Evaluation of a programming toolkit for interactive public display applications

Jorge C. S. Cardoso[1,2]
[1]CITAR/School of Arts
Portuguese Catholic University
Rua Diogo Botelho, 1327
4169-005 Porto, Portugal
jorgecardoso@ieee.org

Rui José
[2]Centre Algoritmi
University of Minho
Campus de Azurém
4800-058 Guimarães, Portugal
rui@dsi.uminho.pt

ABSTRACT
Interaction is repeatedly pointed out as a key enabling element towards more engaging and valuable public displays. Still, most digital public displays today do not support any interactive features. We argue that this is mainly due to the lack of efficient and clear abstractions that developers can use to incorporate interactivity into their applications. As a consequence, interaction represents a major overhead for developers, and users are faced with inconsistent interaction models across different displays. This paper describes the results of the evaluation of a widget toolkit for generalized interaction with public displays. Our toolkit was developed for web-based applications and it supports multiple interaction mechanisms, automatically generated graphical interfaces, asynchronous events and concurrent interaction. We have evaluated the toolkit along various dimensions - system performance, API usability, and real-world deployment - and we present and discuss the results in this paper.

Categories and Subject Descriptors
D.2.2 [Software Engineering]: Design Tools and Techniques – Software libraries, Modules and interfaces;

General Terms
Design, Experimentation, Human Factors

Keywords
Public display applications; interaction abstraction; toolkit

1. INTRODUCTION
Public digital displays are moving towards open display networks "*that are open to applications and content from many sources*" [5]. This movement entails 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. Applications are a central aspect in the concept of open display network and have been the focus of recent research that addresses challenges such as the distribution [4], allowing users to control them on the public display [19], and dealing with privacy [13]. Another challenge is providing application developers with appropriate tools to create interactive public display applications. This work is concerned with this latter challenge.

Interaction with public displays can be very diverse and use many types of interaction mechanisms. SMS [14], Bluetooth naming [11], touch-screens [1], gestures [21], mobile apps [13], and other mechanisms have been used to interact with public displays. In an open display network, applications will be developed by third parties, which will want to distribute them via application stores [4], or other channels, so that applications can be selected to run on a variety of public displays, managed by different display owners. Interactive applications will need to take advantage of the locally available interaction resources, which may vary between displays. Developers need tools that help them incorporate interaction features irrespective of the concrete interaction mechanisms that will be available in a given display. While interaction can easily be achieved for a specific display system with a particular interaction modality, the lack of proper interaction abstractions means that there is too much specific work that needs to be done outside the core application functionality to support even basic forms of interaction. This usually leads to inconsistent interaction models across different displays and, as a result, people are not able to develop any expectations and practices regarding their previous experiences with public displays.

It seems reasonable to make an analogy between this situation and the time when desktop computer programmers had to make a similar effort to support their interaction with users. This problem was addressed with the emergence of reusable high-level interaction abstractions, such as the WIMP model and its associated controls, that provided consistent interaction experiences to users and shielded application developers from low-level interaction details [18]. Nowadays, developers benefit from toolkits that provide user interface widgets that deal with input, and encapsulate behaviour and visual appearance. Users have also benefited, because they have learned to interpret the affordances of these widgets in a way that enables them to more easily tackle new interfaces by building on previous experiences.

An interaction toolkit for public displays should thus provide appropriate high-level controls for public displays and corresponding graphical representations for those controls. In this work, we assume that public displays may exist in a multi-user interaction environment [20] and so a toolkit must support concurrent interactions. To cope with the various input possibilities, it should also support and abstract input from multiple heterogeneous interaction mechanisms. We have built a toolkit for web-based interactive public display applications that meets those requirements, and simplifies the incorporation of interactive features into applications.

MUM '13, December 02 - 05 2013, Luleå, Sweden

http://dx.doi.org/10.1145/2541831.2541834

The main contribution of this paper is an evaluation of this toolkit along various dimensions. First, we evaluate the system's performance. Then we evaluate the API's usability from the perspective of independent programmers. Finally, we evaluate the system in a real-world deployment of a public display running various applications built with the toolkit. Results show that programmers are able to understand the concepts behind the toolkit and apply them to create interactive public display applications, and that users are able to understand the interaction with those applications.

2. RELATED WORK

Interactive public displays have motivated much research in the last years that has sought to explore the possibilities of public displays in various application areas. We first present work on interaction mechanisms for public displays, giving an idea of the plethora of possibilities for interaction and of the challenge of integrating such disparate forms of interaction into an interaction toolkit for public displays. We then present existing middleware and toolkits that provide already some form of interaction abstraction.

2.1 Interaction mechanisms

Many interaction mechanisms have been proposed for public displays. For example, Rohs [17] has implemented a set of widgets for visual marker-based interaction that allows users to activate actions or select options encoded in a visual marker and send it via SMS (using a custom mobile application). The visual marker encodes the type (menu, radio or check button list, sliders, etc.) and layout (vertical or horizontal menu, number of options, etc.) of the widget, so that the mobile phone application can immediately superimpose graphical information about the currently selected item or value. Dearman & Truong [7] developed Bluetone: a widget that is activated through dual tone multi-frequency (DTMF) over Bluetooth (BT). Users interact with an application by changing the BT name of their device to a system command, wait for the display to pair with the user's phone as an audio gateway, and then pressing the keys on the keypad of their phone. Bluetone supports several users, being limited only by the BT protocol. This widget is limited to the DTMF interaction mechanism, and has been developed for an environment where a single application executes at a time. Graphically, it consists of a single widget that encapsulates all the interactive features of the application. Cheverst et al. [3] explored the file exchange functionality of the BT protocol (OBEX) to allow users to exchange photos with office door public displays by sending and receiving files from their mobile phones. SMS interaction has also been used frequently with public display applications. Jumbli [14], for example, is a word puzzle game that allows users to form words with the letters presented on the public display and send those words, via an SMS message with the word sent to a pre-defined number. Bluetooth naming is another approach for providing interactivity to public displays. Lancaster University's e-Campus display system [6] or Instant Places [11] explored Bluetooth naming as an explicit input mechanism. BT scanners on each display continually discover devices in the vicinity and send these sightings information to a content scheduler. To interact, users need only to change the BT name of their personal mobile device using a pre-defined command structure and wait for the BT scanner in the place to pick up the change.

