PuReWidgets: A Programming Toolkit for Interactive Public Display Applications

Jorge C. S. Cardoso
CITAR – Portuguese Catholic University,
Rua Diogo Botelho 1327, Porto, Portugal
jorgecardoso@ieee.org

Rui José
Algoritmi – University of Minho,
Campus de Azurém, 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 a study on interaction widgets for generalized interaction with public displays. We present PuReWidgets, a toolkit that supports multiple interaction mechanisms, automatically generated graphical interfaces, asynchronous events and concurrent interaction. This is an early effort towards the creation of a programming toolkit that developers can incorporate into their public display applications to support the interaction process across multiple display systems without considering the specifics of what interaction modality will be used on each particular display.

Author Keywords

Human-Computer interfaces; User Interface Design; Programming toolkits; Public displays

ACM Classification Keywords

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

INTRODUCTION

Public digital displays have become increasingly ubiquitous artefacts in public and semi-public spaces. Most of them, however, do not support any interactive features, even though interaction is clearly recognised as a key element in making them more engaging and valuable. A key reason behind this apparent paradox is the lack of efficient and clear abstractions for incorporating interactivity into public display applications. While interaction can 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 is an effort that must be replicated by each developer, representing a wasted effort. This also leads to inconsistent interaction models across different displays and, as a result, people are not able to develop, based on previous experiences, any expectations and practices regarding their interaction 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 was quickly recognised as a problem and 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 [12]. Nowadays, with the wide availability of interaction widgets, developers can benefit from ready-to-use interaction elements that deal with input, encapsulating behaviour and visual appearance, and users have learned to interpret their affordances in a way that enables them more easily to tackle new interfaces and programs by building on their previous experience.

In this work, we studied new interaction abstractions for the development of interactive applications for public displays. Our early results are instantiated in a programming toolkit that developers can incorporate into their public display applications. The main contributions of this work are the elicitation of the requirements for public display interaction abstractions and an architecture and software library system for application developers that provides high-level abstraction that can be incorporated into interactive public display applications.

This paper is organized as follows. We first characterize the interaction environment of public display applications; then we describe the main steps that we took while developing this work; we present work related to our own; then we define the requirements that an interaction abstraction should meet; we present the design of PuReWidgets; we provide an initial evaluation of the toolkit; and, finally, we conclude.

Interactive Public Display Applications

Applications for interactive displays are still an emerging topic with a lack of widely accepted and well-defined concepts. In this section we characterise our assumptions regarding the properties of the ecosystem of interactive public display applications.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EICS'12, June 25–26, 2012, Copenhagen, Denmark.
Copyright 2012 ACM 978-1-4503-1168-7/12/06...$10.00.

Display ecosystem

Using the concept of "ecosystem of displays" introduced by Terrenghi et al. [22], we could generally describe the public display environment as perch/chain[1] sized ecosystems for many-many interaction: and environment composed of displays of various sizes (from handheld devices, to medium/large wall mounted displays), and where "many people can interact with the same public screens simultaneously". The different sized displays afford different types of interaction but they can function in an integrated way in this ecosystem. The bigger displays (perch/yard sized) can function as the main information outlets of a place, providing a shared information and interaction point for the whole place – they are public, visible to everybody at all times (usually located in high-visibility locations), and can function as the reference display in a place. Normally, these displays are not meant to be appropriated by single users; they should be perceived as always available to everybody [9]. Medium-size displays (yard/foot sized) can also be present and dedicated to particular uses such as for allowing users to interact with the information on the main display, or for presenting some particular kind of information. These smaller public displays can be used, for example, to provide an interaction point (e.g., using touch interaction) that shows some of the most important interactive features that are locally available. The person that is responsible by the place – to whom we will refer to as the place owner – typically owns both large and medium displays. Small displays (foot/inch sized) are typically the personal mobile devices such as smart-phones, tablets, or laptop computers. Users that own these kinds of devices will want to take advantage of them to interact with their environment, including the available public display applications. The small personal devices will most likely be used as input devices to the public display application, allowing users to interact in a more opportunistic way by sending an SMS message to the application, using Bluetooth naming, or even a custom mobile application to interact. Personal tablets and laptops will most likely be used for more lengthy interactions allowing users to interact with an application via a place web page or directly through the application's web page, perhaps for configuring a user profile, or for upload or downloading large content files.

Applications

We want this display environment to be open to place owners, to application providers (or more generally, content providers), and to users. We have targeted our toolkit at web-based public display applications that can be hosted on third-party servers to serve content to many displays, but take advantage of the locally available interaction resources. Software developers will create these applications and will want to be able to distribute them globally. Place owners will be able to browse, select, and configure the applications they want to display in a given location. An application selected for a place will be sometimes visible on a public display. We assume that each display will show content from multiple applications and will iterate through those applications based on some pre-defined scheduling criteria. Even though an application may not be continually visible on the public display, it will be accessible via many other displays and interaction mechanisms. Once selected for a place, the public display application will be able to receive and process interaction events and produce place specific content that can be accessed in different ways (on a public display, through a web page, through a custom mobile application, etc.).

