Multimed Tools Appl
DOI 10.1007/s11042-011-0805-2

Communicating and migratable interactive multimedia documents

Cyril Concolato · Jean-Claude Dufourd · Jean Le Feuvre · Kyungmo Park · Jaeyeon Song

Abstract In ubiquitous computing environments, new interactive multimedia applications need to be mobile, device independent, potentially distributed across devices and to leverage existing services in the environment. Multimedia documents, when combined with scripting technologies, can represent complex interactive multimedia applications. However, they are still not appropriate for the creation of migratable, distributed applications in dynamic environments. We present a framework for interactive multimedia documents, which enables the communication of documents with a changing environment, the mobility of documents and the distribution of communicating document components in this environment. This framework uses an original approach, which describes communication processing outside of the document. It is based on: an abstraction model for dynamic network services; the definition of a binding description language that describes how to connect the network processing with the multimedia document processing; and on associated script programming interfaces. An implementation is presented and several examples, demonstrating in particular document mobility and document modularity, are discussed.

C. Concolato (✉) · J.-C. Dufourd · J. Le Feuvre
Telecom ParisTech, Paris, France
e-mail: concolato@telecom-paristech.fr

J.-C. Dufourd
e-mail: dufourd@telecom-paristech.fr

J. Le Feuvre
e-mail: lefeuvre@telecom-paristech.fr

K. Park · J. Song
Samsung Electronics Co. Ltd., Suwon, South Korea

K. Park
e-mail: kyungmo.park@samsung.com

J. Song
e-mail: jy_song@samsung.com

Published online: 07 May 2011

Keywords Multimedia documents · Dynamic networking · Multimedia communication · Migration · Mobility · Interactive application · Distributed applications

1 Introduction

In today's multimedia environments, users have access to a wide choice of multimedia devices (e.g. mobile phones, digital cameras, television sets, computers, tablets…) and they want to use these devices seamlessly. In particular, users want the applications running on their devices to communicate and exchange information or multimedia content with the environment, following ubiquitous or pervasive visions [9]. This requires interoperable communications between applications, across devices.

Users can access multimedia content almost everywhere through wireless networks. For example, users would expect, when entering a restaurant, that the menu of the restaurant be automatically presented onto their phone. Such a scenario requires not only service discovery, but also connections between the network and the application layer. We believe that an explicit description of the connections between service and discovery protocols and multimedia applications, as proposed in this paper, can ease the development of applications satisfying such a scenario.

Users also do not want to be tied to one device. They want to use only the most appropriate one, or possibly several devices together to interact with a complex service [3]. When watching a TV program, a user may want to use her tablet to consult associated information without disturbing the other viewers [17]. Our proposal addresses this use case by enabling the mobility of the multimedia applications, or parts thereof, across devices and by allowing modular design of mobile multimedia applications.

At the same time, the Internet is witnessing a change in the way content is created. The difference between application and content is blurring. Traditional web pages evolve into a more and more expressive and capable form of interactive content, i.e. web applications, also called WebApps. WebApps are made of the same types of content as previous web pages: they rely on XML, HTML or XHTML for structuring the content and on CSS for styling it. They reference media elements such as images, audio-video clips or text, and use JavaScript heavily to provide advanced interactive features. Full-fledged applications can now be run over the network in modern browsers. In this paper, we describe our work related to WebApps. We call them "interactive multimedia documents", or in short "documents", to focus on the structure of the content rather than on the media elements and to emphasize the fact that they can run also where there is no Web access.

As modern browsers are available on many platforms from PC to set-top boxes, mobile phones and embedded devices, interactive multimedia documents are potentially portable applications. However, current documents have limited communications capabilities. Most of them rely on the AJAX programming style and communicate with the "origin server" through the XMLHttpRequest object. We found only one very recent example [18] of a scripting library for documents communication with devices in the local/home environment. The reason for these limitations, as the authors of [9] indicate, is that there are two distinct research communities working on either interactive multimedia documents or on networking aspects for device discovery or session mobility. In this paper, we propose an approach which links the two communities, tries to overcome the limitations of document communications and enriches documents with dynamic communication capabilities.

As a consequence of these considerations, this paper presents a proposal for the use of multimedia applications in dynamic environments, designed using document technologies. This proposal is formed of three sub-proposals:

- First, the need for the interoperable communication between documents in a dynamic environment is addressed. For that purpose, a communication engine and a binding description language are introduced.
- Second, context configuration and context information documents used for the migration of interactive multimedia documents are proposed.
- And third, special re-usable documents, called components, are defined to enable the creation of distributed interactive multimedia documents.

This paper is a synthesis of previous work published in [4] and [11], but extends its scope from widgets to communicating documents, gives more details on the implementation and proposes additional aspects, such as document components.

The rest of this paper is organized as follows. Section 2 presents some scenarios and highlights the requirements behind this work. Section 3 introduces the architecture at the foundation of our proposal. Section 4 surveys existing service and discovery protocols and proposes a model abstracting the technical details of these protocols for use at the document level. Section 5 presents the concepts underlying interactive multimedia documents, extracting the assumptions we make on these documents. Section 6 exposes the XML language for the binding between documents constructs and service and discovery protocols. Section 7 addresses the specific aspects of migration of documents. Section 8 extends the proposed XML language and describes new tools for the migration of modular documents. Finally, section 9 concludes this paper and describes future work.

2 Scenarios & requirements

In order to understand the goals of our work, we present in this section some scenarios of interest, featuring the use of communicating interactive multimedia applications in dynamic environments. From these scenarios, we then derive some requirements that have guided the design of our proposal.

2.1 Scenarios

2.1.1 Scenario A

A user downloads an interactive multimedia application onto her phone to control different devices in her home environment, which includes home appliances (e.g. lights) and multimedia devices, such as her DLNA media center and TV set.

This simple scenario illustrates the fact that documents should be able to communicate with their environments. Documents need to have a dynamic access to the services and devices in their environment. Hard coding, in the document, the IP addresses of the devices to be controlled should not be required. Additionally, the document communication capabilities should not be tied to a single protocol, and not only to home networking protocols.

2.1.2 Scenario B

A user is listening to her favorite radio station with a Web Radio application on a desktop PC. While listening, she changes the settings of her preferred radio stations. Because she has to leave, she wants to migrate this application, including all settings, to her mobile phone and to continue listening to the radio on the phone.

This scenario is a classical example of Session Mobility [5] but with the difference that our focus is not on media session mobility but rather on document mobility. The task of migrating a document, which may be highly structured information, requires different tools than those used for migrating media sessions as will be shown in this paper.