All these are good examples of how to provide users with specific interaction mechanism to interact with public displays. However they do not address the question of providing interaction abstractions to applications so that developers don't have to deal with the particularities of each interaction mechanism. In all the previous examples, the assumption was that a specific mechanism would be available.

2.2 Input Middleware and Toolkits

There has also been some work on input middleware for ubiquitous systems that aims at providing some level of abstraction. Magic Broker [9], for example, is an event-based input infrastructure that allows applications to subscribe to input from different sources such as SMS, Voice (using Voice XML), and web interactions. However, it provides a lower level of abstraction than the one we wish to achieve. For example, it does not define how users can address individual applications or interactive features. Other input middleware, such as ICON [8], allow the dynamic mapping of input devices to applications. However, these mappings are created for individual applications, and they work only for local input devices. Also, it does not define high-level controls suitable for public display applications. The Proximity Toolkit [15] can be used to rapidly prototype proxemic interactions of users with various kinds of displays. It provides high-level proxemic measures, such as orientation, distance, motion, identity, and location of users and devices in the environment. Being oriented towards proxemics, the toolkit provides abstractions such as "is pointing at", "is facing", "is touching", which can be used to support various forms of interactions. It does not address, however, interactions such as sending text, downloading or uploading files, or the integration of mobile devices to interact with applications. Additionally, the toolkit relies on specific infrastructure equiment such as a motion capture system. Hardy & Alezander [10] have developed a toolkit for projected displays that supports the creation of multiple virtual displays that can be mapped to various surfaces. Users can interact with the individual displays by touching, gesturing, or placing objects on the surface. Interaction is detected with a Kinect device. The content projected on a surface consists of standard web content, possibly augmented with the toolkit's code for detecting interactions. This toolkit is targeted at a very specific interaction style and, thus, does not provide many of the high-level controls that are generally required for public displays.

3. DESIGN AND IMPLEMENTATION

3.1 Requirements

A toolkit for interactive public display applications must address a number of requirements that are specific to the interaction environment where these applications will run. In previous work we have thoroughly investigated this [2], but for convenience, we briefly outline those requirements here.

Public display interaction controls. The toolkit should provide developers with various types of controls specifically designed to support the interaction tasks that are relevant for public display interaction.

Standard graphical representations. The toolkit should provide graphical cues so that users can easily understand that a public display is interactive and understand its affordances.

Concurrent and shared interaction. The toolkit should support multiple, concurrent, interactions from different users.

Multiple interaction mechanisms. The toolkit should support and abstract several heterogeneous interaction mechanisms.

Ubiquitous interaction. The toolkit should allow users to interact with an application at any time, regardless of whether the application is currently showing content on the public display.

Additionally, while designing the API of the programming library, we had a number of design goals in mind:

Low learning threshold. We wanted an easy to program system, integrated into the regular application development cycle. We did not want to introduce additional steps in the typical development process, nor change too much the existing ones.

Dynamic interfaces. We wanted to allow applications to have dynamic interfaces where widgets can be created, changed, and removed at any time. We did not want to introduce compile time mechanisms that would, for example, force programmers to produce separate interface descriptions with the only purpose of being used for generating the web GUI (an automatically generated mobile user interface for interacting with the display applications).

Flexible. We wanted an easy to program system, where developers could focus on the high level aspects of the interaction, but also have control over fine details of the interface, such as the graphical appearance of the widgets.

3.2 Architecture and Implementation

We targeted our toolkit – PuReWidgets – at web-based public display applications. Even though the central concepts could be applied to other platforms, the web has a number of advantages including the ease of deployment, distribution and updating, easy access to extensive web content, and multi-platform support. Web applications also have some limitations but given the continuous advances in browser technology, we can expect that these will continually decrease.

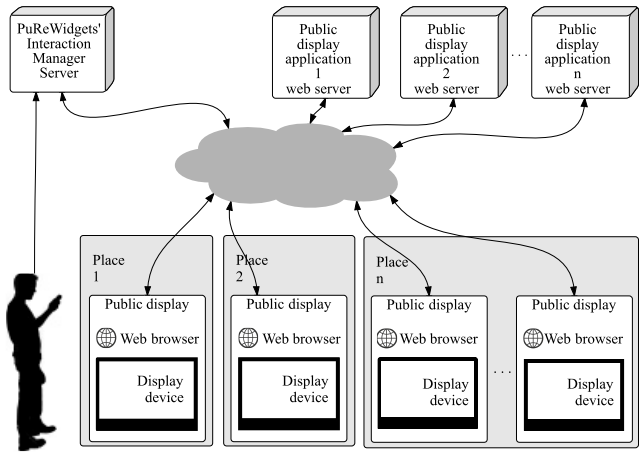


Figure 1. Physical components of the system's architecture.

The PuReWidgets system 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. 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 PuReWidgets 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.

