Dynamic Graphical User Interface Generation for Web-based Public Display Applications

Jorge C. S. Cardoso
CITAR/School of Arts, Portuguese Catholic University, Rua Diogo Botelho, 1327, 4169-005 Porto, Portugal

Keywords: Dynamic Graphical User Interface, Public Displays, Mobile Applications, Toolkits.

Abstract: Public digital displays are moving towards open display networks, resulting in a shift in the focus from single-purpose public displays that are developed with a single task or application in mind, to general-purpose displays that can run several applications, developed by different vendors. In this new paradigm, it is important to facilitate the development of interactive public display applications and provide programmers with toolkits for incorporating interaction features. An important function of such toolkits is to support interaction with public displays through a users' smartphone, allowing users to discover and interact with the public display applications configured in a given display. This paper describes our approach to providing dynamically generated graphical user interfaces for public display applications that is part of the PuReWidgets toolkit.

1 INTRODUCTION

Interactive public display systems have gained considerable attention form the research community in the last years due to their increasing ubiquity. In particular, research has started to focus on pervasive open display networks (Davies et al., 2012) as a means to attain the full potential of the vast number of digital public displays that exist today. In an open network, display owners can easily interconnect their displays and take advantage of various kinds of existing content, including interactive applications. Application developers can create applications and distribute them globally, to be used in any display. Users can not only watch the content played on the display, but also appropriate it in various ways such as interacting with it, expressing their preferences, submitting and downloading content from the display.

In order for applications to work across a variety of different public display systems, developers need to rely on protocols, standards, and interaction abstractions so that they do not have to deal with the specificities of each display technology, in particular interaction mechanisms and modalities that may be very diverse. Short Message Service (SMS), Bluetooth, email, touch, Quick Response codes (QR codes), mobile devices, are a small set of interaction mechanisms that have been used in the public display context. In recent years, smartphones have gained enormous traction and are expected to continue to grow in usage during the next few years (Finance, 2012), making them also a privileged choice for interacting with public displays. It is current practice for many web applications to simultaneously offer desktop and mobile versions of their applications. This approach will certainly be used also for some of the public display applications of the future, which will want to deliver a customised interaction experience to their users. However, public display systems can also provide generic mobile applications that help users discover, explore, and control the public display (José et al., 2013).

In order to attract a large number of developers, incorporating high-level interaction features into applications should be an easy task. These high-level features should work seamlessly with a number of heterogeneous interaction mechanisms, including smartphones with rich user interfaces. Following this line, we have built a toolkit – PuReWidgets – for web-based interactive public display applications that allows developers to incorporate, in their applications, high-level interaction controls that respond transparently to various interaction mechanisms such as SMS, email, QR code, and touch. Additionally, our toolkit is able to generate dynamic user interfaces for mobile devices that allow users to discover and interact with the applications running in a specific place, and their interaction features.

In this paper, we focus on the dynamic generation of graphical user interfaces. We describe our approach; present its architecture and programming model; and present a preliminary evaluation of the resulting system that consisted in a real-world deployment of a public display running various applications built with the toolkit.

2 RELATED WORK

Automatically generating a user interface is a common approach in ubiquitous computing environments and smart spaces where there are services associated with places, which can be accessed through many different devices. In this approach, the application logic and user interface are usually distributed: the user interface runs in a controller device that connects remotely to the application logic through some kind of remote method invocation mechanism. Service developers focus on the implementation of the internal logic and on what functions the service should provide to the outside world. These functions represent the programmatic interface that the service exposes to other system components.

There are several ways to make the user interface appear in the controller device and allow a user to control the service. We describe some alternatives in the following subsections.

2.1 Downloadable UI Code

One approach for creating the user interface is to have the controller download the User Interface (UI) code from the service itself and then execute it locally. This is the approach followed by the Jini Service UI (Venners, 2005), which is based on Jini's service architecture. In Jini, services announce themselves by registering in a lookup service available in the network. This registration involves placing a service proxy object for the service in the lookup service. This service proxy is a runnable Java object that clients use to interact with the service. Clients use the lookup service to find a service of interest and download the associated service proxy. To invoke a function on the service, clients call a local method on the proxy. The proxy object takes care of communicating over the network to the corresponding service, using whatever protocol the service developer chose.