2.1.3 Scenario C

During the broadcast of a TV show, an interactive voting application related to the show is broadcasted. This application has been designed so that a part of it can be migrated to secondary devices, either to show additional information about the voting or because several viewers want to vote using their phones.

In this scenario, inspired from [17], if a TV needs to move a part of the interactive application to a remote device, it needs:

- To be aware of the surrounding devices and of their capabilities,
- To exchange content with these devices,
- And to communicate with them.

Yet, we believe that a master/slave relationship between devices (TV/phone), as described in [17], is not required.

2.2 General considerations about the scenarios

All these scenarios can more or less be achieved today with native applications, i.e. not relying on interactive multimedia documents. However, realizing all of them within a unified framework is a challenge, namely because of application porting. If one considers the range of devices from a PC, to a set top box, to mobile phones based on Android, iOS, Windows or other operating systems, the cost of development of an interactive application supporting all these devices is very high.

The current trend is to create interactive applications using Web technologies such as HTML, JavaScript, etc. Such applications run in all modern browsers, which are mostly interoperable. Using such a document approach to design applications is indeed a solution to the problems of interoperability and development cost. However, several problems remain, such as adaptation of the document to the device characteristics (e.g. screen size, input capabilities) or the communication limitations. In this work, we focus only on this latter problem.

One aspect of this problem is that current web applications have limited communication capabilities because they assume a centralized architecture where one or more servers, identified at authoring time, hold the information that the application will access. In dynamic environments, possibly not even connected to the Internet, this is a severe limitation.

A second part of the problem is communication capabilities checking. We want to avoid loading a document if it cannot run properly due to the lack of some communication capabilities. A communication capabilities check should take place before loading the document rather than at runtime. This check can also be done by the user agent to filter discovered services. Additionally, it may be used for security considerations, warning a user of potential threats due to uncertified communication capabilities.

2.3 Requirements

From the above scenarios and considerations, we have based our work on the following requirements:

- It should be possible to interface interactive multimedia documents with networking protocols;
- This interface should be independent from the presentation language used in the document (HTML, SVG, SMIL, MPEG-4 BIFS or LASeR, X3D);
- It should be possible to interface scripted and non-scripted interactive multimedia documents;
- This interface should be independent from the network protocol used for service discovery and message exchanges (UPnP, Bonjour, XMLRPC);
- It should be possible to interface interactive multimedia documents with remote services in dynamic environments;
- It should be possible to move an interactive multimedia document from one device to another;
- The system should minimize the modifications to the existing presentation engines;
- The system should not require modifications to existing devices and network stacks;
- The system should enable easy-to-check and static declaration of all communication interfaces used by a document;
- It should be possible to connect an interactive multimedia document to multiple entities and to connect multiple documents to the same entity.

3 Architecture

In order to enable the proposed scenarios while satisfying the above requirements, we developed a framework for interactive multimedia documents that enables communication with the environment and the mobility of documents or parts of documents called components. This framework is based on the use of a separate layer for handling communication, migration and component management. This layer is driven by a declarative language, as opposed to solutions based on scripting APIs. The use of this declarative language structures documents through a clean separation of the communication code from the presentation/user interface, in the same manner XForms[1] separates the presentation/user interface from the service logic.

The use of an additional declarative layer implies some modifications to the architecture of traditional interactive multimedia documents user agents. Such user agents are usually composed of a network engine (NE), including an HTTP stack, and a presentation engine (PE), sometimes including a scripting engine. The PE is in charge of rendering the

[1] XForms 1.1, W3C Recommendation 20 October 2009, http://www.w3.org/TR/2009/REC-xforms-20091020/

document and handling animations and user interactions. We propose to define a new component called Communication Engine (CE). The architecture of the modified user agent we consider is presented in Fig. 1.

The CE acts as a broker between the NE and the PE. The behavior of the CE depends on the messages it receives from the network and on the interactive multimedia document that uses these messages. We argue that this behavior cannot simply be inferred from the message structure or the document structure. Hence, authors must be able to describe how the communication engine should forward messages to the PE and how messages and replies can be sent. To this end, we define a new language, the Binding Description Language, and an associated new type of document, called Binding Description, described in Section 6. In order to process a communicating document, in our architecture, the PE reads the document and the CE reads the binding description. The CE establishes bindings between matching interfaces of two documents or of one document and one external service. The CE propagates input messages to the PE and creates output messages based on events received from the PE according to those bindings. Bindings may need to be added (resp. removed) whenever a new document is added to (resp. removed from) the PE, or when the CE detects changes in the availability of services on the network.

4 Service and discovery protocols

In this section, we present first an analysis of the existing service and discovery protocols and then we present our contribution to separate the networking and interactive part of an interactive multimedia document that consists in an abstraction of these protocols and services and that can be used generically within our architecture.

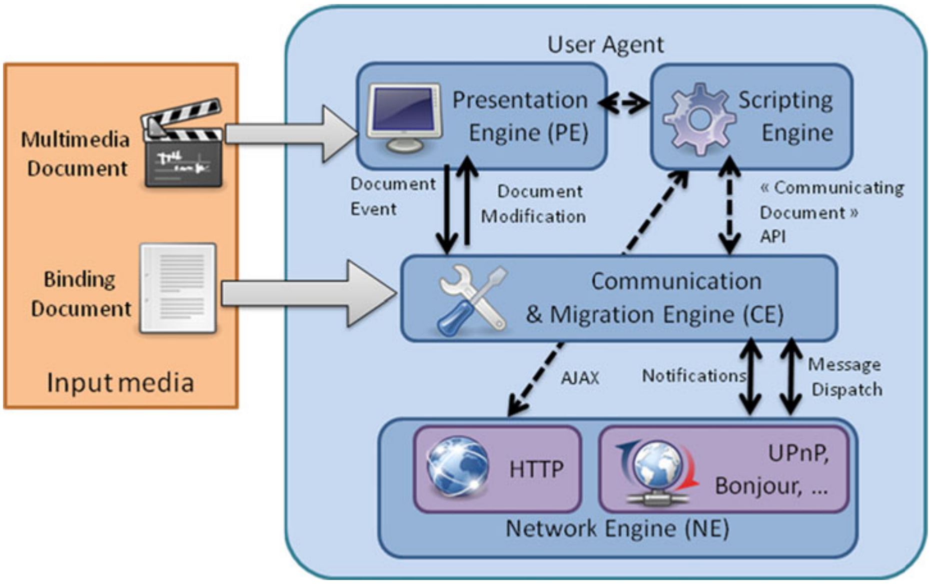

Fig. 1 User agent architecture for interactive multimedia documents communication

4.1 Existing protocols

Our everyday networking environment has been evolving in the last decade from a static environment where communicating entities were pre-configured, pre-identified devices to a dynamic one in which devices join and leave the network at any time. At the same time, devices have evolved from information consumers to information providers, and can assist other devices with various functionalities such as printing, storage, rendering or home automation control. These functionalities are referred to as services, and are largely deployed in corporate networks with Web Services infrastructure, or in the home environment with Bonjour[2] or UPnP[3] infrastructures. Most of these protocols are based on IP networking, but similar functionality is available in other networking environments, such as Bluetooth and its Service Description Protocol [1].