RELATED WORK

Interactive public displays are not new, and there are many systems that explore different interaction mechanisms that can be used by applications for public displays. For example, Rohs [4] 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 [4] developed Bluetone: a widget that is activated through dual tone multi-frequency (DTMF) over Bluetooth. Users interact with an application by changing the Bluetooth 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 Bluetooth 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. SMS interaction has also been used frequently with public display applications. Jumbli [11], 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 (BT) naming is another approach for providing interactivity to public displays. Lancaster University's e-Campus display system [21], for example, 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 channels to public displays. However they do not address the question of providing useful interaction abstractions to applications so, they don't help the application developer who wishes to deploy a public display application without worrying about the specificities of the

[1] 1 chain ≈ 20 meters; 1 perch ≈ 5 meters; 1 yard ≈ .9 meters; 1 foot ≈ .30 meters; 1 inch ≈ 0.025 meters (or ≈ 2.5 cm)

available interaction mechanisms of the various places where his application may run. In all the previous examples, the assumption was that a specific mechanism would be available.

There has also been much work on input middleware for ubiquitous systems. Magic Broker [7], 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, or how the web interface would be generated. Other input middleware such as ICON [6], allow the dynamic mapping of input devices to applications. However, these mappings are created for individual applications, and they work for local input devices. Also, it does not defined high-level controls suitable for public display applications.

Various interaction abstraction models have been used for different purposes and computing platforms. In the WIMP widget based interaction abstraction for GUI, widgets provide a high-level interface to the application in the form of widget events, triggered by user actions, which invoke callback functions in the application. The application does not know the specific action that was used to trigger the event; it has only access to the high-level data exposed by that specific widget. In the dynamic user interface generation, more appropriate for smart environments, programmers describe the application/service interface using an abstract language, which is then used to generate various interfaces for different devices (e.g. widgets for graphical devices; speech interfaces; etc.). Communication between the device and application is usually accomplished via some form of remote method invocation. The abstract language usually allows developers to specify which functions and parameters are associated with a particular interactive feature. There is also the data-driven interaction approach, usually used in cases where we want a single application to be able to receive input from various, different, "dumb" input devices. This approach is usually implemented using a tuple space data structure where input devices and application programmers define their own tuples and a mapping software component maps tuples from input devices to tuples for applications. Thus, programmers are free to define whatever tuples they need and applications simply react to the data-type (and parameters) of the incoming tuples.

SCENARIOS

To provide a better image of the type of interaction we envision of public display applications, we describe next some usage scenarios.

John is a software developer in charge of creating an interactive public display application that will integrate with an existing social news platform developed by the same company. The existing platform allows an institution to post news items on a web page and allows users to "like" and discuss on those individual items. There are already two clients that want to use this new public display application: a university's communication department and a local coffee shop. John has already developed much of the logic for the application and is now on the process of adding the "like" feature. He fires up his favourite IDE – Eclipse, and opens up the application project. In the project settings he configures the application to use an interaction library for public displays. The application creates and displays a list of text items and for each item John needs to associate a "like" action by instantiating an action widget which, when activated, will contact the server to update news platform with the indication that a user liked the news item. He does not need to worry about the specific input mechanisms that will be used to "press" the action button; the interaction library handles all that...

Sophia is waiting for her friends at the university's main hall. Looking at the large display across the hall, one of the entries of the school-related news catches her eye - it's about Adam, a friend on the robotics class, which has won the national robot-dancing contest. There is a button next to the news entry's header that Sophia recognizes: is a ``like'' button with three letters underneath. The instructions on the top of the display tell her how to interact so she fetches her mobile phone and sends a text message to the number on the instructions. A few seconds later, a popup near the button appears with a phone number. Some digits do not show, but she recognizes it as her own. She knows her ``like'' will increase the news visibility on the school's website and on the display. Adam deserves it!

Sarah and George took a break from work to grab a snack at the coffe shop across the street. They sit down and order an entry from the menu that is on their table – the latte+muffin menu. While they're eating and talking, Sarah notices a familiar symbol next to each entry in the menu: a QR code. The description says that they can post a comment. George is not sure how that works, but he pulls his smartphone, launches the default app for visual codes, and scans the code. A webpage opens with a textbox. He enters: "Best blueberry muffin, ever!" and presses Send. A confirmation message pops up thanking and telling him that he can check the result of his interaction in a nearby display. A few moments later they notice that the display in the coffee shop is showing photos of the various menu entries and comments from customers: George's comment appears next to the latte+muffin entry!

RESEARCH METHODOLOGY

This work proceeded in three phases. In the first phase, we elicited the main requirements for interaction abstractions for public displays. For this, we collected academic publications about interactive public display systems by searching online databases (such as ACM, IEEE, Google Scholar) and filtering publications by keywords such as "public display", "interactive display", from the last 20 years. In our analysis dataset, we also included references from these publications to other public display systems (in total, we analyzed about

50 different display systems). We focused on the descriptions of the requirements, functionalities, and properties of the described display systems to extract relevant common features and synthesize them in a set of high-level requirements (cf. Requirements for Public Display Interaction section).

In a second phase, we investigated existing ways of providing interaction abstractions to application programmers, taking note of their main properties and paying particular attention to how they could support our requirements. We analyzed specifically the widget abstraction model, the dynamic interface generation model and the data-driven interaction model (cf. Related Work section). This phase resulted in a set of design guidelines that incorporated features from the various existing interaction abstractions to form a new interaction abstraction for interactive public displays.

