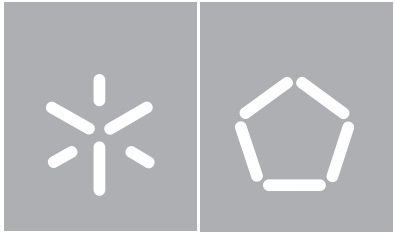
Universidade do Minho
Escola de Engenharia

Jorge Carlos dos Santos Cardoso

An Interaction Abstraction Toolkit for Public Display Applications

An Interaction Abstraction Toolkit for Public Display Applications
Jorge Carlos dos Santos Cardoso
UMinho | 2013

Setembro de 2013



Universidade do Minho
Escola de Engenharia

Jorge Carlos dos Santos Cardoso

An Interaction Abstraction Toolkit for Public Display Applications

Tese de Doutoramento
Tecnologias e Sistemas de Informação

Trabalho efectuado sob a orientação de
Professor Doutor Rui João Peixoto José

Setembro 2013

Declaração

Nome: Jorge Carlos dos Santos Cardoso

Telefone: +351 226196200

Endereço electrónico: jorgecardoso@ieee.org

Número do Bilhete de Identidade: 11730214

Título da tese: An Interaction Abstraction Toolkit for Public Display Applications

Orientador: Professor Doutor Rui João Peixoto José

Ano de conclusão: 2013

Ramo de Conhecimento do Doutoramento: Tecnologias e Sistemas de Informação

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, Setembro 2013

Assinatura: Jorge Carlos dos Santos Cardoso

Acknowledgements

Taking a PhD degree is something that is usually described as a lonely activity. All things considered, although it might feel like a lonely job, it is in fact a journey where the contributions of many people come together. Many people have contributed to this work, to whom I would like to thank.

First, I must acknowledge and thank my supervisor, Professor Rui José. His effort and dedication in guiding, correcting and supporting me during this period makes the present work as much his as it is mine.

I must also acknowledge all the colleagues of the Mobile and Ubiquitous Systems (Ubicomp) with whom I have shared very good moments. I would especially like to thank Bruno, Helder, and Constantin for their support and contribution to my work.

The team of the PD-NET project also deserves a special mention. I have learned a lot from my collaboration with them.

I would also like to thank my colleagues at the Research Centre for Science and Technology in Art (CITAR) and at the School of Arts of the Portuguese Catholic University for providing an excellent work environment, and for supporting and participating in various of the studies in this work.

Finally, and most importantly, Sara for putting up with me during all this time.

This research was supported by the Fundação para a Ciência e Tecnologia (FCT) PhD 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).

Abstract

An Interaction Abstraction Toolkit for Public Display Applications

Public digital displays have become increasingly ubiquitous in our technological landscape. Considering their flexibility and communication potential, public displays can become an important communication channel and even reach the attention, usage, and relevance that smartphones have today. Interaction with public displays is recognised as a key element in making them more engaging and valuable, but most public display systems still do not support any interactive feature. 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.

In this work, we investigate and develop interaction abstractions for public displays. We start by analysing public displays from the point of view of the information that results from the various interactions and that can be used to drive several types of content adaptation behaviour on public displays. We call this information *digital footprints*, and the result is a framework that maps digital footprints to adaptation strategies and to interaction mechanisms. This framework can be used by display designers to help them choose the interaction mechanisms that a display should support in order to be able to collect a given set of footprints, creating more relevant displays that are able to automatically adapt to their environment. We then identify and characterise interaction tasks and controls that are appropriate for public display interaction. This analysis results in a design space that can form the foundation of interaction toolkits, giving system developers with a reference for the types of high-level tasks and controls that can be incorporated into a toolkit. Finally, we design, implement, and evaluate a software toolkit of interaction abstractions for public display applications – the PuReWidgets toolkit. Programmers can use this toolkit to easily incorporate interactive features into their web-based public display applications. PuReWidgets provides high-level abstractions that shield programmers from the low-level details of the interaction mechanisms. We evaluate this toolkit along various dimensions. First, we evaluate the system's performance. We then evaluate the API's flexibility and capabilities using our own experience in developing interactive applications with it. We also evaluate the API's usability from the perspective of independent programmers. Finally, we provide an evaluation of

the resulting system's usability from the perspective of an end-user interacting with a real-world deployment of a public display. The evaluation results indicate that PuReWidgets is an efficient, usable, and flexible toolkit for web-based interactive public display applications.