In a similar approach to device addressing in a network, service addressing can be classified in two categories: static and dynamic. Static approaches, such as UDDI,[4] rely on a centralized architecture in which a server acts as a repository of services available in the environment. The drawback of such systems is a frozen centralized design, requiring a client to know at least one server location. Dynamic approaches, on the other hand, rely on discovery mechanisms between clients and service providers, with no a priori knowledge of the network components.

4.2 Service abstraction model

To enable service and discovery protocols within interactive multimedia documents as proposed in our requirements, we created a service abstraction model for documents, which aims at supporting dynamic networking and at being generic and usable with existing document or service discovery technologies. This model provides an abstraction for the service, as well as an abstraction for the underlying service discovery protocols. Our model is derived from the observation that, from the document point of view, there is no difference between a service offered by the device running the document or offered by a remote device; only the presence or absence of this service matters to the document. Our model therefore considers that the addressing and DNS resolution process, whether static or dynamic, is handled by the network layer and does not need to be controlled, or even known, by the document. We thus consider that the service discovery protocol is handled at the network layer. Our only assumption is that services are uniquely identified by a string identifier (e.g. URI). The document is only aware of services appearing and disappearing, through their unique identifiers, and optionally of the location (address or host name) of the device hosting the service.

A document declares which services are needed or required, using their unique identifier, and may then be notified when each service becomes available or unavailable. As several devices may offer a service of a given type, a document can be notified multiple times of a service binding. For example, in scenario A, several lights may be controlled at a time. The document then uses the addressing information to distinguish between the various service sources.

When designing a communicating document, it is not so unusual that the document not only consumes a service, but also provides its own services to other documents. For

[2] Bonjour is Apple's implementation of ZeroConf protocols (see www.zeroconf.org)
[3] Universal Plug and Play specifications (see www.upnp.org)
[4] Universal Description Discovery and Integration (see www.uddi.org)

example, a virtual keyboard exposes a text input service to the network, or an electronic program guide exposes alert services on selected shows, or in our scenario C, the voting part of the application provides a service (the vote) to the rest of the application. This is reflected in our model by the notion of service provider. This notion allows a document designer to instruct the CE to publish the document as a service on the network using underlying service discovery tools, by giving it a specific identifier (e.g. 'urn:example:keyboard').

To interface services with a document independently of the service type, we abstracted a network service as a uni- or bi-directional communication pipe carrying messages. Each service is uniquely identified by a string identifier which may be used to define which low-level service infrastructure should be used (e.g. UPnP, Bonjour or Web service). Each message in the service is identified by a unique name within the service scope, and is a high-level view of a protocol datagram (e.g. a web service function or UPnP action or event). Each message has an associated direction, input (from service to document) or output.

Enabling bi-directionality of messages means supporting synchronous or asynchronous communication. While simpler to implement, synchronous communication may suffer long, unpredictable delays between client and service provider. In non-threaded execution environment, as is typically the case with web-based application, this can result in application freezes, which is not tolerable for the user. We therefore designed an asynchronous communication model, where the document gets notified when a reply message is received. For the author, the main drawback of asynchronous communication is the need to set up dedicated callback mechanisms, which leads to higher code complexity. However, we believe most web-based application designers are familiar with this approach, since it is quite similar to AJAX application development.

5 Document communications

In this section, our contribution is twofold. First we study the communication options of existing interactive multimedia documents formats. These options should enable the readers to understand how this proposal is applicable to other non-reviewed formats. Then we survey the communication capabilities of these existing formats to highlight the limitations that we propose to overcome.

5.1 Interacting with documents

Interactive multimedia documents may be found in several formats: (X)HTML, jointly used with CSS and JavaScript, that is the core of the web; SMIL, a language for integrating media elements and defining a spatio-temporal presentation; SVG, which inherits from SMIL its animation features and can also be used with CSS and JavaScript; Flash, the de facto standard for animations on the Internet; MPEG-4 BIFS or XMT, the binary and XML representations of an MPEG-4 document; VRML, a textual language used for describing 3D worlds; NCL, a declarative language for interactive TV applications in Brazil; etc.

Documents in all these formats are structured as trees of objects that may be tagged with an ID and that have named properties. All these formats have an event model, some with implicit event propagation up or down the tree, some with explicit event propagation along routes or links. All have explicit or implicit event listeners.

And all may include scripts. Data input into a document may thus be achieved generically by:

- writing a property of an object in the tree;
- triggering an event;
- or calling a script function defined in the document.

Data output from a document may thus be achieved generically by:

- listening for changes of a property of an object;
- listening for an event;
- a script function defined in the document being called as part of the execution of the document.

5.2 Communication capabilities and limitations

Most of the formats listed above already support some communication capabilities, mostly using ECMAScript with the XMLHttpRequest object. This object enables the document to make asynchronous requests to a server and the result may be XML code, as in the AJAX[5] model. A similar approach is to retrieve new script code, as in the JSON[6] model. Using these approaches is not satisfactory in terms of communication, mostly because the dynamic resolution of servers is not possible. Moreover, in order to communicate with a remote service using this system, web applications would have to format messages according to the service protocol, which can be complex and costly if several protocols must be handled in a single application, even if the protocol support could be provided by a specific JavaScript library, such as the one described in [18] for the UPnP protocol.

For document-to-document communication, the latest development of the future HTML 5 standard introduces a cross-window messaging system[7] that should enable such communications. While interesting, this solution relies only on scripting, and does not support dynamic addressing. In a similar way, Sire et al. [16] have proposed a scripting interface to enable widget-to-widget communication within a single page, but it also relies on scripting and is focused on communications resulting from a drag and drop operation. In this context of single page communication, one could also imagine solutions based on the use of single JavaScript context to share messages or on a central repository for information as proposed in [7], but these solutions do not enable cross device communications. To the best of our knowledge, there is no solution compatible with web technologies for device-to-service communication that supports service discovery and abstracts the protocols used, as per our requirements, as the recent formation of the W3C Web and TV interest group[8] seems to indicate.