In a third phase, we re-analyzed the interactive public display systems of phase 1, but this time focusing on analyzing the types of high-level data generated by different types of interactions with public displays [2], and then examining various interactive features proposed in different display system to extract the fundamental properties of those features. This resulted in a set of control types that serve as the basis for the various controls in our toolkit (cf. PuReWidgets System section). While designing the toolkit we made a decision to support control types that would not impose a direct manipulation interaction style.

REQUIREMENTS FOR PUBLIC DISPLAY INTERACTION

The main objective of an interaction abstraction is to facilitate the programmer's task of developing an interactive public display application by abstracting away the details of the multiple interaction mechanisms that may exist in a place, and which may vary across places. At the same time, the abstraction should allow developers to specify what kind of high-level interaction data their applications need. Achieving this objective for public display applications entails addressing several requirements. Some of these requirements are common to other interactive systems, but others are very specific to public displays.

Multiple interaction mechanisms

Unlike desktop systems, which usually rely on a very small set of input devices – most often just a keyboard and mouse – public display interaction can take advantage of several, very different input mechanisms. Many public displays have been developed that use very different input mechanisms, such as SMS [24], email and instant messaging [16], Bluetooth naming [10], Twitter [11], RFID [13], body movement [18], gestures [23], face detection [8], custom mobile applications [19], etc. These different input mechanisms have different costs and requirements and a single place cannot be expected to provide all of them nor can we expect to encounter the same set of input mechanisms in all places. Additionally, not all input mechanisms have the same data capabilities so they may not all be capable of providing the same high-level input controls. Application programmers, however, should be able to specify their interaction necessities in a way that is independent from the specific interactive modalities or input mechanism that will be available at each specific place. A good interaction abstraction should be applicable, in a consistent way, to multiple input mechanisms.

Concurrent interaction

Given the many-many nature of the social interaction with public displays, public displays must explicitly support multiple, concurrently interacting users, possibly using different input mechanisms. Many applications will, at least, require information about the input events that allows them to differentiate users. An interaction abstraction for public displays should give support for concurrent input by multiple users, possibly using different input mechanisms. This is in sharp contrast to desktop systems were the assumption is that, generally, a single user is interacting – in control of the keyboard and the mouse – and applications are indifferent to which user is interacting. This has implications in the public display system support for the interaction because it means that there is a need to differentiate input events for different users.

Shared interaction

Shared interaction works on two levels: the first means that users are aware of each others interactions and, so, may decide to adapt their own behaviour in light of what others are doing; the second means that the display system is able, not only to accept concurrent interaction, but also to conciliate those interactions in its response. In a many-many interaction setting, being aware of each others actions is fundamental to the success of the interaction because it can act on two important aspects of public display interaction: attention and motivation [14]. The first barrier to interaction with a public display is understanding that it is interactive – moving from an unwitting bystander to a witting bystander (as defined by Dix and Sas [5]). If the display system provides some kind of public awareness regarding interactions, it can help attracting users' attention and making users aware that the display is interactive. It can also add to the collaboration motivation factor for interacting with public display, because "collaboration is especially motivating if individual behaviour is recognized by others" [14].

Asynchronous interaction

Our assumptions regarding the life cycle of a public display application are very different from traditional desktop applications. The life-cycle of a desktop application is completely controlled by the desktop user, which decides when the application should run, when it should be in the foreground receiving input, and when it should be terminated. In a public display ecosystem, users may not, in general, control applications. Once an application is associated with a particular place, it should be available for interaction at all times, or at least have that possibility. Also, a public display application should generally be available for users independently of whether there any public display currently showing any of its content. Contrary to desktop

applications, the display is not the only interaction point with a public display application. A good interaction abstraction for public displays must support this kind of asynchronous interaction environment and allow interaction to happen at any time. This kind of interaction can help mitigate the "conflict of pace" mentioned by Dix and Sas [5], which happens because users are not in full control of the public display. An asynchronous interaction environment guarantees that, at least, the display's scheduling does not impose the pace for the interaction with an application.

Clear and decoupled affordances

The interaction abstraction should convey clear affordances in a way that people may easily learn to recognize, enabling potential users to become aware of the existence of the interactive features and their properties. Even when facing a display or an application for the first time, the interaction alternatives should always be clear, even if the semantics of the operation for an unknown application are not. This is a generic interaction guideline and a key function of an interaction abstraction, common to other interactive systems. It responds to the basic interface design principle of visibility, which helps bridging the gulf of evaluation of a system [15]. It is especially important for public displays because, unlike what happens with desktop computers where people are aware of the computer, when facing a public display, users may not even realize it is interactive. The interaction abstraction can partly address these issues and help users move from unwitting bystanders to participants, by providing identifiable graphical representations for widgets on the public display. However, given the environment in which public display applications will operate, we can't expect applications to be continuously shown on a public display nor to have the ideal screen space available to display an application's content. This requires that the affordances for the interactive features be decoupled from the public display screen, because it may not always be possible or desirable to show the graphical representations for the interactive features on the public display. The interaction abstraction should be flexible enough to allow the interactive features of applications to be rendered in other platforms such as web pages, or mobile devices. Ideally, this should be done with minimal or no extra effort needed from the application developer.