By making this toolkit publicly available, we hope to promote the development of more and newer kinds of interactive public display applications inside and, more importantly, outside the research community.

Keywords: Interactive public displays, Interaction abstractions, Programming toolkit, Socially situated displays.

Resumo

Um Toolkit de Abstracções de Interacção para Aplicações de Ecrãs Públicos

Os ecrãs públicos digitais estão cada vez mais presentes na nossa paisagem tecnológica. Considerando a sua flexibilidade e capacidade de ligação em rede, os ecrãs públicos têm o potencial para se tornarem num importante canal de comunicação e talvez até atingir a atenção, utilização e relevância que os *smartphones* têm hoje em dia. A interactividade dos ecrãs públicos é reconhecida como um elemento chave para os tornar mais atractivos e valiosos, mas a maioria dos sistemas de ecrãs públicos actuais ainda não suporta nenhuma forma de interacção. Uma razão por detrás deste aparente paradoxo é a falta de abstracções claras e eficientes para incorporar interactividade nas aplicações para ecrãs públicos. Apesar de a interacção poder ser conseguida para sistemas específicos, com uma modalidade de interacção específica, a falta de abstracções de interacção apropriadas significa que é necessário demasiado trabalho específico fora das funcionalidades nucleares da aplicação para suportar até as formas mais básicas de interacção.

Neste trabalho, investigamos e desenvolvemos abstracções de interacção para ecrãs públicos. Começamos por analisar os ecrãs públicos do ponto de vista da informação que resulta das interacções e de que forma pode ser utilizada em procedimentos de adaptação de conteúdo para ecrãs públicos. Chamamos a esta informação *digital footprints*, e o resultado é uma estrutura conceptual que mapeia as *digital footprints* em estratégias de adaptacção e em mecanismos de interacção. Esta estrutura pode ser utilizada por designers de ecrãs públicos para ajudar a escolher os mecanismos de interacção que um determinado ecrã deve suportar de forma a poder recolher um determinado conjunto de *digital footprints*, criando assim ecrãs com conteúdos mais relevantes e que são capazes de se adaptar ao seu ambiente social. De seguida, identificamos e caracterizamos tarefas de interacção e controlos apropriados para interacção com ecrãs públicos. Esta análise resulta num espaço de desenho que pode servir de base para *toolkits* de interacção, dando uma referência aos designers do sistema para os tipos de controlos que podem ser incorporados no *toolkit*. Finalmente, projectamos, implementamos e avaliamos um *toolkit* de abstracções de interacção para aplicações para ecrãs públicos – o *toolkit* PuReWidgets. Os programadores podem utilizar este *toolkit* para incorporar facilmente funcionalidades interactivas nas suas aplicações, baseadas na web, para ecrãs públicos. O PuReWidgets fornece abstracções de alto nível que protegem os programadores dos detalhes de baixo nível associados aos mecanismos de interacção. O *toolkit* é avaliado segundo várias dimensões. Primeiro, avaliamos o desempenho do sistema. De seguida, avaliamos

a flexibilidade e capacidades da API, usando a nossa própria experiência no desenvolvimento de aplicações interactivas. Avaliamos também a usabilidade da API da perspectiva de programadores independentes. Finalmente, avaliamos o *toolkit* da perspectiva dos utilizadores que interagem com um ecrã público num ambiente real. Os resultados da avaliação indicam que o PuReWidgets é um *toolkit* eficiente, flexível e usável para aplicações interactivas para ecrãs públicos.

Ao tornar este *toolkit* disponível publicamente, esperamos promover o desenvolvimento de mais aplicações interactivas para ecrãs públicos dentro e, mais importante, fora da comunidade de investigação.