6 Linking documents and protocols

This section presents our main contribution to linking service and discovery protocols, as exposed in Section 4, with interactive multimedia documents presented in Section 5. The

[5] Asynchronous Javascript and XML (see http://en.wikipedia.org/wiki/Ajax_%28programming%29)
[6] Javascript Object Notation (see http://www.json.org/)
[7] See http://dev.w3.org/html5/postmsg
[8] http://www.w3.org/2010/09/webTVIGcharter.html

contribution consists in a declarative language used to drive the communication engine introduced in Section 3. In this section, we also present some examples on how to use this language to enable document communication; we describe an implementation and discuss the advantages and limitations of this proposal.

6.1 The binding description

The Binding Description is the declaration of all communication interfaces that the document may use. It defines which services the document may connect to, how the document may initiate output messages, how input messages are to be connected to the document and where to read or write the values of the arguments of the messages. Its syntax is defined by the Binding Description Language (BDL), a declarative, XML-based language presented below.

6.1.1 Interface

An interface element primarily describes a service that the associated multimedia document can use. Since a multimedia document may require communication with multiple services, the Binding Description may contain multiple interface elements. The interface element has the following attributes:

– `type`: this attribute is the string uniquely identifying the service type desired by the multimedia document
– `bindAction, unbindAction`: these attributes describe what should happen when this interface is bound or unbound. Two options exist: an event to trigger, or a script function to call.
– `multipleBindings`: documents may accept connections from multiple concurrent services (e.g. when controlling several light sources with a single button). This attribute indicates if the associated document supports multiple bindings of this interface.
– `seviceProvider`: documents can be sources of information for other documents (e.g. virtual keyboard). This attribute indicates that the associated document will provide the service corresponding to this interface, and instructs the CE to publish the service on the network using the supported network stacks.

Figure 2 shows an interface from a Binding Description, which describes how to connect a UPnP AVTransport service with a document, for instance to realize our

```
<interface type="urn:schemas-upnp-org:service:AVTransport:1"
     bindAction="bindAVTransport"
     unbindAction="unbindAVTransport"
     multipleBindings=”false”>
  <messageOut name="Play">…</messageOut>
  <messageOut name="Stop">…</messageOut>
  <messageOut name="Pause">…</messageOut>
  …
</interface>
```

Fig. 2 An interface element in the Binding Description Language

scenario A. Upon detection by the CE of a service of type `urn:schemas-upnp-org:service:AVTransport`:1, the CE will ask the PE to call the script function `bindAVTransport`. Through the PE, the document can then instruct the CE to issue Play, Stop, Pause… messages to the bound service. Upon notification by the NE that the service is no longer available, the CE will ask the PE to call the script function `unbindAVTransport`.

6.1.2 Messages

In the BDL, an interface element groups all messages that the document requires the service to handle. Messages have a name, a set of arguments, some triggers and a direction, in or out.

An input message is declared with a `messageIn` element, and first has a set of `inputs`, then, if an answer is necessary, a set of `outputs`. The action that the PE should perform for an input message or for the reply to a sent message is specified using an `inputAction` attribute, which indicates the target of the action in the interactive multimedia document: an attribute to modify, an event to trigger or a script function to call.

An output message is declared with a `messageOut` element and first has a set of `outputs`, then if an answer is expected, a set of `inputs`. The trigger that the PE should monitor for an output message or for the reply to an input message is specified using an `outputTrigger` attribute: this can be an event or the modification of an attribute. Figure 3 shows how a CE is instructed on how to process an incoming message.

In this example, whenever an input message named `MessageA` is received, the PE is notified and the 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)` in the document. Then, an event `activate` is triggered on the element `elt1` in the document, as indicated by the `inputAction` attribute on the `messageIn` element. `inputAction` may contain, as shown here, an event to be triggered, but it may also contain an attribute to modify.

For advanced handling of the messages, using a script function name in the `inputAction` attribute is possible. In that case, the values of the message parameters are passed, in document order, as arguments of the function. This is illustrated in Fig. 4. Another attribute `scriptParamType` is specified on the `input` elements. This optional attribute indicates how the value of the associated parameter should be passed to the function. We define four types: number, string, binary or boolean. It helps checking that the interface matches the function signature and reduces the parsing process in the script.

In Fig. 5, the binding description declares that whenever the value of the `attr2` attribute changes on the `elt1` element in the document, a new message of type `MessageB` shall be created. This message shall contain two parameters `ParamX` and

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

Fig. 3 Input message with two parameters

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

Fig. 4 Input message with two parameters declared for usage by a script

`ParamY` whose values are taken from the `attr3 (resp. attr6)` attribute of the `elt3(resp. elt4)` element.

In terms of implementation, this means that the PE needs to attach a listener to the elements and attributes specified in the interface. As discussed in Section 5.1, this design is believed to be generic and independent from the interactive multimedia document format.

6.1.3 Handling replies

When communicating with external entities, interactive multimedia documents may need to reply to previously received messages or to process a reply to a message they sent before. In both cases, as discussed in Section 4.2, replies are asynchronous. The CE, in charge of receiving or sending the messages for the document, needs to be informed that an input (resp. output) message is related with a previous output (resp. input) message. Hence, we use `input` and `output` elements within the same `messageIn` or `messageOut` element as in Fig. 6.

6.1.4 Scripting

We have presented, so far, examples of how a purely declarative description can be used to specify the behavior of the CE. For more advanced use cases, when scripting is used in the interactive multimedia document, scripting programming interfaces are defined, as shown in Fig. 7, in order to provide similar functionality. Examples of use of these interfaces are omitted for brevity.

6.2 Applicability to the scenarios

Let us consider scenario A, in which an interactive multimedia document is downloaded to control a TV set equipped with a UPnP DLNA Media Renderer as shown in Fig. 8.

We have implemented this scenario using a document based on the SVG language for the control panel (with buttons for play, pause, etc.), an excerpt of which is shown in Fig. 9; a script file which implements the control behavior using the binding interfaces as in Fig. 10; and a binding description associated with this script and document as in Fig. 11. In this example, a click on the image will call a script function which will in turn call the

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

Fig. 5 Output message with two parameters

```
<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>
```

Fig. 6 Input message with a reply

CE to format the "stop" message of the given UPnP service and send it to the appropriate device.

