Declarative Interfaces for Dynamic Widgets Communications

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

ABSTRACT

Widgets are small and focused multimedia applications that can be found on desktop computers, mobile devices or even TV sets. Widgets rely on structured documents to describe their spatial, temporal and interactive behavior but also to communicate with remote data sources. However, these sources have to be known at authoring time and the communication process relies heavily on scripting. In this paper, we describe a mechanism enabling the communication between widgets and their dynamic environment (other widgets, remote data sources). The proposed declarative mechanism is compatible with existing Widgets technologies, usable with script-based widgets as well as with fully declarative widgets. 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

Communication interface, Declarative languages, Rich Media, Scripting interface, Widget.

1. INTRODUCTION

Widgets are lightweight interactive and dynamic multimedia applications. Widgets may be embedded within web pages in which case they are called Web Widgets or they may be standalone applications. In all cases, a widget is described using multimedia documents and the application displaying the widget relies on browser technologies to display it. Many widgets are already available on the Internet to provide the weather information for a particular location, to capture notes, to display news items …. Even though many widget systems exist (Apple Dashboard, Google Desktop Gadgets, Microsoft Gadgets, Opera Widgets, Yahoo! Widgets …), most widgets are made in a similar way. An XML configuration document, sometimes called manifest, provides metadata about the widget. The manifest indicates a start file that the browser will use to run and display the Widget. Typically, the start file uses a multimedia description language, such as HTML or SVG, which uses CSS resources for the styling and JavaScript code for querying the data to be displayed using the XMLHttpRequest API. These common practices are under standardization by the W3C 0.

The major problem with existing widgets is that the sources of information from which the widget retrieves its textual data (e.g. RSS feeds) or its media (e.g. images or video streams) has to be known at authoring time. However, there are several cases where a widget may want to communicate with entities not yet known at authoring time. As described in the secondary screen approach 0, there are use cases for the migration of multimedia content from one device to another. This is in particular true for widgets. In such a use case, a widget, placed in a different environment, may need to change its sources of information.

In this paper, we present a mechanism enabling the communication between widgets and their dynamic environment. We envisage in particular the communication with other widgets as well as with dynamically appearing or disappearing remote data sources. This mechanism is based on the definition of a widget service abstraction and on a description of the interface between the widget and the external services. The proposed description uses declarative tools to allow connecting any widget with external services. By declaring in the manifest all the interfaces a widget may use, we make it easy for the widget manager to validate the connections before execution and to implement security policies.

The remainder of this paper is organized as follows. Section 2 motivates this work by presenting a scenario and discussing some important requirements. Then, Section 3 compares this work with related work. Section 4 discusses document issues and describes our proposal, while Section 0 presents the implementation and results. Finally, Section 0 concludes this paper and describes future work.

2. SCENARIOS AND REQUIREMENTS

Throughout this paper, we will consider the following scenario. Imagine a home network, where an advanced printer is added. The printer can print images and can also provide a widget to configure the printing options. In this environment, a user with a mobile phone wants to print some images. The user loads a photo album widget, browses the phone's pictures, and then selects a picture of interest. When the printer is discovered by the mobile, an icon appears on its display. The user is then able to set the option and print the selected picture using the printing widget. If the printer is powered off, its associated icon disappears from the display.

In such scenario, the system should be able to:

- detect remote devices and retrieve associated widgets (e.g. the printing device/widget);
- inform widgets (e.g. the photo album) of the (un-)availability of services (e.g. printing), possibly triggering modifications in the presentation of the widget (e.g. showing/activating the print button).
- transmit messages between widgets (e.g. the image to be printed) possibly written in different presentation languages;
- transmit messages to remote services (e.g. the print command).
- transmit messages to local services (e.g. the image repository).

Of course, in the context of changing environment, the system should also be able to identify services (e.g. printing) independently of their provider name or network address to handle dynamic host naming (e.g. DHCP). In this paper, we assume that this requirement is solved by the underlying network layers.

In order to have an elegant design of the system, we add the following set of requirements. The system should:

- be independent of the presentation language used in the widget (HTML, SVG, SMIL, MPEG-4 BIFS, X3D);
- be independent of the network protocol used for service discovery and message exchanges (UPnP, Bonjour, XML-RPC);
- minimize modifications to the existing widget presentation engines.