Palavras-chave: Ecrãs públicos interactivos, Abstracções de interacção, Toolkit de programação, Ecrãs socialmente situados.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

AWT Abstract Window Toolkit

CMS Content Management System

CSS Cascading Styles Sheet

DPAA Digital Place-based Advertising Association

DHTML Dynamic HyperText Markup Language (HTML)

DTMF Dual-Tone Multi-Frequency

DOM Document Object Model

GPS Global Positioning System

GUI Graphical User Interface

GWAP Games With A Purpose

GWT Google Web Toolkit

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

IDE Integrated Development Environment

IM Interaction Manager

ISL Interface Specification Language

LCD Liquid Crystal Display

MAC Media Access Control

MMS Multimedia Message Service

NFC Near Field Communication

OBEX OBject EXchange

OS Operating System

OTS Opportunity to See

OVAB Out-of-home Video Advertising Bureau

PDA Personal Digital Assistant

PUC Personal Universal Controller

QR code Quick Response code

REST Representational State Transfer

RFID Radio-Frequency Identification

RSS Really Simple Syndication

SDL Service Description Language

SMS Short Message Service

SNS Social Networking Site

SVG Scalable Vector Graphics

UI User Interface

UIID User Interface Implementation Description

URC Universal Remote Console

URL Uniform Resource Locator

USB Universal Serial Bus

WAP Wireless Application Protocol

WLAN Wireless Local-Area Network

XML eXtensible Markup Language

XSL eXtensible Stylesheet Language

YUI Yahoo! User Interface

Chapter 1

Introduction

Contents

Public digital displays have become an ubiquitous presence in our everyday lives. We encounter them in the streets while we drive or walk, in shopping centres, gas stations, subway stations, universities, shops, banks, etc. However, most of them still do not provide any interactive features, even though interaction is repeatedly pointed out as a key-enabling element towards more engaging and valuable public displays. What is currently missing are interaction abstractions that designers and developers can use to incorporate interactivity into their public display applications in an easy manner.

Developments in display technology, particularly Liquid Crystal Display (LCD) and plasma displays, allowed the development of thin and light displays at economically attractive costs. This, in turn, has originated a wide spread deployment of these displays with various sizes and with various functions. Increasingly, digital displays are being used to substitute some older (mostly paper-based) forms of displaying visual information and to introduce many new ones. These new displays have several properties that make them attractive when compared to other media:

- They allow the presentation of dynamic information, and faster updating of the information that is displayed.
- They are usually connected to devices with processing capabilities that can be coupled with a variety of sensors, allowing the displays to become "smart displays" that react to various environmental factors, including people.
- They can be networked, taking advantage of communication networks to access remote information and to create an interconnected landscape of digital displays.

This widespread use of digital displays has fostered significant research around the concept of situated displays. This line of research aims at studying how digital displays can be integrated into their environment and be designed to support the activities and information requirements of a particular place. Public situated displays are a particular class of situated displays, meant to be shared by all visitors, whether frequent or occasional ones, of a specific public, or semi-public, place.

The overall idea of a public situated display is that it is able to deliver "the right information at the right time". This means that the display must be able to show relevant and interesting content to an audience which will have a particular expectation for content in that particular place, and in a particular moment in time.

1.1 Motivation

The most common approach for public displays still inherits much of the traditional broadcast model where content is published from a central entity. Display owners use their knowledge about the type of place, the intended audience, and their own

interests, to define what might be relevant content. However, if all the decisions must be made a priori, they will not take into account the fluidity and heterogeneity of the social context around the display. This limited approach usually results in repetitive and unattractive displays that fail at engaging users and at providing relevant, adapted, content. Sometimes they even fail at being really seen [Huang et al., 2008].

Most current public display systems do not support any interactive features, even though interaction is an obvious path for addressing the problems of user engagement and content relevance. 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 leads to wasted development effort and to inconsistent interaction models and user experiences across different displays.