6.3 Implementation

We have implemented our proposal in the GPAC player [10]. The implementation is divided into two modules written in C, C++ and JavaScript: one for the management of UPnP communications, one for the communication engine.

The UPnP module is using the Platinum UPnP SDK.[9] Using this module, the user agent is augmented with a UPnP control point that monitors the devices and services available on the network and performs UPnP action and event subscription.

The CE is in charge of validating whether discovered UPnP services can be used with the document, by first checking the UPnP service type and then comparing the BDL with the UPnP service description (SDCP) document.

We have tested our implementation across desktops/laptops, mobile phones and a TV. We have designed many communicating interactive multimedia documents: scripted/non-scripted, in MPEG-4 BIFS/XMT or SVG, with 2D or 3D graphics, including local or streamed audio-video data, controlling other documents or remotely controlled. Interested readers may view some of these examples on this YouTube video.[10]

6.4 Results, discussions and limitations

Using this implementation, we have created interactive multimedia documents communicating with third-party UPnP devices such as Intel Media Server and Intel Media Renderer. Then, we have validated that this communication approach indeed works in dynamic environments, when e.g. DLNA players appear or disappear in a LAN. Then, we have also validated the fact that a document may be connected with several services at the same time. The same validations have been made with services provided by documents marked as service providers. As a result, we believe that this system should work with many protocols for discoverable services.

Additionally, we have been able to create interactive multimedia documents mixing AJAX communications and the proposed communication method, in a complementary way:

– The document uses XMLHttpRequest when communicating with well-identified services on the Internet.

[9] See http://www.plutinosoft.com/platinum

[10] See http://www.youtube.com/watch?v=XNIBuUXLJbw

```
interface CommunicatingDocument {
   InterfaceHandler[] getInterfaceHandlers(DOMString
                                             interfaceType);
   attribute Function onInterfaceBind;
   attribute Function onInterfaceUnbind;
}
interface InterfaceHandler {
   readonly attribute DOMString boundHost;
   readonly attribute DOMString type;
   MsgHandler msgHandlerFactory(DOMString name,
                                Function callBackFunction);
   void invoke(MsgHandler msgHandler, …);
   void invokeReply(MsgHandler msgHandler, …);
   MsgHandler msgHandlerFactory();
}
interface MsgHandler {
   readonly attribute DOMString msgName;
   readonly attribute InterfaceHandler interfaceHandler;
}
```

Fig. 7 Overview of the Binding Programming Interfaces

– And the same document uses our framework whenever the service location or presence cannot be predicted at authoring time.

In terms of authoring, although manually written, the overhead of creating the binding description was found negligible compared to the usual development of AJAX-enabled documents. Finally, we found that one advantage of our system is to be able to work with devices, which cannot afford embedding a scripting engine.

7 Interactive multimedia document migration

In the previous section, we have presented our solution for the communication between documents. However, with an increasing number of interactive display devices in her environment, a user may want to move a running communicating application (or document) between devices for a better experience, for example switching from a mobile phone to an interactive TV. This concept is often called "Session Mobility" [5] [12] or "Service Mobility". In this paper, to avoid confusion with device mobility, we call "migration" this process of moving the application between devices.

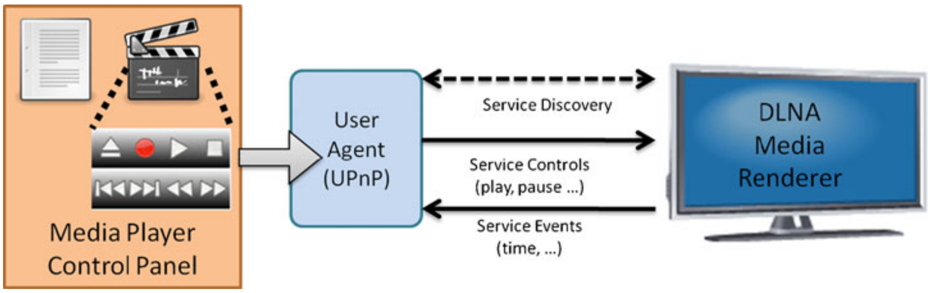

Fig. 8 Communication between an interactive multimedia document (Media Player Control Panel) and a remote service (DLNA Media Renderer)

```
…<g>
  <image xlink:href="media-playback-stop.png" x="205" y="5"
         width="50" height="50">
    <handler ev:event="click">stop();</handler>
  </image>
</g>…
```

Fig. 9 Excerpt of interactive multimedia document

As explained in [5], when migrating a running application, it is important to be able to restart the application in the same state. This implies the notion of application context, e. g. the set of parameters an application will save when being stopped and restore when being restarted. This context can be saved on the device persistent storage and restored for regular application restart not involving migration, or sent over the network during application migration.

When migrating a communicating document, it is also important to reestablish communication after the migration. Depending on the situation, devices previously connected will rediscover the migrated document or the document will rediscover previously connected services, and similar bindings will be restored according to the Binding Description described in Section 6.1. In this respect, our contribution exposed in this section is the definition of configuration and context information enabling the migration of communicating documents.

7.1 Related work

Migration has been covered by several related research works in the last decade. Some approaches have focused on socket APIs for application migration [8] or on protocols for migration of streams such as voice or video (Mobile IP [13], RTP or SIP [15][14]). Other approaches looked at the migration of applications between virtual machines like Java. Some other work used a distributed approach for partial mobility. A good example of this latter approach can be found in the AURA platform [6], where each user task is abstracted as a service. When migrating, the originating device can suspend any task and instruct the target device to resume the task. Our approach is slightly different since we do not attempt to provide a model for user tasks but rather give context restoration tools to the application designer. Moreover, we focus on being able to use the same application during the migration and not on finding applications that could perform the same task on the target device.

Our approach towards preference storing is very similar to HTML cookies or the more recent WebStorage[11] specification. However, these tools do not take into consideration application migration, and would require an external server to store and retrieve the data (e. g. using XMLHttpRequest), which is not always possible (local network with no outside link) or desirable (user privacy).

7.2 Context configuration and information

In our framework, a context configuration, describing the different parameters used to save and restore the state of the application, is given in a simple XML language (similar to the context digital item in [5]), and the user agent manages the saving and restoring of these

[11] See http://dev.w3.org/html5/webstorage/

```
function stop() {
  var instanceID = 0;
  var iha = getInterfaceHandlersByType(
                    "urn:schemas-upnp-org:service:AVTransport:1");
  if (iha == null) return;
  iha[0].invoke(iha[0].msgHandlerFactory("Stop"), instanceID);
}
```

Fig. 10 Example of use of the binding interfaces

parameters. We use the saved state of the interactive multimedia document for later restoration either on the same device (persistent storage) or on a remote device (migration). These two cases do not necessarily require saving the same set of parameters. Therefore, each parameter is assigned, by the application author, a migration type indicating in which condition the parameter should be saved and restored: persistent storage only, migration only or both of them, as shown in Fig. 12.