Finally, we also add the requirement that the system should be declarative and structured, avoiding scripting for the communication process, while being compatible with scripted widgets. This last requirement is added for two reasons: to be compatible with the constraints of consumer electronics devices; and to enable easy reuse.

3. RELATED WORKS

We describe in this section similar works related to the communication between documents. We will leave out network related work such as the Web4CE Standard 0. We will also not focus on the technologies to describe widgets such as Google Gadgets, Microsoft Gadgets, Opera Widgets, Apple Dashboard or Yahoo Widgets.

From the literature, we can find several approaches to document (or widget) communication. Sire et al. in 0 propose a mechanism to enable widget to widget communication, with an emphasis on a drag and drop scenario. The proposed mechanism relies on the use of script functions such as *addWidgetEventListener* to declare that a widget wishes to start or stop receiving events. Other functions are also proposed for the specific case of drag and drop events. While this solution, and similar ones like HTML 5 Cross-document messaging or XBL, are well integrated with the current web solutions (DOM Events, Scripting), their usage of JavaScript as a core item makes them hard to use on constrained devices both for processing and security reasons. Additionally, it assumes that the associated presentation language is compatible with DOM navigation, which is not the case for some existing languages like MPEG-4 BIFS or X3D. Finally this approach only focuses on local widget to widget communications and does not consider remote widgets or data sources.

Jansen et al. 0 propose “SMIL State” which could be viewed as a related approach to our work. In this work, they use an externalized data model that can be shared among different components of a single web page. This data model could be used as a centralized storage system within which widgets could read and write data. As we will see, we have chosen a different approach in particular because we do not necessarily consider a centralized system.

A third approach to this work can be viewed in Communication with State Machines (SCXML), an XML representation of complex state machines. The framework defines, among other things, input events and output responses to the state machine, which can be seen as communication interfaces. It does however not describe how these events are exchanged with the external application or document, which is the main focus of our work.

4. CONNECTING WIDGETS WITH EXTERNAL SERVICES

In this proposal, in order to be independent from the service description (UPnP, WSDL), we consider that widgets communicate with an abstract notion of service. During such communication, messages are received by and emitted from a widget manager. Messages which can be exchanged with a specific service are grouped within an interface. Input messages are propagated by the widget manager to the widget document, possibly triggering a reply message. Output messages are created by the widget manager from the widget document. These messages may also require further processing of a reply.

4.1 Document pins

In order to fulfill our requirements, we define a declarative language enabling the widget manager to connect widgets with external services. This language is based on the usage of document input and output pins. We present here the principles of this language taking into account in particular the requirement of presentation language independence.

4.1.1 Input pins

To propagate an incoming message into documents described using these languages, there are several options.

Most languages used in today's widgets are web standards like HTML or SVG which support DOM Events. The first option is therefore to create a 'custom' DOM event out of the input message parameters and to dispatch it in the target document. For this option, one would need to specify which parameters of the input message are used and which element in the document is the target of the event. In such case, detection of the event does not require scripting and it is possible for example to start an animation upon reception of such custom event. However, full usage of this event requires scripting to retrieve the values of the parameters of the input message and apply them to the document.

The second option, still applicable in the context of DOM compatible languages but also applicable to VRML-based languages, and we believe to others, is to directly modify selected XML attributes or VRML fields. This is illustrated in Example 1.

```
<messageIn name=“MessageA” inputAction=“elt1.activate”>
 <input name=“Param1” setAttribute=“elt2.attribute1”/>
 <input name=“Param2” setAttribute=“elt3.attribute2”/>
</messageIn>
```

Example 1 – Input message with two parameters, no script

In this example, whenever an input message named `MessageA` is received, the widget document is modified. First, the value of the parameter `Param1` (resp. `Param2`) in the message is used to set the attribute `attribute1` (resp. `attribute2`) of element `elt2` (resp. `elt3`). Then, an event `activate` is triggered on the element `elt1`, as indicated by the `inputAction` attribute on the `messageIn` element. This last action is optional. It may be used when some processing has to be done after the values of the message have been propagated to the document. The `input` sub elements are optional, when no value is carried within the message. In this case, only the `inputAction` is used. As shown, this attribute may contain an event to be triggered, but it may also contain an attribute to modify. Both these solutions remain declarative and already allow for a large number of possibilities: start and stop animations; change values (e.g. text fields); trigger simple interactions (e.g. using MPEG-4 BIFS Conditional node).