1.1.1 The importance of interaction for public displays

Displays that offer the possibility to interact can lead to stronger user engagement and user-generated content. They will also be able to produce traces of user activity upon which multiple adaptation processes can be implemented.

Interaction can be used to spark and mediate socialisation in public spaces as in the Opinionizer [Rogers and Brignull, 2002] or GroupCast [McCarthy, 2002] displays; to let people make public announcements (for example to sell or advertise something) as in WebWall [Vogl, 2002], or Digifieds [Alt et al., 2011]; to share content of interest to a group of people as in the Plasma Poster network [Churchill et al., 2003b]; to leave personal messages to specific people when they are near the display as in the SynchroBoard [Jansen et al., 2005]; to collaborate and show-off content to others as in the Dynamo display [Brignull et al., 2004]; to allow users to leave feedback about what they have seen in the display, as in the CommunityWall display [Grasso et al., 2003]; or to let users manifest interest in particular topics as a way to provide a characterisation of a place as in the Bluetooth Instant Places display system [José et al., 2008]. Although these different interactive features will not make sense in every place, whatever one is offered will generally contribute to a more interesting display from the user's point of view, and consequently to a more valuable display from the display owner's point of view.

Interactivity is also important for impact assessment and content adaptation for public displays. Although there are some audience metering products, such as the ones from Quividi [2013] and TruMedia [2013], that employ computer vision techniques to infer the number, distance, age group, gender and attention of people near a public display, these techniques can only provide an outside view of the display's use. They are not capable of accurately measuring direct engagement with a public display and so, have limited applicability as an impact assessment tool. Interaction

features on a display, on the other hand, can provide a much more accurate view of how the display is being used, when, and by what kind of people. When interacting with a display, people are directly or indirectly communicating their personal needs, views, likes and dislikes. The digital footprints that result from the various interactions with displays have the potential to provide information that can be used to assess the overall impact of the display, or of specific content, much like what happens today on the web where many of our actions are analysed to provide us with targeted content. These digital footprints can also be used as a way to automatically characterise the place where the display is installed and its audience, providing a base for automatic content selection and adaptation.

1.1.2 Difficulties in providing interactivity for public displays

Despite its recognised importance, current public displays have not yet attained their full interactive potential. Only a few interactive display systems have left the research labs and entered the everyday life of real users, but even these have not gained wide spread dissemination and utilization.

Designing and developing for public displays is very different from designing and developing for desktop systems. Just as Weiser [1991] points out in his ubiquitous computing vision, the software that runs on these yard sized displays, or "boards", is not the same as that for a desktop computer. The way people interact with these large(r) screens is necessarily different from the way people interact with a desktop computer. Not only because the size of the display and usual spatial configuration are different but also because their intended use is very different. While desktop interaction is currently a mature paradigm, with a well defined environment (a single user sits at a desktop and uses a mouse and keyboard to interact with a computer screen), public display interaction enjoys no such maturity. When it comes to interaction with public displays, although it can easily be recognised as important, and even a central feature of a display, the questions of what interactive features to provide and how to provide them to users are still largely unanswered. There is not yet a good enough language to think or talk about interactive features in a public display nor tools to help developers implement them. Although it is currently easy to implement a single interactive public display with specific features and interaction mechanisms, it is very difficult to generalise that implementation to include other features and mechanisms because there are no accepted interaction abstractions. This is particularly true for non touch-based public displays, where interaction often occurs at a distance through the use of some personal interaction device. Bellucci et al. [2010, p. 72] succinctly reinforce this idea:

> "*At present, there are no accepted standards, paradigms, or design principles for remote interaction with large, pervasive displays.*"

Because interaction with public displays is very different from interaction with desktop computers, developers of applications for public display cannot directly apply

their knowledge from developing desktop 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. Additionally, this is an effort that must be replicated by each developer, representing wasted effort and leading to inconsistent interfaces. Users of these public displays are faced with very different interaction models and inconsistent interfaces across and, sometimes, within displays, which inhibits knowledge transfer about how to interact with different displays.