2.2 Abstract UI Description

A different approach for showing the user interface in the controller device is to have a service interface description that can be read by a controller to dynamically generate a suitable user interface (graphical, speech, gesture, command line, etc.). For example, (Roman, Beck, & Gefflaut, 2000) developed a device independent, XML-based, language for representation of services and the respective dynamic interface generation for mobile devices. The service description language allows the definition of a list of methods with parameters, which can be invoked by the client. The service description also includes a description of GUI elements that should be used in the user interface to represent the parameters and actions that trigger the invocation of methods. Clients obtain the service description and employ an eXtensible Stylesheet Language (XSL) transformation particular to the device to generate the HTML user interface, which is then presented in a native browser.

In XWeb (Olsen et al., 2000), a service exposes its data through a Data URL that can be manipulated by clients but, in order to provide a user interface, services also specify a View URL that points to an XView. The XView specifies the interaction that can be accomplished with the data, defining the possible data types and transformations from internal representations to user-friendly representations called interactors (numbers, dates, times, enumerations, text, links, groups, and lists). An XWeb client uses the XView to generate a user interface appropriate to the interaction modality of the device. Because XViews are independent of any concrete interaction modality, devices are able to generate user interfaces for graphical and speech-based interaction.

Another system that resorts to dynamic user interface generation is the Personal Universal Controller (PUC) (Nichols et al., 2002). PUC uses a peer-to-peer communication architecture between client devices and services, and a service specification language, based on XML, which models data (state variables) and actions (commands). A service is modelled as a group tree of elements (variables and commands); this structure allows service developers to group similar elements together providing cues for the interface generators. The specification language also allows the definition of dependencies between elements. The graphical user interface generator uses the combination of the group tree and dependencies to organise the interface components into logical groupings.

2.3 Hybrid Approaches

There are also hybrid solutions for generating the user interface: custom-designed user interfaces can be associated with services for some types of devices but if a suitable user interface for a particular device does not exist, the device can dynamically generate a user interface from the service description. (Hodes & Katz, 1999) for example, proposed an XML-based Interface Specification Language (ISL) that allows the specification of methods that can be invoked on the service and also of user interfaces for that service, available to be downloaded and executed in the device. ISL includes a <ui> tag to specify a user interface for the service, which can be the name of a component that the device somehow knows about or a network address where the component can be found. If no user interface is specified for a service, or if no suitable one for that particular device is found, the device can use the ISL to dynamically create a user interface that allows users to interact with the exposed methods of the service.

In iCrafter (Ponnekanti et al., 2001), services register in an Interface Manager and send it a service description in an XML-based language called Service Description Language (SDL) – which lists the operations supported by the service, in a similar way to ISL. Clients obtain a list of available services from the Interface Manager and can ask for the user interface of a specific service (or a combination of services). When asked for a user interface, the Interface Manager will search for a suitable interface generator: it first searches for a generator for that specific service, then for a generator for that service interface, and finally for the service-independent generator. This allows the creation of custom user interfaces for a service, if the developer chooses to, but guarantees that a suitable user interface can always be presented to the user. The interface generator uses a template to generate code in a user interface language supported by the controller device (iCrafter supports HTML, VoiceXML, SUIML and MoDAL), so controller devices are assumed to be capable of running a user interface interpreter that can then render the received user interface code.

The toolkit presented in this paper draws inspiration in the dynamic user interface generation approach and provides web-based user interface generation for public display applications. Unlike the approaches presented in this section, our approach does not require programmers to use an interface description language to explicitly define the user interface of the application. Instead, our toolkit continually gathers information about which widgets the application has created in order to be able to replicate them in the dynamically generated interface.