For advanced handling of the messages, using a script function name in the `inputAction` attribute is possible. In this case, the values of the message parameters, as specified in the `input` elements, are passed, in document order, as arguments of the function name. This is illustrated in Example 2. We can also see in this example, that an additional attribute `scriptParamType` is specified on the `input` elements. This attribute indicates how the value of the associated parameter should be passed to the function. We define three value types: number, string or Boolean. Specifying this attribute is optional but serves two purposes: help checking that the interface matches the function signature; and reduce the parsing process in the JavaScript.

```
<messageIn name="MessageA" inputAction="processMessageA">
 <input name="Param1" scriptParamType="Boolean"/>
 <input name="Param2" scriptParamType="String"/>
</messageIn>
```

Example 2 – Input message with two parameters, scripted

In terms of implementation, this requires the ability to modify a given attribute of a given element, which is generally supported by DOM interfaces. Author should note that such approach will have similar effects as manipulating by script a DOM tree of a running timed document (e.g. SVG animations).

4.1.2 Output pins

Upon interaction within the widget, our system should be able to send messages to remote services. We present here how we use widget document constructs to create and trigger the sending of output messages.

As with input pins, a first option is to rely on DOM events. We could imagine that events of a particular type could be used to create messages when they are done bubbling in the DOM tree. Again, such a solution would be only applicable to DOM trees and would require scripting to create the events. Moreover, the application needs to be modified whenever a new type of event has to be supported.

We chose a purely declarative solution. Indeed, we noted that within DOM trees, whenever an attribute is modified, such modification can be detected using the "DOMAttrModified" event. Similarly, in VRML scene trees, whenever a field is modified, it triggers an event carrying the new value. We therefore decided to use these mechanisms to trigger the creation and sending of the message, as illustrated in Example 3.

```
<messageOut name="MessageB" outputTrigger="elt1.attr2">
 <output name="ParamX" attributeModified="elt3.attr3"/>
 <output name="ParamY" attributeModified="elt4.attr6"/>
</messageOut>
```

Example 3 – Output message with two parameters, no script

In this example, the XML description declares that whenever the value of the `attr2` attribute changes on the `elt1` element, a new message of type `MessageB` shall be created. This message shall contain two parameters `ParamX` and `ParamY` whose values are taken from the `attr3` (resp. `attr6`) attribute of the `elt3` (resp. `elt4`) element.

In terms of implementation, this just means that a new listener is attached to the elements and attributes specified in the interface.

For cases where a widget would want to send a message from script code, Example 3 can be simplified as follows. The `outputTrigger` attribute is omitted because the message is sent upon call to the `invoke` function and not upon attribute modification. Similarly, the `attributeModified` attributes are not present because parameters values are passed in the `invoke` function. The associated script would use the `invoke` function as illustrated in Example 4. We will see in Section 4.2, the usage of the message handler object `mh`.

```
<handler ev:event="click">
 var service_type = "urn:telecomparistech:service:directory:1";
 var ih = Widget.getInterfaceHandlerByType(service_type);
 var mh = ih.msgHandlerFactory("MessageB", null);
 var x = document.getElementById("r2").getAttribute("x");
 var y = document.getElementById("r2").getAttribute("x");
 ih.invoke(mh, x, y);
</handler>
```

Example 4 – Usage of 'invoke' in a widget document

4.2 Handling replies

When communicating with external remote entities, widgets may need to reply to previously received messages or to process a reply to a message sent before. In both cases, the widget manager, in charge of receiving or sending the messages for the widget, needs to be informed that an input (resp. output) message lies within the context of a previous output (resp. input) message. This context could be maintained by forcing synchronous processing of the message, waiting for the answer before exiting. However, the heavy use of AJAX programming in existing widgets or web pages illustrates how, within widgets, communication with remote entities is a fundamentally asynchronous process where data packets are exchanged over the network or between applications with no control over the response time. Widgets do not use threading models. Supporting synchronous handling of messages would imply suspending the processing of the widget until the response is received. This may freeze user interactions or communication with other services, which is not acceptable in terms of user experience.

```
<messageIn name="MessageA" inputAction="elt1.activate"
                          outputTrigger="elt9.attr12">
 <input name="Param1" setAttribute="elt2.attribute1"/>
 <input name="Param2" setAttribute="elt3.attribute2"/>
 <output name="ParamA" attributeModified="elt4.attribute6"/>
 <output name="ParamB" attributeModified="elt5.attribute7"/>
</messageIn>
```

Example 5 – Input message with a reply, no script

To maintain the context of a message in the fully declarative solution, we define `input` and `output` elements within the same `messageIn` or `messageOut` element as in Example 5.