Multiple, public display specific, interactive features

A good interaction abstraction for public display must allow applications to have many different and individually addressable interactive features, just like standard desktop applications. Desktop applications typically need several interactive features of different types of controls. A single desktop form screen, for example, may require several text boxes, list boxes, radio buttons, and action buttons. The different types of controls allow programmers to choose the ones that best fit the application's data needs. There are many different controls for desktop applications such as data entry, selection, imperative, and display controls [3], and each type may have several variations that provide applications with different high-level data and give users different affordances. Public display applications also need a set of controls for developers to choose from, but these controls must be appropriate for public display interaction. An interaction abstraction should provide a set of useful control types that allow a wide range of meaningful interactions. Programmers should be able to specify any number of interactive features that the application needs, and users should be allowed to address those features individually.

PUREWIDGETS SYSTEM

The PuReWidgets system is composed of a widget library and web service that handles interaction events. A widget is an interaction abstraction that: provides developers with high-level interaction data, hides the specific details of the underlying input mechanism; and can have different graphical representations in different platforms. The development process of a public display application that uses PuReWidgets is similar to the development of a regular web application. The developer includes an external code library in his project and uses the available functions of the library to code the application, instantiating widgets and registering interaction event callback functions. The developer then deploys the set of HTML, CSS, and Javascript files on a web server. The life cycle of a public display application (start, stop, and reacting to input events), however, is very different from the life cycle of a traditional application: the application is instantiated and terminated by a scheduler software that drives all the content of the public display, and interaction events can be generated via multiple local or remote sensors. When a widget is instantiated by an application, some metadata about the widget are sent to the PuReWidgets service. A remote I/O infrastructure is responsible for accepting raw input events from users. This I/O infrastructure can serve multiple displays, or even places, and its function is mainly to provide an initial abstraction over several sensor data such as SMS, Bluetooth naming, OBEX, etc. These input events are then used by the PuReWidgets service, which routes them to the application/widget that was addressed by the user. This service acts as an input event queue, storing the widget input until the application is ready to receive them, allowing applications to receive widget events even if they were generated when the application was not executing at the public display. When the PuReWidgets library asks for input, the service replies with the stored input. The library (running within the application) then forwards the input to the correct widget instance so that it can trigger the high-level application event. This requires a distributed architecture in which some widget information is kept by remote services, effectively decoupling widgets from applications. PuReWidgets provides two application models depicted in Figure 1, and described next.

PuReWidgets Library

The library provides high-level interaction abstractions to applications (widgets), and it is actually composed of two separate libraries: one for server-side code, and one for client-side code. This allows programmers to develop

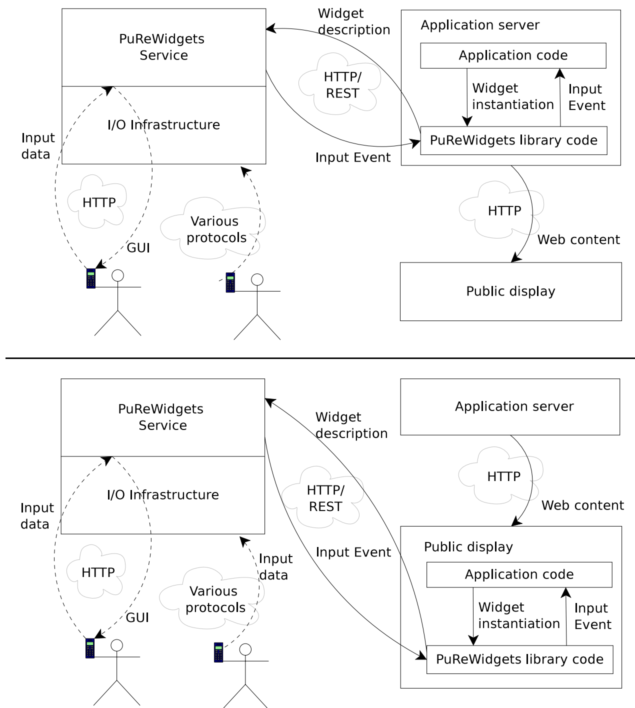


Figure 1: Application models for the PuReWidgets toolkit.

different types of applications, depending on the particular needs. The server-side library (Figure 1, top) allows developers to create applications than run mainly on the web server (i.e., their main logic resides on the server). These applications can run independently of the public display scheduling, e.g., they react immediately to user input, updating their internal state or calling external services, that may affect the content that it will display next, even if the application is not currently showing content on the display. The client-side library (Figure 1, bottom) allows developers to create applications that are more tightly coupled with the public display in the sense that their main, or even only, content output is on the display itself. These applications may not need to react immediately to interactions if they are not currently showing content on the public display so the widget life cycle, in this case, is coupled to the scheduling of the application in the public display. For these cases, it makes more sense for the main application logic to reside in the client code that is transferred to the public display so that it can create and control the necessary widgets. The PuReWidgets toolkit support this development mode by providing a client-side library (even though the code is still transferred from the application server in the form of Javascript, HTML, and CSS).

Control types

Widgets are provided in the form of an object-oriented library in which each widget has a type that defines the type of high-level data that it exposes to the application. Programmers can choose which widgets to use, according to the application's data needs (in some case there may be alternative widgets for the same data need), by instantiating the respective widget class and registering a callback to receive the high-level events generated by the widget instance. The toolkit also allows programmers to extend the existing widgets and provide new ones, more suited to some specific interactive features needed by a particular application. We have based our toolkit's controls on the analysis of different types of high-level information generated by interaction with public displays [2], and categorized them in five categories: imperative/selection, entry, download, upload, and check-in controls.