3 OVERVIEW OF PUREWIDGETS

PuReWidgets is an interaction toolkit for web-based public display applications. In this section we briefly outline its main features and architecture, but a more in-depth description can be found in (Cardoso & José, 2012). PuReWidgets provides the following main features:

Multiple, extensible, widgets. The toolkit is structured around the concept of widget. It incorporates various types of interaction widgets that support the fundamental interactions with public displays. Existing widgets can be customised and composed into new widgets, and completely new widgets can be created by application programmers.

Dynamically generated graphical interfaces. The toolkit automatically generates graphical user interfaces for mobile devices (web GUI). It also generates QR codes for user interaction through camera equipped mobile devices.

Independence from specific input mechanisms. and modalities. The toolkit supports several interaction mechanisms such as SMS, Bluetooth naming, OBject EXchange (OBEX), email, touch-displays, in addition to the already mentioned desktop, mobile, and QR code interfaces. These input mechanisms are abstracted into high-level interaction events through the available widgets, so that programmers do not have to deal with the specificities of the various concrete mechanisms.

Asynchronous interaction. The toolkit supports asynchronous interaction, allowing applications to receive input events that were generated when the application was not executing on the public display. Generally, this allows users to send input to any application configured in a display, including the ones not currently executing at the display. More specifically, this can be used, for example, to allow off-line customisation of an application so that relevant content is shown to a particular user or group of users when the application is displayed.

Concurrent, multi-user interaction. The toolkit supports concurrent interactions from multiple users, and provides applications with user identification information that allows them to differentiate user input.

Graphical affordances. The toolkit provides default graphical representations for its widgets. Widgets also provide graphical input feedback on the public display when an input event occurs.

3.1 Architecture

PuReWidgets was designed to support displays in various independent administrative places, running various applications developed by third-party developers. Figure 1 depicts the main physical components of a network of public displays.

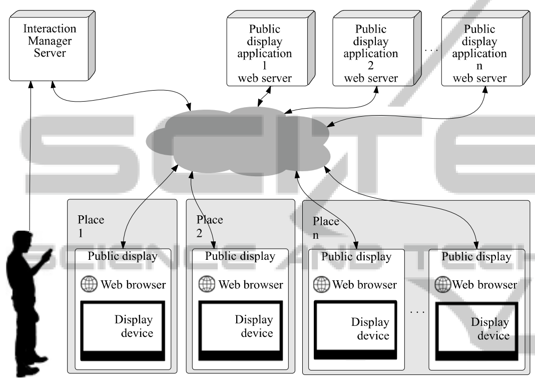
Interaction Manager Server
Public display application 1 web server
Public display application 2 web server
Public display application n web server
Place 1
Place 2
Place n
Public display
Web browser
Display device

Figure 1: Physical components of the PuReWidgets architecture.

From the perspective of a public display, a PuReWidgets-based public display application is a standard web application that is downloaded from a third-party web server and runs in a standard web browser component in the public display. Interaction with a public display application is accomplished through an Interaction Manager (IM) server that is part of the PuReWidgets toolkit. A single IM supports various independent applications and displays.

The toolkit is composed of a widget library that programmers include in their application's code, and a web service that handles interaction events (see Figure 2). When the application is on-screen, the library receives input events from the IM and passes them on to the widgets being used by the application.

The development process of a public display application that uses PuReWidgets is similar to the development of a regular web application: developers include the library in their projects and use the available functions of the library to code the application, instantiating widgets and registering interaction event callback functions. Developers then deploy the set of HTML, CSS, and Javascript files that compose their application to a standard web server.

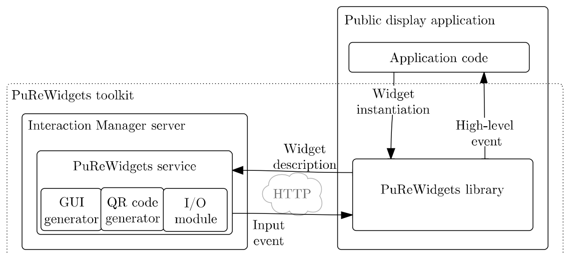

Figure 2: Logical components of the toolkit.