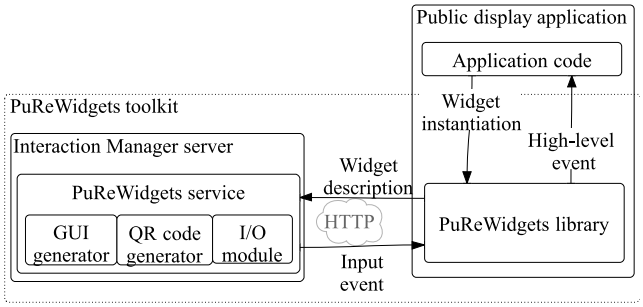


Figure 2. Logical components of the toolkit.

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. The set of HTML, CSS, and Javascript files are then deployed to a standard web server.

3.2.1 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. It is also capable of dynamically generating web-based graphical user interfaces for desktop and mobile platforms (GUI generator), QR codes for individual widget interaction (QR code generator), and accepting input from various text-based mechanisms such as SMS and email (I/O module). 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. One of the uses for off-screen applications is to support customization at any time, similarly to [13]: users can send input to off-line applications to convey their preferences; when later the application appears on the display it can reflect those preferences.

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.

3.2.2 PuReWidgets library

The PuReWidgets library is the toolkits' interface with programmers. It is a web client library for the Google Web Toolkit (GWT) platform that offers programmers various widgets that abstract user interaction into high-level interaction events to applications. The library transparently handles communication with the IM for various bookkeeping operations, including registering the widgets, receiving user input, and forwarding input to the correct widget. It also generates system-level graphical input feedback on the public display.

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).

Upload. An upload widget allows users to submit media files to the public display application (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 the application that they are present (Figure 3d).

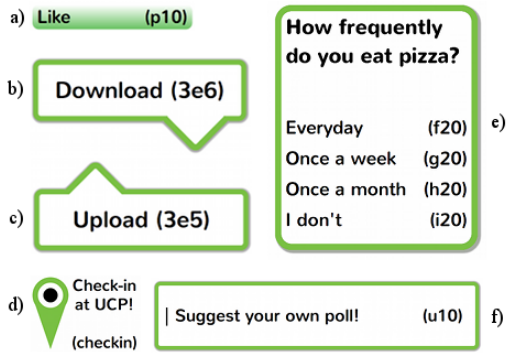


Figure 3. Default graphical representations for widgets.

3.2.3 Interaction mechanisms

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

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. [16] 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: *<place reference code>.<widget reference code> <additional parameters>*. The additional parameters are widget-specific and not all widgets require them. For example, to send a poll suggestion to the text box of Figure 3.f, in the UCP place, a user would send "*ucp.u10 How often do you read books?*" to the address (SMS number, email address, etc.) of the display system, which must somehow be advertised to users.

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. The information registered when the application instantiated the widget is used to determine what web controls are needed to render the widget in the web GUI. The web GUI allows users to interact anonymously, or logged in via one of several authentication providers (Google, Facebook, Twitter, LinkedIn, etc.).

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. Display owners also have access to the QR codes through a web interface. For example, display owners may want to draw attention to a specific feature of an application, or to a new application, by printing and distributing the QR codes associated with those features. QR codes embed the place id, application id, widget id, and option id in an URL that points to the IM server. When accessed, the IM generates a web interface for interaction with the specified widget.

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.

4. EVALUATION

Given the novelty of the application area of our toolkit, we performed an evaluation with the aim of covering various dimensions, making sure we had a generally viable system. We evaluated PuReWidgets along three major dimensions. First, we looked at the system's performance to determine if the current implementation had any evident bottlenecks or limitations. Then, we evaluated the API's usability to determine if programmers faced any major difficulties in understanding the toolkit's concepts and structure. Finally, we performed a real-world deployment of a public display to determine what issues might arise for users of applications built with our toolkit.

4.1 System Scalability

The IM is a central component of the PuReWidgets toolkit because it handles all the interactions that happen with a public

display application. It is, therefore, important to determine if its implementation has any performance bottlenecks or limitation when handling various simultaneous places and applications.

4.1.1 Procedure

We measured the resource usage and the execution time of the various requests to the IM for an increasing number of applications. The IM is implemented over Google's Appengine[1] which measures the amount of resources that an application uses – CPU, API calls, bandwidth, and storage are the general resource types measured by Google. For the IM, the most relevant resources to measure are the *frontend instance hours* – the sum of the running time of the various server instances that Appengine automatically creates to handle the server load; *datastore write and read operations* – the number of low-level operations over the application's datastore; *channels* – the number of persistent connections between a Javascript client and an Appengine server application. Appengine assigns each resource a daily free quota above which applications are billed for the resources they consume.

To estimate the resource needs of the IM, we developed a testing public display application, which executes the following steps: 1) creates five button widgets (immediately after startup); 2) sends/receives one input to one randomly chosen widget (30 seconds after startup); 3) optionally, deletes one of its own widgets (60 seconds after startup); The base load consisted in one place running 10 application instances, scheduled to run consecutively, giving each application 3 minutes of display time, during a period of 10 hours. Five of those applications were configured to not delete any widget in step 3. This simulates a fairly loaded display with 10 interactive applications that continually create and delete widgets, and receive input. This has impact on the performance of the IM because the PuReWidgets library needs to make requests to the IM to carry out these operations.

We simulated an increasing number of places, up to 24, with the same application configuration. At the beginning of each simulation, we reset all the data in the IM server. In addition to the resource usage, we also measured the time it took for each request to the IM to execute.

4.1.2 Results

The results for the quota usage for the various resources and the request execution time, during the 10-hour simulation period, are shown in Figure 4 and Figure 5. Quota usage results are indicated as a percentage of the daily free resource quota provided by Appengine.