Imperative/Selection controls

Imperative/selection controls allow users to trigger actions or select options in the public display application. From the abstraction point of view, an imperative control can be viewed as a selection control with just one option. The high-level event generated by these controls just needs to identify the option that the user selected. Many concrete widgets such as different types of buttons, list boxes, and check boxes, are of this type. Currently, PuReWidgets provides a button and a listbox widget.

Entry controls

Entry controls allow users to input simple data such as free text or bounded values. These controls generate high-level events that contain the input data. In this category we can include widgets such as textboxes, but also bounded data widgets such as number boxes. We have currently implemented a textbox widget that accepts unbounded text.

Upload controls

An upload control allows users to submit media files to the public display application. The high-level event generated by these controls includes an URL to the uploaded file so that the application can then process it. Concrete widgets can be specialized in particular media types, providing high-level events only is the media type of the uploaded file matches the required one.

Download controls

Download controls allow the application to provide files that users can download to their personal devices, or forward to their email, etc. This type of control generates a high-level event that simply signals that a user wants to download the item. The process of actually sending the file to the user is handled transparently by the toolkit. When instantiating the widget, applications are required to specify the location (an URL) of the associated media file.

Check-in controls

Check-in controls allow users to signal the application that they are present. In this case, the high-level event is just the identification of the user that has just checked-in.

Decoupled widgets

Decoupled widgets are widgets that do not depend on the application that created it for graphical representation or interaction. A decoupled widget allows the public display system to provide alternative graphical representations and interaction points to a widget created by an application. PuReWidgets provides automatic generation of desktop, mobile, and QR code interfaces for all widgets. The desktop

and mobile interfaces are web-based and provide a rich graphical interface to an application's widgets (the interfaces are kept in synchronization with the widgets created by the application). The QR code generation can be used by place-owners who wish to draw attention to specific interactive features by printing the codes and distributing them locally. The codes can also be explicitly used by applications that wish to provide an alternative QR code based graphical interface on the public display itself.

This decoupling is accomplished by using a PuReWidgets service that stores metadata and input information about the instantiated widgets and exposes this information to system applications. All this is done transparently to the application and to the application developer. Whenever a widget is instantiated or updated by an application, the PuReWidgets toolkit sends the widget description data to the PuReWidgets service. The data that is sent to the server includes the widget unique id within the application, the type of control (imperative, entry, upload, download, check-in), a short and long textual description of the widget (used to give contextual information to the user), and a list of possible widget options (for widgets with several options). A widget option is composed of an option id, and short and long descriptions.

Public display applications are still responsible for creating and destroying widgets, during the course of their lifetime, allowing applications to behave much like desktop applications, which are responsible for graphically laying out their widgets and rendering them on the display, but it also allows the display infrastructure to keep track of the widgets that each application is using and providing alternative interaction points. It should be noted, however, that this does not preclude application developers from creating a custom web or mobile interface to their applications. Both can even be integrated in the display system, which can provide users with the custom application web or mobile interface, but fall back to the dynamically generated one if the former does not exist.

Addressing an input routing

PuReWidgets takes advantage of an I/O infrastructure that provides input data acquisition and basic level parsing to third-party components. This infrastructure manages a variety of sensors and input mechanisms and pre-processes the data input coming from these sensors. The I/O infrastructure works on two levels. On the lower level, the infrastructure is able to parse the raw input data and structure it into abstract "commands", using a pre-defined command syntax. As an example, the SMS, email or even Bluetooth

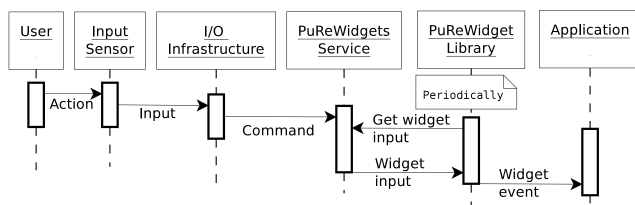


Figure 2: Sequence diagram of user sending input.

modules can be used to send keywords to a public display, which the I/O structures into a "keyword" command, with a parameter consisting of the actual keywords. A client of the I/O service is able to request a list of "keyword" commands issued and respective parameters (along with other metadata, such as timestamp, input mechanism id, etc.). The I/O infrastructure is also able to extract and store media files received via OBEX or through other mechanisms, and provide them on request to clients. On a higher level, the I/O infrastructure is able to associate individual input data with user identities. This optional service allows users to register and associate several personal input mechanisms (phone number, Bluetooth MAC address, etc.), which the infrastructure uses to identify which user is interacting. Depending on the available level, the PuReWidgets service is able to get a user id and associated nickname, or at least an input mechanism id (such as an anonymised phone number or Bluetooth MAC address) that allows it to differentiate among users.

PuReWidgets relies on this I/O service to support several low-level input mechanisms such as SMS, Bluetooth naming, etc. We use an I/O service developed for another project [10], but other I/O middleware such as the one by Paek et al. [16] could have been used. For these interactions, our approach to addressing is based on a simple referencing scheme that relies on unique textual reference codes that are generated for each widget instance and that become the address of the widget. These reference codes are small (3 or 4 alphanumeric characters), and are generated automatically by the PuReWidgets service. Widgets can have several distinct reference codes to allow addressing options within a single widget. These reference codes can be used explicitly by users on an SMS, Bluetooth naming, email, and other text-based mechanisms.

