newTargetValue` is taken from the attribute `Boolean` of the element `elt_2`. The message `SetTarget` is then sent to the UPnP service.

The `messageIn` element declares how to process messages coming from the device carrying the information about the status of the device: on or off. When the message `Status` is received by the widget manager, the value of the `Status` parameter of the message is copied to the attribute `boolean` of the element `elt_4` and the event `activate` is sent to the element `elt_3`.

In this example, both messages are connected to attributes of elements, thus allowing a script-less implementation. It should be noted that an input message can also be connected to a script function and an output message can be triggered from a script.

5.2 Architecture

We have implemented our proposal in the GPAC framework [5][5]. The implementation is divided into two modules: one for UPnP communications, one for the management of widgets.

The UPnP stack is using the Platinum UPnP SDK with low-level protocol functionalities allowing us to create custom UPnP devices as well as communicating with third-parties UPnP devices (media renderer and controllers).

The service discovery follows the usual UPnP process. When desired, we use the existing presentationURL in the device description to announce a widget capable of interacting with the device services.

Each widget manager has a UPnP control point monitoring the available devices and services, and performing UPnP action and event subscription. The monitoring is unaware of the type of the service and can be used with usual UPnP AV services as well as any custom service. The validation of the widget UPnP interfaces is achieved by checking the SDCP (service description) document which is available as part of the service discovery process.

In order to transfer a running widget from device A to device B, we use a regular UPnP media setup. Device A acts as a UPnP AV control point as well as a small web server, and Device B is a UPnP media renderer. Note that these services need not be associated with any widget.

5.3 Experiments

We have been able to create UPnP widgets communicating with third-party UPnP devices such as Intel Media Server and Intel Media Render.

Since the GPAC player supports the XMLHttpRequest object, we have been able to create various classic widgets such as News or Weather widgets. We have demonstrated several scenarios covering our requirements:

- battery widget communicating with a home-made UPnP service notifying about battery status. In this demonstration, the widget is announced along with the UPnP service and is automatically installed, and the service can be disabled from the widget,

- widget controlling a remote UPnP light device,

- photo album widget communicating with any UPnP AV Media Server for Image Browsing,

- media control Widget communicating with any UPnP AV Media Renderer for simple operation (play/pause/stop/info).

Each widget in our system can be migrated between devices, and we successfully tested widget migration across Linux, Windows and Windows Mobile platforms.

6. CONCLUSION AND FUTURE WORK

We have described scenarios for widget mobility motivating our work. We have reviewed existing widgets systems and highlighted their limitations for our use cases. We have proposed a mechanism enabling the communication between widgets and their environment, handling application and service mobility while retaining compatibility with existing Widget technologies. This proposal has been submitted to MPEG and is now part of the Working Draft of MPEG-U, “Rich Media UI”.

Future work includes the extension of the described mechanism to handle application context migration without any context server, allowing the seamless transfer of a widget from device to device with its execution state; and the extension of the implementation to other message exchange and service discovery protocols such as Bonjour and WSDL.

7. ACKNOWLEDGMENT

This work has been partially financed by the European Network of Excellence INTERMEDIA (IST-FP6-38419).

8. REFERENCES

[1] M. Caceres, Widgets 1.0: Packaging and Configuration, W3C Working Draft 22 December 2008, available at http://www.w3.org/TR/widgets/

[2] Cesar, P., Bulterman, D. C., and Jansen, A. J. 2008. Usages of the Secondary Screen in an Interactive Television Environment: Control, Enrich, Share, and Transfer Television Content. In *Proceedings of the 6th European Conference on Changing Television Environments* (Salzburg, Austria, July 03 - 04, 2008). M. Tscheligi, M. Obrist, and A. Lugmayr, Eds. Lecture Notes In Computer Science, vol. 5066. Springer-Verlag, Berlin, Heidelberg, 168-177.

[3] Dees, W. and Shrubsole, P. 2007. Web4CE: accessing web-based applications on consumer devices. In *Proceedings of the 16th international Conference on World Wide Web* (Banff, Alberta, Canada, May 08 - 12, 2007). WWW '07. ACM, New York, NY, 1303-1304. DOI= http://doi.acm.org/10.1145/1242572.1242820

[4] CEA, ANSI/CEA-2014 Web-based Protocol and Framework for Remote User Interfaces on UPnP Networks and the Internet (Web4CE), June 2006, http://www.ce.org/standards/StandardDetails.aspx?Id=2865&number=CEA-2014

[5] Le Feuvre, J., Concolato, C., and Moissinac, J. 2007. GPAC: open source multimedia framework. In *Proceedings of the 15th international Conference on Multimedia* (Augsburg, Germany, September 25 - 29, 2007). MULTIMEDIA '07. ACM, New York, NY, 1009-1012. DOI= http://doi.acm.org/10.1145/1291233.1291452