3.2 Interaction Manager

The IM server mediates all user interaction with the public display applications. The IM keeps a database of every widget created and in use by applications and is capable of routing the various interactions to the correct application. The PuReWidgets library communicates with the IM via an HTTP/REST protocol service, for submitting and receiving widget information and input events. The IM is structured around the following set of concepts:

Place. A place is an administrative area defined by the display owner. A place can have different levels of granularity: it can refer to something small like a specific cafeteria, with a single public display, or to a wider place like a university campus, with various public displays. A single IM server can handle multiple independent places, each identified by a unique place id.

Application. An application is a web application, identified by its URL, which uses the PuReWidgets library. Display owners may associate several applications with a single place. Each association is an application instance in the IM, identified by an instance id. When an application instance is running on the public display and showing its content, it is said to be on-screen, otherwise it is off-screen. Off-screen applications can still receive input, but are not able to react immediately on the public display.

Widget. A widget represents an interaction feature of an application. Applications instantiate widgets at runtime, and give them unique (in the scope of the application) widget ids. When widgets are instantiated, they are automatically registered in the IM, i.e., their description is sent to the IM. The registration process itself is hidden from the application and is done by the PuReWidgets library. Widget instances may be on-screen, or off-screen (visible on the public display, or invisible); in either case, the widget instances are able to receive input and trigger an event.

Widget Option. Widget options are independently actionable items within a widget. Most widgets have a single option, but some, for example list boxes, may have various options that users can independently select. Each widget option must have a unique widget option id in the scope of a widget.

Reference code. The IM assigns a unique (within the scope of a place) textual reference code to each widget option. Reference codes are human-readable identifiers to be used in text-based interactions, allowing users to address individual options within a widget. Additionally, places also have reference codes assigned by the display owner that, together with the reference code of the widget option, uniquely identifies an interactive feature across all places and applications of the IM.

Web GUI. The web GUI is a web-based interface that the IM dynamically generates for all applications in a given place, allowing users to interact with any widget of any application through a standard web interface.

3.3 Widgets

Widgets are capable of receiving simultaneous input from multiple users using diverse interaction mechanisms. We have studied interaction features across a large number of public display systems, and developed widgets to support the most common interaction scenarios. The following widgets are currently provided:

Button. A button allows users to trigger actions in the public display application (Figure 3a).

List box. The list box allows users to select among a set of related items (Figure 3e).

Text box. A text box allow users to input free text (Figure 3f).

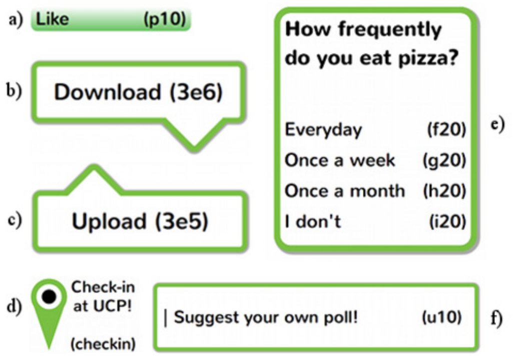

Figure 3: Default graphical representations for widgets on the public display interface.

Upload. An upload widget allows users to submit media files to the public display (Figure 3c).

Download. A download widget allows the application to provide files that users can download to their personal devices (Figure 3b).

Check-in. Check-in widgets allow users to signal their presence to the application (Figure 3d).

3.4 Interaction Mechanisms

In its current version, PuReWidgets supports four kinds of input that allow users to interact with applications: text-based input, QR codes, touch-screen interaction, a web GUI for mobile devices.

Text-based interaction includes various different input mechanisms such as SMS, instant messaging, email, Bluetooth naming, and other mechanisms where the communication is made mainly via text messages. We use an approach similar to the one used by (Paek et al., 2004) where the IM server is composed of a set of I/O modules that can receive raw input from different sources and interpret the interaction commands that are present. We define a simple command structure to address a specific widget on an application and pass it additional parameters.