In some cases, routing behaves a little differently. For example, the check-in widget is naturally global to the place: users check-in to a place, not a specific application. In these cases, routing must also be global in the sense that all widgets of that particular type, regardless of in what application they were instantiated, will receive the input. This kind of routing is applied on an input mechanism basis or using place generated reference codes. For example, all input from a magnetic card reader may be interpreted as global data that should be sent to all check-in widgets. In these cases, routing the input data is a matter of associating the input with all applications that are currently using these types of controls.

When using the rich graphical interfaces or the QR codes for interactions, routing is more simple: the generated interfaces use the widget id and communicate directly with the PuReWidgets service to create input events directly associated with a widget instance from a particular application.

The input sequence from the time the user issues the input to the instant the application receives the input event is illustrated in Figure 2.

Graphical input feedback to users' actions

An important aspect of desktop widgets is the system-level feedback they provide and that helps users understand the response of the system, independently of how the application will react.

For public displays, feedback can also be used as a way to convey a sense of awareness about other users' actions. Displaying input feedback on the public display effectively helps creating a shared interaction environment independent of the application itself. This is an important aspect for creating more engaging public displays [1]. However, public display interaction also imposes practical considerations that may require other solutions for input feedback. In some cases, providing feedback through the main display itself may not be the best solution, in part because the available screen real-estate may dictate other priorities, but also because there are other feedback channels that can be more efficient considering the multi-user and multi-modality nature of the interaction.

Our approach is to provide a base mechanism for presenting feedback on the public display: the graphical representation of a widget includes the associated graphical input feedback. This is similar to what happens for the desktop, with the difference that, given the multi-user scenario, feedback information must be much more explicit for public displays, providing an indication of which user is responsible for the input. The feedback mechanism ensures that in a shared interaction environment, users are able to identify the feedback to their own input. Also, feedback can be decoupled from the graphical representation of the widget: programmers can choose to display feedback for a particular widget, even if the widget itself is not displayed on the public display.

Implementation

PuReWidgets was implemented using Google's App Engine platform (http://code.google.com/appengine) and Google's Web Toolkit (http://code.google.com/webtoolkit). The library is provided as a GWT module that developers can include in their GWT projects and the service is implemented as an App Engine application that exposes a REST API to the library. The graphical components of the widgets take advantage of the standard GWT widgets.

Current set of interaction mechanisms

PuReWidgets is designed in a way that allows the user of multiple interaction mechanisms. Currently, PuReWidgets supports the following interaction mechanisms: SMS, email, Bluetooth naming, Bluetooth OBEX, QR codes, mobile application, and desktop web application.

Using the toolkit

To show how PuReWidgets can be used to create a display application, we now describe a simple Hello World public display application. Using Google's GWT platform and PuReWidgets, the main application class would simply be the one in Listing 1. The code is very similar to what we would need if we were developing a desktop application.

```
 1 public class HelloWorld implements EntryPoint {
 2    @Override
 3    public void onModuleLoad() {
 4        PublicDisplayApplication.load(this, "HelloWorld", true);
 5        GuiButton guiButton = new GuiButton("helloButton",
 6                                            "Hello World");
 7        guiButton.setShortDescription("Say hello!");
 8        guiButton.setLongDescription("Say hello to be greeted
 9                              by the HelloWorld application");
10        guiButton.addActionListener(new ActionListener() {
11            @Override
12            public void onAction(ActionEvent<?> e) {
13                PopupPanel popup = new PopupPanel();
14                popup.add( new Label("Hello " +
15                           e.getPersona() + "!") );
16                popup.show();
17            }
18        });
19        RootPanel.get("main").add(guiButton);
20    }
21 }
```

Listing 1: Hello World application main class.

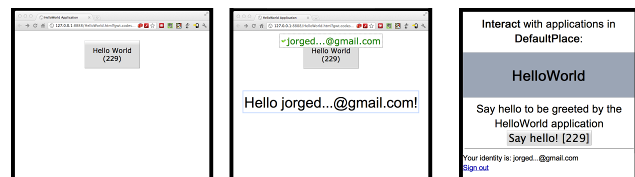
The main differences are in line 4, which initializes some background data structures and processes to communicate with the PuReWidgets service. Line 5, which creates a button with an application defined-named 'helloButton'. This name is needed so that the PuReWidgets service is able to distinguish widgets and to make sure that, if a widget was already created, it is not recreated. Lines 7 and 8 are needed to provide some application-specific context information in case the widgets are used in other platforms (see Figure 3-c for the mobile interface). Figure 3 shows the output of the Hello World application: a) the regular output; b) the reaction to a user input from the mobile interface; and c) the automatically generated interface for a mobile device. The popup on top of the button is the input feedback provided by PuReWidgets (which can be disabled by applications).

EVALUATION

Evaluating a programming toolkit like PuReWidgets is challenging, mainly because interactive public display applications are a new thing, and there are no programming communities for this platform. Given the current state of this field, our best approach to begin evaluating the toolkit was to develop some applications that could be deployed as real public display applications, and try to assess, through hands-on experience, whether the main requirements are met. We have implemented two interactive public display applications: a public video player, and voting application.