The plot of Figure 4 shows a linear increase in the datastore write and read operations and on the number of channels, with the increase of the number of places. The instance-hours resource usage remained fairly constant and near the real time percentage of the simulation time relative to the number of hours in the free quota (10 hours equals 36% of the 28 hours of the free quota). This means that the server was below its CPU capacity and was able to handle most of the requests with a single application instance.

Figure 5 shows the box plots of the execution time of the various requests. Visual inspection of the box plots shows that the execution time did not vary much with the number of places. This is congruent with the instance-hour usage and indicates that CPU is not greatly affected by the number of places.

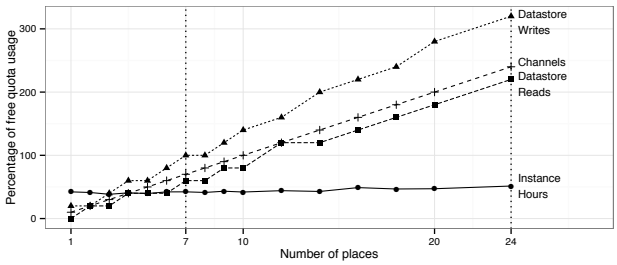


Figure 4. Resource usage.

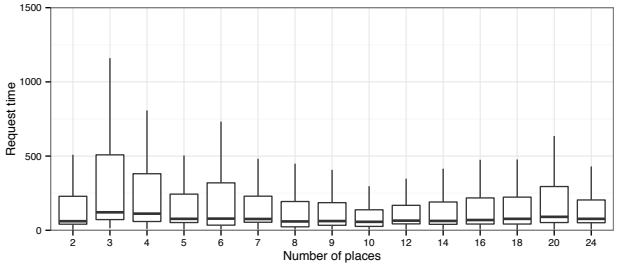


Figure 5. Request execution time.

4.2 API Usability

The second dimension in our evaluation was the usability of the toolkit's API. We conducted a usability evaluation of the API by asking a group of programmers to use our toolkit through a series of pre-defined programming tasks in a lab environment. We were interested in assessing if programmers understood the application life-cycle and associated callback methods, the various widget related tasks (creating, deleting, extending, and styling widgets), and the various input feedback tasks (changing the default behaviour, and styling the feedback panels). We also wanted to find out possible problems with the online documentation (programmer's guide, and API javadocs) and how to improve it.

4.2.1 Participants

Six programmers participated in our study, selected from a computer-engineering course. All participants were male, aged between 21 and 24 years. All participants were experienced programmers with at least four years (six, on average) of experience. They all had experience with the Java programming language, Eclipse IDE, and web technologies (HTML, CSS, Javascript), but none was familiar with the Google Web Toolkit framework for web development. Participants were rewarded monetarily for their collaboration.

4.2.2 Procedure

The session had three phases: an introduction to the toolkit, a set of programming tasks, and a final questionnaire. In total, the session lasted for about 4 hours. In the introduction, we presented the study and its purpose, and we introduced participants to the toolkit. This presentation followed roughly the sequence of topics of the Getting Started section of the wiki documentation on the toolkit's Google code web page[2], which explains: the main concepts around PuReWidgets, how to setup the development environment, a HelloWorld application, and how to test and

[1] *http://code.google.com/appengine/*

[2] https://code.google.com/p/purewidgets/wiki/TableOfContents

deploy a PuReWidgets application. Participants were not required to set up their development environment, as this was done previously for them in the laboratory computers, but they were asked to import, run and test a HelloWorld application during this presentation.

In the next phase, we asked participants to perform four programming tasks using our toolkit. These tasks were designed so that participants would have to use particular features of the toolkit such as creating and removing widgets, using the web GUI to test the interaction with their application, deal with input from different users, deal with on-screen and off-screen widgets, and customize the input feedback messages of the toolkit. Task 1 – HelloWorld, was a warm up task that consisted in changing the existing HelloWorld application. The HelloWorld application consisted of a single button placed in the centre of the screen, which toggles the background colour between white and black, when activated. The task consisted in adding a listbox widget with several colour options so that users could first select the background colour from the listbox and then activate the button to effectively change the background colour of the application. Task 2 – SlideShow, and subsequent tasks consisted of creating a picture slide show application, based on a given skeleton project that included functions to fetch pictures from a picture service, and functions to display a set of thumbnail images on the screen, but no user interaction code. In task 2, participants were asked to add a button to each thumbnail so that the corresponding photo was displayed in large view, the thumbnail was removed from the list, and a new thumbnail added, when the button was activated. In task 3 – Multi-user, participants should change the previous application in order to only display the large view image after two different users had selected the same thumbnail. In task 4 – Enhanced feedback, participants were asked to change their previous implementations so that the slideshow application would display specific input feedback messages when users interacted. We asked participants to complete the tasks using the wiki and javadoc documentation whenever needed, but also to freely ask the researcher for help. This step was video-recorded for later analysis of the main difficulties and comments made during the programming tasks.

In the last phase of the study, we asked participants to fill in a questionnaire about the PuReWidgets toolkit and the tasks they had just completed.

4.2.3 Results

We collected three main sources of data regarding this study: the participants' source code, the results from the final questionnaire, and comments from the video recording of the session.

4.2.3.1 Source code

Inspection of the source code produced by the various participants revealed that in general all participants were able to accomplish the tasks. However, tasks 3 and 4 revealed some difficulties.

Task 3 required participants to count the number of different interacting users for each button, and make the application react only after two different users had activated the button of a given thumbnail. Four participants mistakenly used the *getNickname*() method instead of the *getUserId*() method from the input event object to get the id of the user. The nickname is not guaranteed to be unique, so using that method would cause the application to behave incorrectly in some cases, although this was hard to detect during the short coding session. Ignoring this mistake, only one participant was unable to complete the task. This participant correctly used the *getUserId*() method, but did not complete the logic to count the number of users that had activated a given button.