Widgets in PuReWidgets are also touch-enabled. In this case, interaction is always anonymous and the widgets must be on-screen in order to be interacted with. Currently, PuReWidgets supports touch interaction with buttons, list boxes, and text boxes.

PuReWidgets also creates QR codes for individual widget options, allowing interaction with specific application features simply by scanning a visual code. Applications can use the library to create and show QR codes on the public display for any widget they have created. Additionally, display owners can also access a webpage where they can see all the QR codes associated with a particular application, and print and distribute them physically near the public display.

PuReWidgets also dynamically generates a web-based GUI for desktop and mobile devices. For each place, the IM server provides a web GUI that allows users to see the available applications in that place, and interact with any widget currently in use by any application in that place.

4 DYNAMIC UI GENERATION

In order to generate a web graphical user interface for a public display application, PuReWidgets uses the information that applications register when they create or update widgets.

4.1 Structure of a Widget

A widget holds the information necessary to render it locally at the public display, but also for rendering in the web GUI. Figure 4 shows the main data structures for a widget. The information contained in these data structures is sent to the IM during the widget registration process.

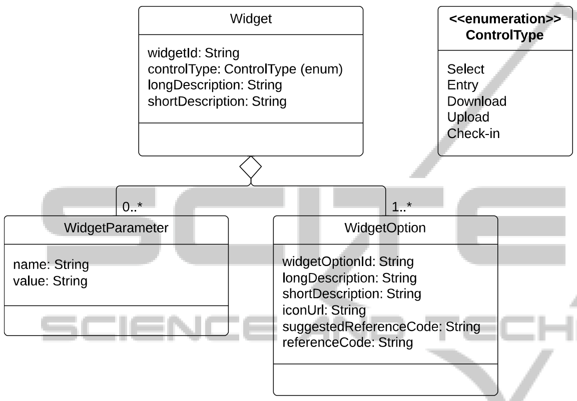

Figure 4: Widget data structures.

Widget id. The widget id is a unique id (within the application) assigned by the application that allows the IM to identify each widget.

Control type. The control type defines the kind of information that results from the interaction and the general behaviour of the widget. We have defined five control types: select - for imperative (actions) and selection widgets; entry - for data entry widgets; download - for file download widgets; upload - for file upload widgets; and presence - for check-in widgets.

Long and short descriptions. Long and short descriptions are textual labels for the widget. The long and short versions may be used differently by different widgets. For example, the list box widget uses the long description as the title of the list box in the public display interface. The button widget uses the short description as the button label. In the web GUI these descriptions are used together to provide additional contextual information to the user.

Widget options. Widget options define independently actionable items within the widget. For example, a list box widget has several widget options corresponding to each of its items. A button has only one widget option corresponding to the action that it triggers. A widget option has a unique id within the widget, and short and long descriptions that are used as labels for the options. Additionally, a widget option may have an icon associated with it. Widget options have a reference code that is assigned by the IM and that corresponds to human-readable textual code used in text-based interactions for selecting a specific item in a widget. The reference code is a random 3-letter code generated by the IM, but applications may suggest their own reference codes, that are honoured if they don't conflict with other existing widgets.

Widget parameters. A widget parameter is a name/value pair that can be associated with each widget instance. Parameters can be used in application-specific or widget-specific manners.

4.2 Rendering

When users access the web GUI for a given place, the IM automatically generates a graphical user interface, using the information about the registered widgets. For each place, the IM server provides a web GUI that allows users to see the available applications in that place, and interact with any widget currently in use by any application. It should be noted, however, that this automatic interface generation does not preclude application developers from creating a custom web or mobile interface of their applications.