There is a clear analogy between the current situation with public displays and the time when desktop computer programmers had to make a similar effort to support their interaction with users. This was recognised as a problem and addressed with the emergence of reusable high-level interaction abstractions that provided consistent interaction experiences to users and shielded application developers from low-level interaction details. One of first successful developer's toolkit for Graphical User Interface (GUI) applications was the XToolkit [Swick and Ackerman, 1988], which has served as the basis for various other widget toolkits still in use today. Nowadays, with the wide availability of interaction widgets, developers can benefit from ready-to-use interaction elements that deal with input and encapsulate behaviour and visual appearance. Users have learned to interpret their affordances in a way that enables them to more easily tackle new interfaces and programs by building on their previous experience.

1.1.3 Benefits of an interaction toolkit for public displays

The underlying idea in this thesis is that the way to ease the burden of public display application designers, programmers and users is to follow a similar path to the one taken by the desktop platform and to develop an interaction toolkit that incorporates useful interaction abstractions, supporting the specificities of public display interaction.

> "*A user-interface toolkit is a library of interaction techniques, where an interaction technique is a way of using a physical input device (such as mouse, keyboard, tablet, or rotary knob) to input a value (such as command, number, percent, location, or name), along with the feedback that appears on the screen.*" [Myers, 1989, p. 16]

A toolkit is thus responsible for drawing the components on the screen, capturing the user input, giving input feedback (usually graphically) to the user and to inform the application about input events over the components. Toolkits allow programmers to create graphical user interfaces by laying out these components, programmatically or, in some cases, visually. These different responsibilities of a toolkit may not all have the same importance for public displays, but their essence should still be present in an interaction toolkit for public displays.

It is generally agreed that these libraries of user-interface components – or widgets – immensely ease the task of programmers. Without the toolkits, programmers would have to delve into low-level system calls to poll user input and deal with the details of the graphical appearance of the whole application interface. This is especially important for public display applications because, as we have argued, there is much less knowledge about how to build these applications and how to provide interactive features.

However, toolkits are beneficial not just because they take some of the effort out of programming graphical user interfaces, but also because:

> "*toolkits for novel and perhaps unfamiliar application areas enhance the creativity of these programmers. By removing low-level implementation burdens and supplying appropriate building blocks, toolkits give people a 'language' to think about these new interfaces, which in turn allows them to concentrate on creative designs.*" [Greenberg, 2007, p. 139]

Particularly for public displays, which are still a very young area in many aspects, including the user interface, toolkits can be an unblocking factor for the rise of more creative public display applications.

A toolkit for public display interaction would greatly facilitate not only the designer's and the programmer's task but also the final user's experience. For the designer, responsible for defining the behaviour and interactivity of an application, it provides a language to think about and decide which interactive features should be available and what the user's experience will be like. For the programmer, responsible for the implementation of the public display application, a toolkit will obviate the need to think about and implement the low-level details regarding the variety of input mechanisms, input handling, and input feedback. For the user of a public display application, the consistency of the interaction model, the types of controls, their behaviour, visual appearance, and feedback, will allow him to build up knowledge about how to interact with public displays. When faced with a new display application, his past experience with other displays will enable him to easily understand the new one.

By facilitating the development of interactive content, the existence of a user-interface toolkit will also contribute to the emergence of one or more instances of the concept of *public display application*: software entities developed by third-parties that can run on various public displays owned by different people. The possibility of anyone developing an application for a public display, much like anyone does today for desktop computers, would be an important step towards the development of more and new uses of interactive public displays. It will also contribute to moving public displays away from a world of closed display networks that are operated as multiple isolated islands, to scenarios in which large-scale networks of pervasive public displays and associated sensors are open to applications and content from many sources [Davies et al., 2012]. In these scenarios, displays would become a communication medium ready to be appropriated by users for their very diverse communication goals, and interaction would necessarily become an integral

part of the whole system. Developers should be able to create interactive display applications that would run across the many and diverse displays of the network.