When the interactive multimedia document is migrated, the user agent serializes the context according to the context configuration into a special description called Context Information. This process is depicted in Fig. 13.

This context information is then used to restore the parameters. For that purpose, it may be transmitted in multiple ways: embedded in the binding description; stored in a file and packaged and delivered with the binding description and interactive multimedia document; or retrieved separately. Figure 14 shows a Context Information corresponding to Fig. 12. One should note that all parameters not taking part in migration, i.e. with their migratable attribute set to 'saveOnly', have been removed.

Another problem when performing migration is that the application (e.g., the interactive multimedia document and its resources) may already be present on the target device. In most cases, the target device cannot rely on the notified URL to check if the application is already present, and will have to download the application package to perform this check. To avoid this situation, each application can be assigned a unique identifier (UUID) and this unique identifier is inserted in the context information document, along with some versioning information. This allows the target device to check whether the corresponding application is already installed only by retrieving the context information document.

7.3 Applicability to the scenarios

Consider our scenario B. A simple web radio player is described by an interactive multimedia document, where the user can choose her favorite station. The document acts as a service provider and exposes a tuner interface, which allows a remote application to select the current station. The user first loads the interactive multimedia document on her mobile, controls volume and other audio effects, and later migrates it on the multimedia system of her living room. The presentation then resumes the previously playing radio with the same sound settings. If another remote device was originally bound to the tuner interface as discussed in previous section, it is automatically bound again when resuming the interactive multimedia document presentation, assuming that the living room multimedia systems supports the same service discovery protocol as the controlling device. This scenario is demonstrated in this YouTube video.[12]

[12] http://www.youtube.com/watch?v=rbDddMMGg8U

```
<interface type="urn:schemas-upnp-org:service:AVTransport:1">
    …
  <messageOut name="Stop">
    <output name="InstanceID" scriptParamType="string"/>
  </messageOut>
    …
</interface>
```

Fig. 11 Example of binding description

7.4 Implementation

In our implementation, we have experimented migration of documents based on UPnP. We initiate the migration using the SetAVTransportURI function of the standard UPnP MediaRenderer service, which instructs a remote media renderer to play the given resource. We use the associated mime type of the content to filter regular audio/video from interactive multimedia document. Context information is serialized by the originating device and shared through an embedded HTTP server, which also acts as an UPnP Media Server.

7.5 Results, discussions and limitations

We have tested our migration framework across several operating systems (Windows, Mac OS X, Linux, Windows Mobile and iOS) in both LAN and WLAN environments. We observed a large variation in UPnP protocol stack reactivity: on LAN environments, a typical migration, including Context Information retrieval, between desktop computers would complete in about one second; the same migration over WLAN between a desktop computer and a mobile phone, would take much longer to complete, typically a few seconds. This can be explained by the use of a mobile device, whose low processing power introduces latency in the UPnP HTTP requests handling. However, migrating between desktop computers over WLAN is also significantly longer than over LAN (typically a couple of seconds). We have not yet investigated how redesigning the UPnP stack could reduce this latency.

This latency is one limitation of our system. The Context Configuration data has to be stored on the originating device until fetched by the target device, which introduces HTTP download with its own latency. This will have to be addressed in future work, using techniques such as multi-part MIME HTTP requests in UPnP. Moreover, the context configuration reflects the state of the application at the time the migration is initiated, which could be different from the state of the application when it is resumed as explained in [5], for example when animations are running and the application is migrated to several devices at the same time, such as in our scenario C. This limitation should be addressed in future works, using techniques such as Network Time Protocol tagging of the context information.

Finally, we have not yet studied how our migration framework can be used together with multimedia stream migration using SIP [15][14] or other techniques. This will have to be addressed in future work on use cases where the application contains animation synchronized with streamed content (annotations, subtitles).

```
<contextConfiguration>
 <preference name="color" value=”red” migratable=”saveOnly”/>
 <preference name="currentPage" value=”0” migratable=”saveAndMigrate”/>
 <preference name="lastRequest" value=”foo” migratable=”migrateOnly”/>
</contextConfiguration>
```

Fig. 12 Context Configuration example

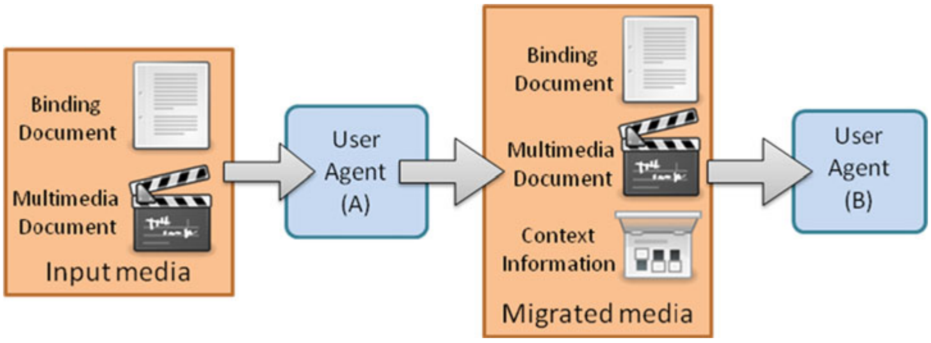

Fig. 13 Document Migration and Context Information

8 Document modularity

In Section 6, we exposed our proposal to enable communicating documents. In Section 7, we presented how communicating documents can be migrated. This section shows how we can leverage communication and migration to design modular migratable and communicating documents. Our contribution is an extension to the binding description language presented in 6.1 that enables the modularity of documents and enhances the communicating and migrating capabilities of the initial document.

8.1 Related work

Document modularity is the capability for designers to create interactive multimedia documents in pieces or modules, each piece being a simpler, more focused document, and to link these pieces together to assemble the complete application. Modularity is important in multiple ways: to reduce complexity during authoring, porting and maintenance; to allow the creation of libraries of components and thus foster reuse.

When designing an application (i.e. a document) using reusable modules (i.e. sub-documents), the most important problem is designing a data exchange system between the application and the modules. Existing approaches usually define exported symbols (X3D[13]) or interfaces (X3D Proto, SVG Parameters[14]) for each module document. These designs imply that the host document knows the location of the modules and loads them at runtime. This is not fitting well in a distributed environment, where the modules could be on remote devices.