Public video player application

The public video player is an application that searches for, and plays youtube videos. Search is based on tags taken from a tag cloud that is built using tags defined by the place owner, suggested by users, and extracted from videos that users liked. The application is composed of three screens (Figure 4) which iterate over time: (left) a screen for playing the current video in full screen, (center) a screen that shows the recent activity (played videos, liked videos, and



a) Application output b) Reacting to input c) Mobile interface

Figure 3: Hello World application.

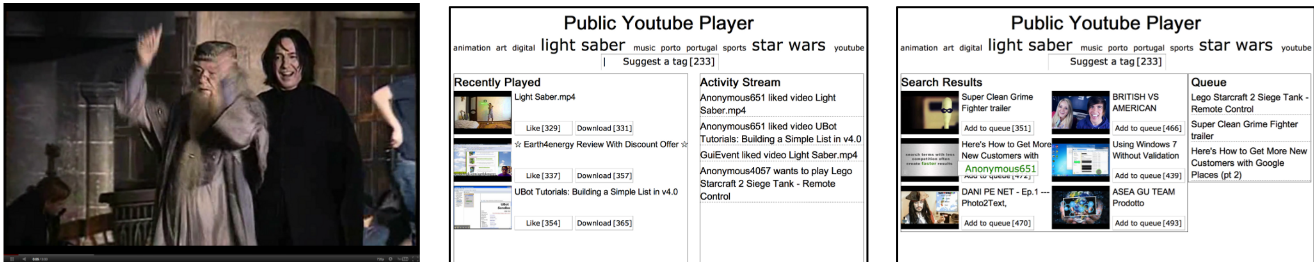


Figure 4: Public video player and voting applications.

suggested tags), and (right) a screen which shows alternative videos to play next along with a video play queue; the last two screens also display the current tag cloud. The application's interactive features are: 1) allow users to suggest tags. This was implemented using a custom tag cloud widget that incorporates a textbox widget. The tag cloud widget accepts keywords and automatically creates a tag visualisation. 2) Allow users to "like" videos. This feature was implemented with an action button that is displayed on the activity screen and allows users to "like" a specific video. 3) Allow users to download a reference to a recently played video. This feature was implemented with a download widget by providing a link to the corresponding youtube page. 4) Allow users to select a video to play from the list of search results. Action buttons are displayed to allow this. Selected videos are put in a play queue.

Voting application

The voting application is composed of two screens, depicted in Figure 5: (left) an open polls screen which iterates through the open polls, showing their description and options, and (right) a closed polls screen which iterates through the closed polls and shows their voting results. Polls are created by the place owner in a backoffice interface. The application offers the following interactive features to users: 1) Vote on a specific poll. The options of a poll are presented using a poll widget, which was built on top of a listbox widget but additionally shows a graphical representation of the votes when someone interacts. 2) Suggest a poll. A textbox widget is displayed briefly after someone interacts and on the closed polls screen, to signal that users can also suggest questions for polls.

Analysis

Developing these two applications enabled us to observe some important properties of PuReWidgets. The transparent support for multiple mechanisms, for example, enabled the place owner to create QR codes for some of the long running polls and to place them in wall posters or flyers drawing

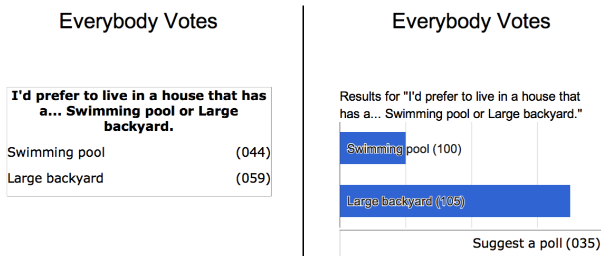


Figure 5: Voting application.

more attention to those polls. This was done transparently to the application; while developing it we paid no specific attention to this possible use.

The identification of users/interaction mechanisms was also an important aspect of the interaction abstraction. Without it the poll application would not be possible. In this application, we used this identification to determine if a user had already voted on a specific poll, thus allowing a more correct voting count (there is still the problem of a single user voting using different input mechanisms, in which case multiple votes will be counted). This feature was also used in the youtube application, allowing us to create a play queue when multiple users selected a video to play next.

Support for asynchronous interaction was also an important feature, specifically for the voting application. This application is only shown on the display for a brief period at a time, but because the toolkit supports asynchronous interaction, users can still be aware of this application through the printed QR codes, for example, and vote on the existing polls.

While developing the youtube application we also demonstrated the flexibility of the widget classes, namely the possibility of creating new widgets by composing existing ones. We composed the tag cloud widget by incorporating the existing textbox widget into a new widget that automatically keeps a list of tags and tag frequencies and displays a tag cloud visualization. From the point of view of the application, this is a widget just like any other.

CONCLUSION

We have created a toolkit for developing interactive public display applications, which handles much of the work a developer would have to deal with to develop even the simplest interactive public display application. PuReWidgets provides high-level interaction abstractions that suit the kind of interaction one normally does with public display applications and transparently supports various interaction mechanisms. The toolkit provides a widget addressing and an input routing mechanism, supports concurrent, asynchronous interaction and provides decoupled graphical affordances that can be used directly on the public display, or on alternative platforms. This toolkit fills a clear gap in the area of interactive public displays. Having a foundational tool like PuReWidgets allows designers and programmers to focus on the real creative work of designing interesting applications and user experiences.