In task 4, three of the participants called the feedback configuration methods inside the button event callback, causing the first feedback to be displayed with default messages (for the subsequent inputs, the feedback messages would have been set correctly). The correct point to call these methods would be right after instantiating the button widgets.

4.2.3.2 Questionnaire

The questionnaire was meant to assess if participants felt they understood the various concepts and functions of the toolkit, their opinion regarding the documentation, and their confidence about using the toolkit to create a public display application.

We asked participants to rate on a 1 (strongly disagree) to 5 (strongly agree) scale how much they agreed with a set of statements, distributed by 4 groups:

A) "I understood the concept of [Place, Application, Widget, Feedback] as used in PuReWidgets."

B) "I understood how to [Create, Delete] widgets", "I understood how to modify input feedback."

C) "I think the documentation was sufficient", "I think the Javadoc was clear", "I think the Wiki was clear".

D) "I believe I could create a public display application using PuReWidgets by myself."

Results are presented as box plots in Figure 6. The left and right of the boxes represent the first and third quartiles, the band inside the box represents the median, the whiskers represent the lowest and highest values within 1.5 IQR of the lower and higher quartile, and dots represent outliers. Colours are used to represent the four groups of questions. Results clearly indicate a general positive assessment of the toolkit by programmers.

4.2.3.3 Comments and observations

The third source of data from the API usability study was the various comments made by the participants during the programming session (comments were collected from the transcription of the video recording and from written comments requested at the end of the session). We analysed and grouped the comments into 3 categories: documentation improvement, new toolkit features, and confusing aspects of the toolkit.

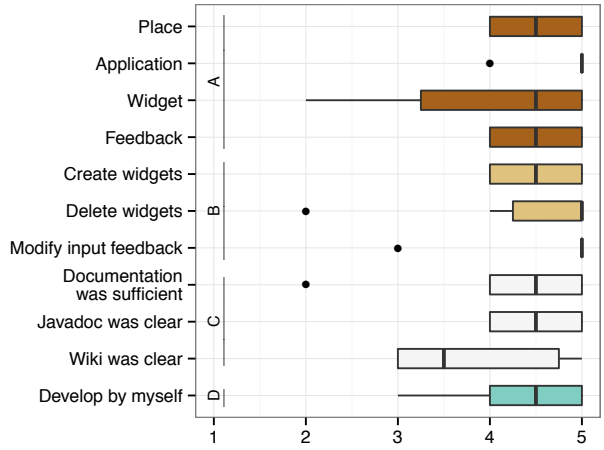


Figure 6. Box plots of the answers to the questionnaire.

Regarding the documentation improvement, some comments focused on very specific aspects such as "*there should be a*

reference that the 'delete' operation needs to be explicit", "*the input feedback on/off screen is automatic, and there should be a reference to that*", and "*the documentation should indicate which characters are valid for the widget ids*". Other comments were more general, requesting more examples and tutorials: "*The 'getting started' section of the wiki should have one more example besides the hello world (a more complex example) . . .*", and "*Regarding the general documentation, the wiki could me more enlightening. . . . it would be useful to have one tutorial that covered these more common operations (creating widgets, removing them, inspect the input and produce output) . . ..*"

Regarding the toolkit features, participants wished the toolkit offered things such as automatic widget removal "*there should be something like a garbage collector*", automatic widget id sanitization: "*It would be also interesting that, when an invalid id was entered, that id was 'cleaned' internally . . . to avoid problems during development.*", a loading indicator on the dynamically generated web GUI: "[the web GUI could be improved by] *show a loading status . . . just to let users know that it's refreshing* [the list of widgets]".

Participants also expressed their confusion with some aspects of the toolkit, particularly about two related aspects: the configuration of the feedback messages and the concept of on-screen and off-screen widgets: "*About task 4 I was a bit confused about how to call the setOnScreenFeedback or the setOffScreenFeedback* [methods] *because I thought it was necessary to detect if the widget was on screen or off. However, after I found out you just call the two and that the toolkit does the rest I thought it was interesting.*"

4.3 Real-World Deployment

The final part of our evaluation involved a public display deployment. The goal of this study was to find out what problems would arise during a real-world deployment of interactive applications developed with PuReWidgets. In this phase, we were particularly interested in finding out any issues related to the user interaction process: finding out that an application was interactive, interacting through various mechanisms, and determining the result of the interaction.

4.3.1 Display configuration

We developed three interactive public display applications during this phase: the Public YouTube Player, Everybody Votes, and "Wrod Game".

The Public YouTube Player is an application that searches for, and plays YouTube videos. It provides several interaction features to users such as "liking" videos that have been recently played; getting the URL of a recently played video to play it in their devices; selecting a video to be played next from the list of search results; and reporting inappropriate videos. The application is composed of four screens which iterate sequentially: the selected video played in full screen, the recently played, a tag cloud of the current keywords used for searching, and a list of videos available to play next (Figure 7a). The display owner can customize this application using a web interface that allows setting various parameters such as the duration of each screen, the list of initial search keywords, the maximum duration of the videos, and other YouTube search parameters.

The Everybody Votes application allows users to vote on polls created by display owners. It is composed of three screens: a screen that iterates through the open polls (Figure 7b), showing the poll question, possible answers, and time left before the poll closes; a screen that iterates through the closed polls and shows their voting results; and a suggest box screen enticing users to suggest their own polls (which will be moderated by the display owner).