8.2 The component approach

In order to solve the problem of modularity in communicating and migratable documents, we reuse the service abstraction model defined in Section 4, and define the notion of component: a document, usable by another document regardless of its network location. The notion of component document is very loose, in the sense that the defining feature is the ability of a document to start/stop another document. Whether the two actually communicate is not really important to the notion of component document, but it can be an added value.

Depending on the designer choice, specific components can be enforced or any component fitting a set of requested functionalities may be used (e.g., any virtual keyboard

[13] Extensible 3D Format (see http://www.web3d.org/)
[14] See http://www.w3.org/TR/SVGParam/

Fig. 14 Context information description

```
<contextInformation uuid="someid" version="1">
 <preference name="currentPage" value="1"/>
 <preference name="lastRequest" value="bar" />
</contextInformation>
```

with the appropriate interface). We furthermore take into account the fact that some components are required for the document, which cannot be used without them, while others are optional.

A component element, in the BDL (see Section 6.1), signals that the document associated with the binding description requires other component documents. The main attributes of the component element are:

- `activateTrigger and deactivateTrigger`: these two attributes declare how the requesting document will trigger the activation or deactivation of the component document. Their value can be a document event or an attribute whose modification will trigger the action.
- `activatedAction, deactivatedAction and activationFailure`: these attributes declare how the document will be informed of the successful activation, deactivation or failure of the component. The value may be a script function to call, an event to fire or an attribute to modify.
- `id`: this attribute is used to identify a component when the document uses scripting interfaces for component management.

The component element syntax offers two ways to link a document with its components: by using a URL attribute or by listing the type of the interfaces that the component shall match, indicated in `requiredInterface` elements.

In both cases, the specificity of using the proposed component approach is that upon activation, not only the component will be displayed with the main interactive multimedia document, as processed by the PE, but if the interfaces indeed match, the component will be bound to the document by the CE and will provide a service to the main document. This is illustrated in Figure 15.

Figure 16 shows the binding description of a document using a component with two interfaces. It should be noted that when scripting is used in the document, it could be interesting to load, activate, deactivate and unload components using scripts. For that purpose, we have defined a component programming interface that is omitted for brevity.

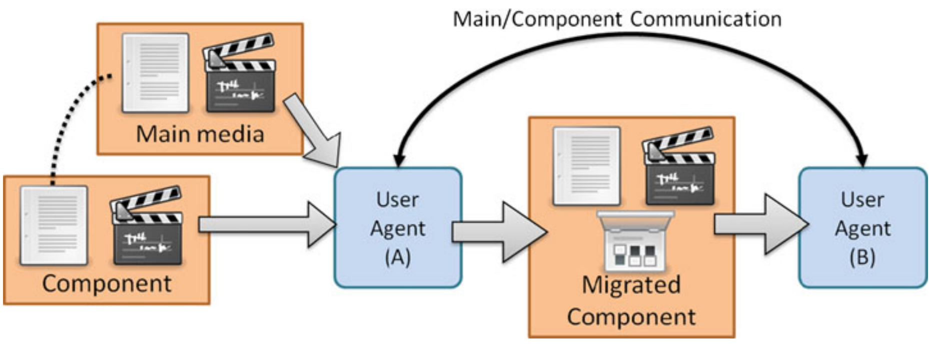

Fig. 15 Migration of a component and communication with the main document

```
<interfaces>
  <interface type="urn:telecomparistech:serv:confprint:1">
    <messageIn name="text" inputAction="addText">
      <input name="text" scriptParamType="string"/>
    </messageIn>
  </interface>
  <interface type="urn:telecomparistech:serv:confpref:1">
    <messageOut name="pref" outputTrigger="t5.textContent">
      <output name="preference" attributeModified="t5.textContent"/>
    </messageOut>
  </interface>
</interfaces>
<component>
  <requiredInterface type="urn:telecomparistech:serv:confpref:1"/>
  <requiredInterface type="urn:telecomparistech:serv:confprint:1"/>
</component>
```

Fig. 16 Declaration of a component with two interfaces

8.3 Applicability to the scenarios

Consider our scenario C. A TV channel broadcasts a document branded with the current show. The document is run on the TV set, and voting for the current TV show can be done with the TV remote when there is a single active viewer in front of the TV set. But as soon as there are many people in front of the TV set, each of them having a mobile phone capable of document rendering, and each wanting to vote, some way of voting needs to be given to the additional viewers. Imagine that the document is broadcasted with additional resources: an on-TV, "local" voting component, a touch screen voting component for smartphones and a small-keyboard voting component for older phones. Only the on-TV component is launched by default. Upon request from the user, the TV application could send appropriate components to each discoverable mobile phone. In this case, the main document would expose a voting interface, acting as a service provider, and would be bound to all voting components using the `multipleBindings` attribute.

8.4 Results, discussions and limitations

We have experimented with the component approach in several examples. These experiments showed us that components indeed help in solving interesting scenarios such as our scenario C. This approach however has some limitations that we would like to address in the future. For instance, when many documents and components coexist in the environment, we would like to guarantee that the binding between these documents and components is indeed the one that solves the user's problem. In a first approach, this could be solved by asking the user but we would like to have a deterministic binding algorithm that helps authors design applications. A related problem concerns the synchronization of the bindings. In our scenario C, we would like to insure that the binding only happens when the broadcast show requires voting.

9 Conclusion & future work

In this paper, we have proposed a framework to link interactive multimedia documents with service and discovery protocols, factoring the network processing out of the document. This framework is compatible with many document formats and service and discovery protocols.

We have proposed a service abstraction model based on careful analysis of existing protocols, and defined the Binding Description Language. This language allows application designers to create documents that communicate with the various services that may appear and disappear in the network during the lifetime of the application. This language has been designed for low-complexity, script-less documents as well as full-fledged scripted multimedia applications. We have also proposed tools for interactive multimedia document migration between devices allowing for application context restoration. To help developers build powerful, distributed multimedia applications, we introduce a modular architecture for interactive multimedia documents with reusable components communicating with the main application.

We have implemented our proposal in a multimedia player and demonstrated several applications, designed with W3C SVG or MPEG-4 BIFS, communicating with existing DLNA devices and with other applications through generic UPnP services. We have demonstrated document migration between various devices (TV sets, mobile phones, PCs) and distributed applications through our component system.