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. This research has also received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 244011 (PD-Net).

REFERENCES

1. Brignull, H., & Rogers, Y. (2003). Enticing People to Interact with Large Public Displays in Public Spaces. In M. Rauterberg, M. Menozzi, & J. Wesson (Eds.), *INTERACT'03* (pp. 17-24). IOS Press.
2. Cardoso, J. C. S., & Jose, R. (2009). A Framework for Context-Aware Adaptation in Public Displays. In R. Meersman, P. Herrero, & T. Dillon (Eds.), On the Move to Meaningful Internet Systems: OTM 2009 Workshops (Vol. 5872/2009, pp. 118-127). Vilamoura, Portugal: Springer Berlin / Heidelberg. doi:10.1007/978-3-642-05290-3_21
3. Cooper, A., Reimann, R., & Cronin, D. (2007). About face 3: the essentials of interaction design. New York, NY, USA: John Wiley & Sons, Inc.
4. Dearman, D., & Truong, K. N. (2009). BlueTone. Proceedings of the 11th international conference on Ubiquitous computing - Ubicomp '09 (p. 97). New York, NY, USA: ACM Press.
5. Dix, A., & Sas, C. (2008). Public displays and private devices: A design space analysis. Workshop on Designing and evaluating mobile phone-based interaction with public displays. CHI2008. Florence.
6. Dragicevic, P., & Fekete, J.-D. (2001). Input Device Selection and Interaction Configuration with ICON. In P. G. A. Blanford, J. Vanderdonkt (Ed.), People and Computers XV Interaction without Frontiers: Joint proceedings of IHM 2001 and HCI 2001 (IHM-HCI '01) (pp. 543-558). Springer Verlag.
7. Erbad, A., Blackstock, M., Friday, A., Lea, R., & Al-Muhtadi, J. (2008). MAGIC Broker: A Middleware Toolkit for Interactive Public Displays. 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom) (pp. 509-514). IEEE.
8. Grasso, A., Muehlenbrock, M., Roulland, F., & Snowdon, D. (2003). Supporting communities of practice with large screen displays. In K. O'Hara, E. Perry, E. Churchill, & D. M. Russel (Eds.), Public and Situated Displays - Social and Interactional Aspects of Shared Display Technologies (pp. 261-282). Kluwer.
9. Huang, E.M., Mynatt, E.D., and Trimble, J.P. When design just isn't enough: the unanticipated challenges of the real world for large collaborative displays. Personal Ubiquitous Comput. 11, 7 (2007), 537-547.
10. José, R., Otero, N., Izadi, S., & Harper, R. (2008). Instant Places: Using Bluetooth for Situated Interaction in Public Displays. IEEE Pervasive Computing, 7(4), 52-57.
11. LocaModa. (2010). LocaModa App Store. Retrieved March 2011, from http://locamoda.com/apps/
12. McCormack, J., & Asente, P. (1988). An overview of the X toolkit. Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software - UIST '88 (pp. 46-55). NY, NY, USA: ACM Press.
13. McDonald, D. W., McCarthy, J. F., Soroczak, S., Nguyen, D. H., & Rashid, A. M. (2008). Proactive displays. ACM Transactions on Computer-Human Interaction, 14(4), 1-31. New York, NY, USA: ACM.
14. Müller, J., Alt, F., Michelis, D., & Schmidt, A. (2010). Requirements and design space for interactive public displays. Proceedings of the international conference on Multimedia – MM'10 (pp. 1285-1294). New York, NY, USA: ACM Press.
15. Norman, D. A. (2002). The Design of Everyday Things. Basic Books.
16. Paek, T., Agrawala, M., Basu, S., Drucker, S., Kristjansson, T., Logan, R., Toyama, K., et al. (2004). Toward universal mobile interaction for shared displays. Proceedings of the 2004 ACM conference on Computer supported cooperative work (pp. 266-269). New York, NY, USA: ACM.
17. Rohs, M. (2005). Visual Code Widgets for Marker-Based Interaction. 25th IEEE International Conference on Distributed Computing Systems Workshops (pp. 506-513). Washington, DC, USA: IEEE.
18. Sawhney, N., Wheeler, S., & Schmandt, C. (2001). Aware Community Portals: Shared Information Appliances for Transitional Spaces. Personal and Ubiquitous Computing, 5(1), 66-70. London, UK: Springer-Verlag.
19. Scheible, J., & Ojala, T. (2005). MobiLenin combining a multi-track music video, personal mobile phones and a public display into multi-user interactive entertainment. Proceedings of the 13th annual ACM international conference on Multimedia (p. 199). New York, NY, USA: ACM Press.
20. Shneiderman, B., & Plaisant, C. (2005). Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th ed.). Addison Wesley.
21. Storz, O., Friday, A., & Davies, N. (2006). Supporting content scheduling on situated public displays. Computers & Graphics, 30(5), 681-691.
22. Terrenghi, L., Quigley, A., & Dix, A. (2009). A taxonomy for and analysis of multi-person-display ecosystems. Personal and Ubiquitous Computing, 13(8), 583-598.
23. Vogel, D., & Balakrishnan, R. (2004). Interactive public ambient displays. Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04 (p. 137). New York, NY, USA: ACM Press.
24. Vogl, S. (2002). Coordination of Users and Services via Wall Interfaces. (PhD Thesis). University of Linz, Linz, Austria.