The Wrod Game application displays anagrams of words and invites users to guess what the word is (Figure 7c). When players guess the correct word they earn points proportionally to the word size and can see the definition of the word they guessed correctly.

For this study, we used an existing display placed at a bar of the School of Arts - Portuguese Catholic University, which was being used only for non-interactive content. We created a content schedule that included the three interactive applications developed with PuReWidgets described previously, and some non-interactive applications.

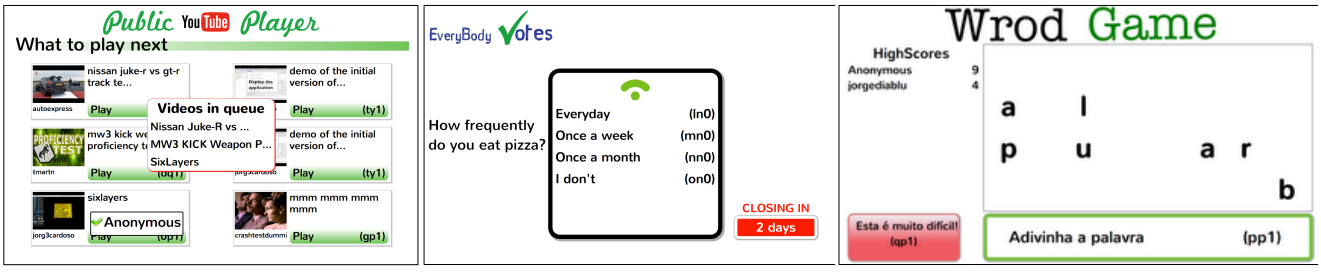
4.3.2 Participants

In order to have a group of users interacting periodically with the display, we asked a group of people from the School of Arts to interact with the display whenever they went to the bar, during a two-week period. Four people answered our email request for participants: two teachers, and two PhD students.

4.3.3 Procedure

We had an initial meeting with each participant where we explained the purpose of the study and the procedure. We told participants that the display had three interactive applications and asked them to interact with those applications as much as possible during their normal visits to the bar. We asked participants to try to use at least two interaction mechanisms (SMS, Web page, Email, or QR codes). We also asked them to take notes of problems they found during their interactions with the display, suggestions of things to improve, or just general comments.

We did not explain to participants how to interact with the display system. During the study, we distributed at the bar printed flyers with interaction instructions. We also distributed two QR code flyers for two specific polls of the EveryBody Votes application: one about the best city to live in Portugal, and another about which team would win the championship that year. These flyers



a) b) c)

Figure 7. Sample screens of the public display applications.

were distributed regularly – usually every other day – at the bar. These flyers, in addition to the public display application that shows how to interact, were the only source of interaction instructions for the participants of the study.

At the end of the two-week period, we met again with participants and interviewed them. The interview was unstructured but we had a set of probe questions (did you understand how to interact with the display?, with which applications did you interact?, which mechanisms did you use to interact with the display?, which one did you prefer?, which problems did you encounter?) to get participants to elaborate on the difficulties they encountered and on possible solutions. The interviews were audio-recorded and later analysed.

4.3.4 Results

Altogether, participants used all the available interaction mechanisms to interact with the display, and they were generally able to understand and use the display system. Participants successfully interacted with the existing applications via SMS, email, web and QR codes to activate buttons, to send text to text boxes, and choose options from list boxes. However, they faced a few initial difficulties. There were three main issues identified by participants during the interaction. We present the main issues and also some of the suggestions for improving the system.

The first issue was about how to interact. Two participants reported some difficulty because the display system addresses (web address, SMS, email) were not always visible. The Interaction Instructions application was only shown for brief periods of time, and printed flyers were not always available. Since participants did not memorize the display system addresses, on some occasions they were unable to interact. However, after seeing the instructions, participants said they had no difficulty in sending input to the display system, as the instructions were clear and the steps easy to follow.

Another issue was related to the asynchronous interaction model supported by PuReWidgets. Although users did not express any difficulty in understanding the reaction of the system in the cases were they were interacting with an on-screen application, one participant pointed out his confusion when interacting with off-screen applications. "*Sometimes it* [the display] *was slow to react. Sometimes it reacted immediately, other times it took a lot of time.*" This participant's mental model of the system was that interacting with a particular application would cause the application to immediately appear on the public display to react to his input. This caused him to understand the lack of feedback on the public display as a system error, and try to send input again.

Another issue was the interaction with public display applications when away from the public display. Although we did not encourage participants to interact with the public display applications from the desktop computers in their offices in the school, some participants did so. One participant expressed his confusion about his interactions with the Wrod Game application. "*for one word, I tried ever combination I could think of, but it didn't change in the application* [web GUI]. *I was in doubt about whether it* [the input] *really got there*". This happened because the web GUI does not reflect changes to the application's widgets immediately.

Participants also had some suggestions to make the system easier to use. A common suggestion was to put up a poster with printed instructions permanently next to the public display so that they would be always accessible, instead of having to rely on the printed flyers, which were not always available. A final suggestion by one participant was to have simpler reference codes for SMS interaction: "*for example, on the football championship poll, instead of having arbitrary references codes, why not simply use the football team names?*"

5. DISCUSSION

The toolkit met its goal of achieving a low learning threshold as can be attested by the programming study in which programmers were able to use the toolkit after a short introduction, and by the real-world deployment, in which users were able to interact with the developed applications without any major difficulties. It also met its goal of being flexible, supporting various interaction mechanisms and giving fine control over the visual appearance and behavior of the interaction features of public display applications.