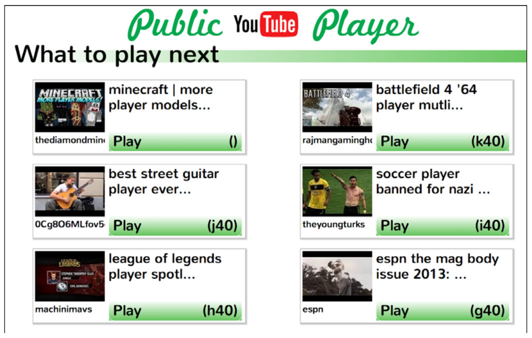
Public YouTube Player
What to play next
minecraft | more player models...
thediamondmin
Play ()
battlefield 4 '64 player mutli...
rajmangamingh
Play (k40)
best street guitar player ever...
0Cg8O6MLfov5
Play (j40)
soccer player banned for nazi ...
theyoungturks
Play (i40)
league of legends player spotl...
machinimavs
Play (h40)
espn the mag body issue 2013: ...
espn
Play (g40)

Figure 5: Public display interface of an application.

Figure 5 shows the public display interface of an application created with the PuReWidgets toolkit. This particular screen allows users to select the next video to be played. If users access the web GUI for this application, they will see an interface similar to the one depicted in Figure 6.

4.2.1 Widget Layout

The web GUI has slightly different layouts depending on the size of the mobile device, but the general structure is the same. Figure 6 shows the layout for the widget list screen in a tablet device.

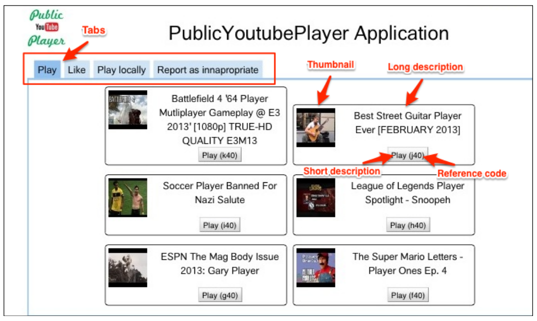
Public YouTube Player
Tabs
PublicYoutubePlayer Application
Play
Like
Play locally
Report as innapropriate
Thumbnail
Long description
Battlefield 4 '64 Player Mutliplayer Gameplay @ E3 2013' [1080p] TRUE-HD QUALITY E3M13
Play (k40)
Best Street Guitar Player Ever [FEBRUARY 2013]
Play (j40)
Short description
Reference code
Soccer Player Banned For Nazi Salute
Play (i40)
League of Legends Player Spotlight - Snoopeh
Play (h40)
ESPN The Mag Body Issue 2013: Gary Player
Play (g40)
The Super Mario Letters - Player Ones Ep. 4
Play (f40)

Figure 6: Main elements of the web GUI.

In order to provide some visual structure to the web GUI and help users to discover a specific widget, the short description is used for grouping: widgets with the same short description are grouped together under the same panel (in Figure 6, for example, the depicted widgets share the same "Play" short description).

PuReWidgets uses the various widget fields to decide how to render a widget. For example, the widget control type is used to determine what HTML controls are needed to provide the interaction (a textbox, a list box, a button, an upload form, etc.). Figure 7 shows the elements used to render a list box and a text box.

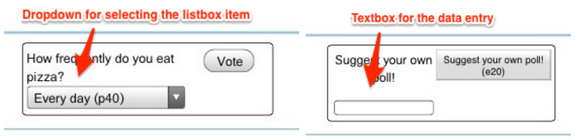
Dropdown for selecting the listbox item
How frequently do you eat pizza?
Vote
Every day (p40)
Textbox for the data entry
Suggest your own poll!
Suggest your own poll! (e20)

Figure 7: Rendering of list box and text box widgets in the web GUI. (Left: list box widget rendering; right: text box widget rendering).

In addition to grouping, PuReWidgets allows application developers to define the order of the widgets. By default, the widget id is used as the sorting field that defines the order by which widgets are rendered (left to right, top to bottom), but developers may use a special widget parameter to override the default sorting field.