This work has been standardized by the Moving Picture Expert Group as part of ISO/IEC 23007–1 (MPEG-U). Our implementation of the standard has been publically released as part of the GPAC project and acts as the reference software. Our framework is also fully compatible with web-based applications defined in W3C Widgets Packaging and Configuration [2].

In the future, we plan to explore the integration of our tools with other service discovery protocols (WS-Discovery, Bonjour) and existing web browsers for XHTML/HTML5 support. In particular, we are interested in implementing a communication engine as an extension of existing web browsers using Javascript. Additionally, we would like to provide answers to the limitations identified in the paper to improve the latency in migration, to enable collaboration with media-specific session mobility schemes, or to better handle complex component based applications.

References

1. Avancha S, Joshi A, Finin T (2002) Enhanced Service Discovery in Bluetooth. Comput 35:96–99. doi:10.1109/MC.2002.1009177
2. Caceres M (2008) Widgets 1.0: Packaging and configuration. W3C Working Draft 22 December 2008, available at http://www.w3.org/TR/widgets/
3. Cesar P, Bulterman DC, Jansen AJ (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 3–4, 2008. doi:10.1007/978-3-540-69478-6_22
4. Concolato C, Le Feuvre J, Dufourd JC (2009) Declarative Interfaces for dynamic widgets communications. In Proceedings of the 9th ACM Symposium on Document Engineering, Munich, Germany, Sept 15–18, 2009. doi:10.1145/1600193.1600245
5. De Keukelaere F, De Sutter R, Van de Walle R (2005) MPEG-21 Session Mobility on mobile devices. In Proc 2005 Int'l Conf on Internet Computing, pp 287–293, Las Vegas, USA
6. De Sousa J, Garlan D (2002) Aura: An architectural framework for user mobility in ubiquitous computing environments. In Proc IEEE-IFIP Conf Software Architecture
7. Jansen J, Bulterman DC (2009) SMIL State: an architecture and implementation for adaptive time-based web applications. Multimedia Tools Appl 43(3):203–224. doi:10.1007/s11042-009-0270-3
8. Kaneko K, Morikawa H, Aoyama T (2003) Session Layer Mobility Support for 3C Everywhere Environments. In Proc 6th Int'l Symp Wireless Personal Multimedia Comm. IEEE Press, pp 347–351
9. Kernchen R, Meissner S, Moessner K, Cesar P, Vaishnavi I, Boussard M, Hesselman C (2010) Intelligent multimedia presentation in ubiquitous multidevice scenarios. IEEE Multimed, April 2010, pp 52–63. doi:10.1109/MMUL.2009.75
10. Le Feuvre J, Concolato C, Moissinac JC (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:10.1145/1291233.1291452

11. Le Feuvre J, Concolato C, Dufourd JC (2009) Widget mobility. In Proceedings of International Conference on Mobile Technology, Applications and Systems, Mobility, Nice, France, September 2–4, 2009. doi:10.1145/1710035.1710060
12. Mate S, Chandra U, Curcio IDD (2006) Movable-multimedia: session mobility in ubiquitous computing ecosystem. In Proc 5th Int'l Conf Mobile and Ubiquitous Multimedia. ACM Press, doi:10.1145/1186655.1186663
13. Perkins CE, Myles A (1997) Mobile IP. In Proceedings of International Telecommunications Symposium, pp 415–419
14. Schacham R et al (2007) Ubiquitous device personalization and use: the next generation of IP multimedia communications. ACM Trans Multimed Comput Comm Appl 3(2), doi:10.1145/1230812.1230818.
15. Schulzrinne H, Wedlund E (2000) Application-layer mobility using SIP. Mobile Comput Commun Rev 4 (3), July 2000. doi:10.1145/372346.372369.
16. Sire S, Paquier M, Vagner A, 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, pp 1115–1116. doi:10.1145/1526709.1526884
17. Soarez LFG, Costa RMR, Moreno MF, Moreno MF (2009) Multiple exhibition devices in DTV systems. In Proceedings of the 7th ACM International Conference on Multimedia, Beijing, China, October 19–23, 2009. doi:10.1145/1631272.1631312
18. Web Device Connectivity, Ericsson Labs, https://labs.ericsson.com/apis/web-device-connectivity/

Cyril Concolato is Associate Professor in the Multimedia Group at Telecom ParisTech, Paris, France, where he received his master and doctoral degree in Computer Science in 2000 and 2007, respectively. His interests lie in multimedia scene descriptions and in interactive multimedia applications, in particular for the mobile domain. He is an active participant to the standardization bodies of MPEG and W3C. Finally, he is one of the project leaders of the Open Source project GPAC.

Jean-Claude Dufourd is currently a Professor at Telecom ParisTech. Before, he was co-founder and Chief Scientist of Streamezzo, a mobile multimedia software vendor from 2004 to 2008. He has been an active contributor to MPEG since 1995, editor of MPEG-4 LASeR, chairman of the Integration group from 2002 to

2006. He is a former student of the Ecole Normale Supérieure de Paris, a graduate from ENST and holds a PhD and Habilitation in Computer Science of Paris VI.

Jean Le Feuvre received his Ingénieur (M.Sc.) degree in Telecommunications in 1999, from TELECOM Bretagne. He has been involved in MPEG standardization since 2000, and joined TELECOM ParisTech in 2005 as Research Engineer within the Signal Processing and Image Department. His main research topics cover multimedia authoring, delivery and rendering systems in broadcast, broadband and home networking environments. He is the project leader and maintainer of GPAC, a rich media framework based on standard technologies (MPEG, W3C, and Web3D).

Kyungmo Park Kyungmo Park received BS and MS degrees in Electronics from Kyunghee University in 2003 and 2005, respectively. He has been with Samsung Electronics Co., Ltd. since his graduation in 2005 and has been involved in many multimedia standard developing organizations (SDOs) including MPEG, CEA and DLNA. Since 2007, he has been involved intensively in MPEG standardization, and has contributed more than 30 contributions in areas of MPEG-U widgets, Digital Item Adaptation, Dynamic Adaptation Streaming over HTTP and stereoscopic file format. His research interest areas include multimedia processing and communications, as well as interactive multimedia services.

Jaeyeon Song received her B.S., M.S. and Ph.D. degrees in Electronic Engineering from Hong-ik University, Korea in 1995, 1997 and 2001 respectively. She joined Samsung Electronics, Co., Ltd as a senior engineer in 2001. She has participated in many projects for mobile broadcast service standardization. Her research interests include Smart TV and interactive rich media service. She is a director for multimedia standarization in Samsung.