5.1 Programming model

The programming model of PuReWidgets makes it easy for programmers to start developing public display applications. Even though some concepts about public display applications are not familiar to programmers, the fact that the development cycle is very similar to that of standard web applications makes it easy for them to start developing. Even after only a very short contact of about 4 hours with the toolkit, participants in the programming study were confident that they could create an interactive public display application by themselves using PuReWidgets.

In general, programmers stated they understood the concepts behind the toolkit, and the results from the programming tasks show that they were able to successfully apply them. Perhaps the most salient problem that the programming study identified was the confusion about configuring the feedback for on-screen and off-screen widgets. This confusion was explicitly mentioned by one participant and is also apparent in the analysis of the source code for task 4 – only one participant stated he did not complete the task, but in fact three participants failed to correctly configure the feedback messages. The main issue to be addressed in a future version of the toolkit is the documentation regarding the concept of on-screen and off-screen widget. We did not include in the questionnaire an explicit question about this concept, but we believe participants factored it together with the concept of widget, which accounts for a slightly lower score of the statement about the concept of widget. Traditional desktop widgets are associated with graphical objects that can only be interacted with when they are visible on the display, so participants may have had some difficulty understanding that invisible widgets can still be interacted with and generate interaction events. The PuReWidgets documentation must draw special attention to these differences in order to facilitate developer's adaptation to the concept of widget as used by PuReWidgets.

Another issue is conveying more precisely why the two graphical states of a widget – on-screen, and off-screen – may require different input feedback information and how that information can be configured. Currently, the documentation (wiki and javadoc) does not provide a clear listing of all input parameters that can be used in the feedback information, which may account for some of the participants' difficulty with task 4. Finally, the documentation is currently missing a clear explanation of the life-cycle of an input event. In task 4, three participants used the correct methods to configure the feedback, but they failed to successfully complete the task because they invoked those methods in the wrong place. This mistake may be attributed to by the lack of documentation regarding at which point the input feedback is triggered. Currently, input feedback is displayed immediately before the application receives the event, so

configuring the feedback message inside the callback method has no effect on the first feedback for that widget.

5.2 Mental models

PuReWidgets introduces some new concepts that application users must deal with and understand. The fact that participants of the real-world deployment study expressed no particular difficulty in using the various interaction mechanisms, and were able to use different mechanisms to interact with the same application feature, is a positive result. It supports one of the main functions of the toolkit: to abstract input from various sources. In addition, participants did not express any difficulty understanding the applications' interaction features implemented with the various widgets. This is also a positive aspect that supports the claim that the current widgets are generally understood and usable. In this regard, the mental model that users made of the system seems to have been enough to interact with the public display.

The on-screen/off-screen application model was a problem for some users because it introduced a discrepancy between the system's model and the mental model that those users created. The system's model is that even if an application is off-screen, users can still send input to it, but 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. This is something that cannot be totally resolved by an interaction toolkit alone. This behaviour depends on the specific application scheduler, and possible scheduling restrictions imposed by the display owner. In our deployment, applications followed a sequential time-based scheduling, which did not allow for the behaviour that some participants expected. Supporting asynchronous interactions imposes fewer restrictions on the application and interaction models, allowing users to send input to an application without having to wait for it to be on-screen. Clearly, however, this behaviour requires additional capabilities on the display system in order to provide users with enough information about the behaviour of the display. For example, telling users how long it will take before the application is put on-screen, would help users form a better mental model of the display's behaviour. Currently, we have modified the generated web GUI to inform the user when the targeted application is off-screen and that it may take a while for it to react to the user's input. However, this is a specific solution for the mobile interface. A more general solution would be to display additional feedback on the public display itself, but this would require a different application scheduler, capable of showing application notifications.

Another issue mentioned by participants in the real-world deployment was the fact that they had trouble knowing the various display system addresses for interaction, even though these addresses were shown periodically on the display, and were on the paper flyers distributed on the bar. This suggests that system addresses should be always be available and visible near or on the public display. We have since created a sticker application and widget that can be used to display this information on the display, while taking up only a small screen space. Another alternative is to provide a mobile application that supports the various phases of interaction with a public display, including discovering its address [12].

5.3 Functionality and Flexibility

Developing the applications for the real-world deployment gave us considerable insight on the functionality that the toolkit should provide. We used the requirements from these applications to improve the toolkit and make it more flexible to accommodate the necessities of various types of public display applications. Additionally, the programming study and the real-world deployment allowed us to better understand the current limitations of the toolkit. Next, we highlight some of the functionality and flexibility of the toolkit and some of its current limitations.

5.3.1 Application parameters

While developing the Public YouTube Player application, it became obvious that there were application instance parameters that display owners should be able to configure. These parameters are automatically stored in the server of the application and can be edited through a web interface. Programmatically, parameters are accessed in a similar way to Javascript's local storage, as key-value pairs. Although this is not directly related to the interaction or specific to public display applications, it makes sense to support application parameters in PuReWidgets because they are related to a specific instance of an application. This alleviates programmers from explicitly having to deal with different places and application instances. Additionally, application parameters can be overridden by URL parameters, providing more flexibility in the way that applications can be configured.

5.3.2 QR Codes

In the real-world deployment we used the QR code generation interface for display owners to create flyers for some of the polls of the EveryBody Votes application. This interface is generated automatically for display owners without the need for any intervention from the application programmer. This proved to be a valuable functionality as it allows display owners to draw attention to any interaction feature by using the generated QR codes in custom flyers that can be distributed on the place.

5.3.3 Custom reference codes

Although one participant suggested that reference codes could be made simpler to memorize by using words from application domain, this feature was already implemented in the toolkit during the real-world deployment, but was not used by any of the applications. Applications can suggest their own reference codes to be used in each widget, instead of the randomly generated ones. If there is no conflict with other already in use reference codes, the IM will honour the application's request.