4.3 User Authentication

The web GUI allows users to interact anonymously, while making sure that the system is able to distinguish different users. For this, the web GUI randomly generates a user id and nickname (in the form of "Anonymous####") the first time a user accesses the web GUI.

If users choose to identify themselves, they can do this by logging in with one of several authentication providers (the web GUI currently uses the Janrain service (Janrain, 2013), which provides a unified authentication mechanism across several providers such as Google, Facebook, Twitter, LinkedIn, etc.). In this case, the user's id and nickname are extracted from the provider's account profile.

4.4 Synchronisation

Applications built with PuReWidgets can create, destroy, and modify their widgets at any time. In order to maintain consistency between the public display interface and the web GUI, any changes in the widgets must be propagated from the application to the web GUI. Currently, the web GUI uses a polling approach where it contacts the IM periodically, asking for any changes in the current set of widgets, and modifying the rendering of the widgets, accordingly.

5 EVALUATION

We have done a preliminary evaluation of the dynamically generated user interface in a real-world deployment of a public display. We have developed three interactive public display applications using PuReWidgets for this study: the Public YouTube Player, Everybody Votes, and "Wrod" Game. The Public YouTube Player is an application that searches for, and plays YouTube videos. The Everybody Votes application allows users to vote on polls created by display owners. The Wrod Game application displays anagrams of words and invites users to guess what the word is. For this study, we used a public display that was already in use at the School of Arts of the Portuguese Catholic University. We asked a group of people from the School of Arts to interact with the display whenever they went to the bar, during two weeks. At the end of the two-week period, we interviewed participants regarding their interaction experience. Next, we present the main issues we found relative to the dynamic user interface.

5.1 Results and Discussion

In general, participants were able to use the web GUI without major difficulties to interact with the available public display applications. There were, however, two issues that participants identified during this study.

5.1.1 Asynchronous Interaction

The first issue was about the asynchronous interaction model. PuReWidgets allows users to send input to an application, even if the application is not running on the public display at the time. In these cases, the application will only react when it comes back on-screen. However, some participants expected applications to immediately appear on the display if they interacted with it. One way to mitigate this problem is to provide better feedback to the user about what is going to happen with his input. In the version used during the study, the web GUI showed only a standard "input sent" message as feedback to the user's interaction. As a result of this study, we have enhanced the feedback to provide more information. The current version now detects if the target application is on-screen or off-screen and provides different messages to the user. If the application is on-screen, it instructs the user to look at the display to see the result of his interaction. If the application is off-screen it informs the user that it may take a while for him to be able to see the reaction of the application (currently, it's not possible to predict how long it will take). We have also planned a new version of the web GUI that allows applications to customise the feedback message

5.1.2 Rapidly Changing Widgets

The second issue is related to applications that change their widgets very rapidly. To limit the number of persistent connections to the server, the web GUI uses a polling approach to periodically ask the IM server for updates about the application's widgets. For applications that change their interface frequently, by adding, removing, or changing the description of existing widgets, this polling approach results in temporarily out-of-sync interfaces. This was noticeable in the Wrod Game application, which changes the description of the text box widget to reflect the current anagram. Frequently, the anagram displayed in the web GUI was not the same as the one displayed in the public display, leading users to submit wrong guesses. This was not a major problem, as the text box on the web GUI does not necessarily need to display the anagram: users would most likely look at the public display to see the letters. Still, this problem may be addressed with persistent connections, making sure the web interface is updated at a fast enough pace.

6 CONCLUSIONS

We have presented a widget toolkit for the development of web-based interactive public display applications, focusing on the dynamic generation of web graphical user interfaces. This toolkit provides high-level controls that abstract the input from several heterogeneous interaction mechanisms, allowing programmers to focus on the interaction features of their applications, instead of on the low-level interaction details. In addition, the toolkit is able to generate rich interfaces that allow users to discover and interact with the applications in a given place. We have presented our approach, and a preliminary evaluation that resulted in the identification of possible improvements to the user interfaces.