In widgets relying on scripting, a new function, called `invokeReply` is introduced to indicate that a message should be sent as a reply to a previous message. The usage of this function is showed in . The first argument of the `invokeReply` function is a message handler object created when the input message was received.

```
var ih;
var msgHandler;
void init(param1, param2) {
 ih = widget.getInterfaceHandlerByName("searchinterface");
 msgHandler = ih.msgHandlerFactory();
 // … do something … with the given input
}
void done(result) { // called when the processing is finished
 ih.invokeReply(msgHandler, result.paramA, result.paramB);
}
```

Example 6 – Usage of `invokeReply` in a widget document

Reciprocally, when sending a message and processing its reply, a message handler object is created when the output message is sent and a call-back function is set, to be processed when the reply is received.

5. IMPLEMENTATION AND RESULTS

To validate our work, we have implemented, in the GPAC Framework 0, the support for the W3C packaging format as described in 0. In particular, the player is capable of retrieving widget packages ('.wgt' extension) and rendering them if the multimedia document language used to represent the widget is supported, namely SVG, BIFS, X3D. This basic support allows presentation of traditional widgets (weather, news, pictures) using XmlHttpRequest objects and retrieving RSS feeds from remote web servers.

We have then extended the W3C configuration document to include our proposed syntax in a different namespace and we have extended the widget manager with the associated behavior. Thanks to these extensions, we can successfully control remote UPnP services such as Light bulbs and DLNA Players with either fully declarative widgets described using BIFS or SVG or with scripted widgets. Additionally, since the GPAC player is capable of mixed rendering 0, we can demonstrate the simultaneous presentation of and communication between SVG and BIFS-based widgets.

To demonstrate our proposal, we have either used existing widgets manually modified or we created them from scratch. However, we believe it would be easy to extend widget authoring environments like Apple Dashcode to select or create the APIs to be included in the manifest and to connect their messages to scene constructs (nodes, events or functions).

6. CONCLUSION AND FUTURE WORK

In this paper, after reviewing a possible scenario for widget communication and the associated requirements, we have presented related work highlighting the need for a declarative description of widget communication interfaces. We have proposed an XML language for this description and shown how this language can be used with existing multimedia description languages such as SVG or MPEG-4 BIFS. An implementation of the proposed syntax has allowed its validation.

This work has been submitted to the Moving Picture Expert Group (MPEG) in the course of the new standardization activity related to the Rich-Media User Interface. Consequently, this work has been selected as a basis for the MPEG-U standard.

In future work, we plan to investigate how parts of widgets could be transferred between devices. This will require a finer interface description. We also envisage aggregation of widgets in complex widgets. Finally, we will consider implementing a JavaScript library for the support of these tools in existing browsers.

7. ACKNOWLEDGMENT

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

8. REFERENCES

[1] Sire, S., Paquier, M., Vagner, A., and Bogaerts, J. 2009. A messaging API for inter-widgets communication. In *Proceedings of the 18th international Conference on World Wide Web* (Madrid, Spain, April 20 - 24, 2009). WWW '09. ACM, New York, NY, 1115-1116. DOI= http://doi.acm.org/10.1145/1526709.1526884

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

[4] 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

[5] Concolato, C. and Le Feuvre, J. 2008. Playback of mixed multimedia document. In *Proceeding of the Eighth ACM Symposium on Document Engineering* (Sao Paulo, Brazil, September 16 - 19, 2008). DocEng '08. ACM, New York, NY, 219-220. DOI= http://doi.acm.org/10.1145/1410140.1410185

[6] 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

[7] Jansen, J. and Bulterman, D. C. 2008. Enabling adaptive time-based web applications with SMIL state. In *Proceeding of the Eighth ACM Symposium on Document Engineering* (Sao Paulo, Brazil, September 16 - 19, 2008). DocEng '08. ACM, New York, NY, 18-27. DOI= http://doi.acm.org/10.1145/1410140.1410146