5.3.4 Rapidly changing widgets

The deployment of the Wrod Game application showed one limitation of the current implementation of the web GUI: its inability to cope with rapidly changing widgets. 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 problem may be addressed with persistent connections, making sure the web interface is updated at a fast enough pace.

5.3.5 Delayed synchronization technique

The PuReWidgets library uses a delayed synchronization technique, where updates to a widget are only propagated to the IM a few seconds after the last change to the widget. In this technique, changing a property of a widget marks that widget as "dirty" and resets a delay timer. When the timer expires, all dirty

widgets are sent to the IM. This allowed us to remove the need to explicitly send a widget to the IM, simplifying the API and, at the same time, minimize the network and server overhead.

6. CONCLUSION

We have presented a widget toolkit for the development of web-based interactive public display applications. 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.

We have evaluated the toolkit along various dimensions – system performance, API usability, real-world deployment – that allowed us to cover the various aspects of the system. The evaluation results are generally positive, and show that the toolkit reaches its goals. We realize that the abstractions embedded in the desktop computing model exist at multiple levels and are the result of many years of evolution in interface design. In this work, we do not aim to reach anywhere near the equivalent of that for public displays, but simply to provide a first step in that direction. Hopefully, this toolkit will inspire the development of other toolkits and libraries, with different aims and offering different interaction models, contributing to open up the development of interactive public display applications.

7. ACKNOWLEDGMENTS

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.

8. REFERENCES

[1] Alt, F. et al. 2011. Digifieds: Evaluating Suitable Interaction Techniques for Shared Public Notice Areas. *Adjunct Proceedings of the 9th International Conference on Pervasive Computing* (2011).

[2] Cardoso, J.C.S. and José, R. 2012. PuReWidgets: a programming toolkit for interactive public display applications. *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '12* (New York, NY, USA, Denmark, Jun. 2012), 51.

[3] Cheverst, K. et al. 2005. Exploring bluetooth based mobile phone interaction with the hermes photo display. *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services - MobileHCI '05* (New York, New York, USA, 2005), 47.

[4] Clinch, S. et al. 2012. Designing application stores for public display networks. *Proceedings of the 2012 International Symposium on Pervasive Displays - PerDis '12* (New York, New York, USA, Jun. 2012), 1–6.

[5] Davies, N. et al. 2012. Open Display Networks: A Communications Medium for the 21st Century. *Computer*. 45, 5 (May. 2012), 58–64.

[6] Davies, N. et al. 2009. Using bluetooth device names to support interaction in smart environments. *Proceedings of the 7th international conference on Mobile systems, applications, and services - Mobisys '09* (New York, New York, USA, 2009), 151–164.

[7] Dearman, D. and Truong, K.N. 2009. BlueTone: a framework for interacting with public displays using dual-tone multi-frequency through bluetooth. *Proceedings of the 11th international conference on Ubiquitous computing - Ubicomp '09* (New York, New York, USA, 2009), 97–100.

[8] Dragicevic, P. and Fekete, J. 2002. ICON: input device selection and interaction configuration. *Companion proceedings of the 15th ACM symposium on User Interface Software & Technology (UIST'02), Paris, France* (2002), 27–30.

[9] Erbad, A. et al. 2008. MAGIC Broker: A Middleware Toolkit for Interactive Public Displays. *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)* (Mar. 2008), 509–514.

[10] Hardy, J. and Alexander, J. 2012. Toolkit support for interactive projected displays. *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia - MUM '12* (New York, NY, USA, Dec. 2012), 1.

[11] José, R. et al. 2008. Instant Places: Using Bluetooth for Situated Interaction in Public Displays. *IEEE Pervasive Computing*. 7, 4 (Oct. 2008), 52–57.

[12] José, R. et al. 2013. Mobile applications for open display networks: common design considerations. *Proceedings of the 2nd ACM International Symposium on Pervasive Displays -- PerDis '13* (Jun. 2013), 97–102.

[13] Kubitza, T. et al. 2013. Using mobile devices to personalize pervasive displays. *ACM SIGMOBILE Mobile Computing and Communications Review*. 16, 4 (Feb. 2013), 26.

[14] LocaModa App Store: 2010. *http://locamoda.com/apps/*.

[15] Marquardt, N. et al. 2011. The Proximity Toolkit: Prototyping Proxemic Interactions in Ubiquitous Computing Ecologies. *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11* (New York, New York, USA, Oct. 2011), 315.

[16] Paek, T. et al. 2004. Toward universal mobile interaction for shared displays. *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work* (New York, NY, USA, 2004), 266–269.

[17] Rohs, M. 2005. Visual Code Widgets for Marker-Based Interaction. *25th IEEE International Conference on Distributed Computing Systems Workshops* (Washington, DC, USA, 2005), 506–513.

[18] Swick, R.R. and Ackerman, M.S. 1988. The X Toolkit: More Bricks for Building User Interfaces, or Widgets for Hire. *Proceedings of the Usenix Winter 1988 Conference* (1988), 221–228.

[19] Taivan, C. et al. 2012. Selection and Control of Applications in Pervasive Displays. *6th International Conference on Ubiquitous Computing & Ambient Intelligence* (Victoria-Gasteiz, Spain, 2012).

[20] Terrenghi, L. et al. 2009. A taxonomy for and analysis of multi-person-display ecosystems. *Personal and Ubiquitous Computing*. 13, 8 (Jun. 2009), 583–598.

[21] Vogel, D. and Balakrishnan, R. 2004. Interactive Public Ambient Displays: Transitioning from Implicit to Explicit, Public to Personal, Interaction with Multiple Users. *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04* (New York, New York, USA, 2004), 137–146.