Interfaces for (smart) mobile devices can provide simple, familiar, and rich interaction with public displays applications. As the usage of smartphones increases, and users become more familiar with them, we can expect that these devices will also become a natural choice for interacting with public displays. Additionally, the dynamically generated graphical user interfaces also provide *discoverability* of the interaction possibilities with a particular public display application. A system that dynamically generates mobile graphical user interfaces for public display applications can be an important component in a public display network.
Even if application developers are not willing to invest in a customised mobile version of their public display application, the dynamically generated interface provides a low-effort alternative for developers. This may be an important adoption factor for public display applications, and for their wider dissemination.

ACKNOWLEDGEMENTS

Jorge Cardoso has been supported by "Fundação para a Ciência e Tecnologia" (FCT) and "Programa Operacional Ciência e Inovação 2010", co-funded by the Portuguese Government and European Union by FEDER Program and by FCT training grant SFRH/BD/47354/2008.

REFERENCES

Cardoso, J. C. S., & José, R. (2012). PuReWidgets: a programming toolkit for interactive public display applications. In S. R. José Creissac Campos, Simone D. J. Barbosa, Philippe Palanque, Rick Kazman, Michael Harrison (Ed.), *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '12* (p. 51). New York, NY, USA, Denmark: ACM Press. doi:10.1145/ 2305484.2305496.

Davies, N., Langheinrich, M., Jose, R., & Schmidt, A. (2012). Open Display Networks: A Communications Medium for the 21st Century. *Computer*, *45*(5), 58–64. doi:10.1109/MC.2012.114.

Finance, Y. (2012). Market Research Projects Smartphone Market Growth at 19% CAGR Through 2016. Retrieved from http://finance.yahoo.com/news/market-research-projects-smartphone-market-080800406.html.

Hodes, T. D., & Katz, R. H. (1999). A document-based framework for internet application control. In *Proceedings of the 2nd conference on USENIX Symposium on Internet Technologies and Systems - Volume 2* (p. 6). Berkeley, CA, USA: USENIX Association. Retrieved from http://portal.acm.org/ citation.cfm?id=1251480.1251486.

Janrain. (2013). Janrain - user management platform for the social web. Retrieved from http:// www.janrain.com/

José, R., Cardoso, J., Alt, F., Clinch, S., & Davies, N. (2013). Mobile applications for open display networks: common design considerations. In *Proceedings of the 2nd ACM International Symposium on Pervasive Displays – PerDis '13* (pp. 97–102). ACM. Doi:10.1145/2491568.2491590.

Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., & Pignol, M. (2002). Generating remote control interfaces for complex appliances. In *Proceedings of the 15th annual ACM symposium on User interface software and technology – UIST '02* (p. 161). New York, New York, USA: ACM Press. Doi:10.1145/571985.572008.

Olsen, D. R., Jefferies, S., Nielsen, T., Moyes, W., & Fredrickson, P. (2000). Cross-modal interaction using Xweb. *Proceedings of the 13th annual ACM symposium on User interface software and technology – UIST '00*, *2*, 191–200. Doi:10.1145/354401.354764.

Paek, T., Agrawala, M., Basu, S., Drucker, S., Kristjansson, T., Logan, R., … Wilson, A. (2004). Toward universal mobile interaction for shared displays. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work* (pp. 266–269). New York, NY, USA: ACM. Doi:10.1145/1031607.1031649.

Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., & Winograd, T. (2001). Icrafter: A Service Framework for Ubiquitous Computing Environments. In *Proceedings of the 3rd international conference on Ubiquitous Computing* (pp. 56–75). London, UK: Springer-Verlag. Retrieved from http://portal.acm.org/ citation.cfm?id=647987.741344.

Roman, M., Beck, J., & Gefflaut, A. (2000). A device-independent representation for services. *Proceedings Third IEEE Workshop on Mobile Computing Systems and Applications*, 73–82. doi:10.1109/ MCSA.2000.895383.

Venners, B. (2005). The Jini ServiceUI API Specification. Retrieved from http://www.artima.com/jini/serviceui/ Spec.html.