1.2 Challenges

There are several reasons why it is challenging to develop an interaction toolkit for public display application development. In order to be successful, the toolkit must be grounded on an understanding of the particular properties of the interaction that happens with public displays. To analyse and characterise the challenges posed by public display systems, we use the same structure that Bellotti et al. [2002] have used to analyse "sensing systems". This structure is based on five questions:

- How do I address one (or more) of many possible devices?
- How do I know the system is ready and attending to my actions?
- How do I effect a meaningful action, control its extent and possibly specify a target or targets for my action?
- How do I know the system is doing (has done) the right thing?
- How do I avoid mistakes?

Address

The first question a user needs to answer to be able to use any computer system is: how do I address one (or more) of many possible devices? This question is so basic that we usually do not even consider it when thinking about desktop computers, in which the system and the devices used to interface with it are very well defined: the computer box, monitor, mouse and keyboard. These assumptions, however, do not hold for public display systems. While we can expect a keyboard and mouse to be available when we sit at a desktop computer, we cannot expect them to be present to interact with a public display. In fact, we cannot expect that there will be a single input mechanism to interact with the display, or even that the same set of mechanisms will be available for all interactive public displays. A variety of different input mechanisms have been explored to interact with public displays. Camera-phones, Bluetooth naming, Bluetooth OBject EXchange (OBEX) file exchange, Short Message Service (SMS), email, instant messaging, Twitter, Radio-Frequency Identification (RFID) tags, gestures, face detection, and many others, have been employed in research and commercial systems as input mechanisms for public displays.

Faced with a public display, users should be able to understand what interaction mechanisms are available and how to use them. The challenge is not only in communicating the available mechanisms to users, but also in integrating and abstracting these various possible mechanisms in a consistent way for programmers. In order to

develop an interaction toolkit for public displays, we must understand the capabilities of each interaction mechanism and determine which interaction features can be supported by each mechanism in what way.

Attention

How do I know the system is ready and attending to my actions? On desktop computers, we recognize the affordances of the most basic graphical components such as buttons, textboxes, etc., and their possible states. We know that they are usually waiting for our input, unless they are in a disabled state which we are also usually capable of distinguishing. We know that the fact that the mouse cursor moves in response to our actions is an indication of readiness to accept input and, in many cases, the graphical feedback of the components under the mouse cursor immediately tells us whether they can accept a mouse click or not. These things are taken for granted when we build desktop computer applications. However, they essentially rest on an interaction style – direct manipulation – that may not be possible or desirable to apply on a public display, given the breadth of available input mechanisms. In addition, public display applications may not always be visible on-screen, but still be receptive to input. In this situation, how do users know the application even exists, given that usually users don't have full control over what is displayed and when?

Communicating the interactive features of the public display is a significant challenge. Although related to the previous challenge, the issue here is helping users determine that a given content is interactive and what features it offers, regardless of the interaction mechanism that will be used. Ideally, displays should be easily recognizable as interactive and users should have the possibility to discover what features it offers. The challenge is that there aren't any accepted graphical representations that an interaction toolkit for public displays can use.

Action

How do I effect a meaningful action, control its extent and possibly specify a target or targets for my action? In a desktop computer we have come to identify and learn to use the common controls such as icons, buttons, menus, windows, text boxes, etc. We know we can act on a button by positioning the mouse cursor over it and clicking, we know that dragging a scroll bar will make the associated window's content scroll up or down, we know that the action associated with a menu item will be performed on some previously selected graphical object, and we easily identify form pages to introduce various types of data. Public displays don't generally benefit from this accumulated knowledge, because there are no accepted interaction paradigms and abstractions.

This is perhaps the major challenge for interactive public displays: creating interaction abstractions that programmers and users learn to identify and use, in a usage

context that can be much more limited than the desktop context. An interaction toolkit for public displays must provide high-level controls, but first we must understand what those controls should be and how we can abstract them from the various possible interaction mechanisms that can be used to act on them.

Alignment

After a user has issued some input to the system how can he determine that the system is doing the right thing? The typical answer of desktop computer systems is to use several feedback mechanisms such as echoing input characters, showing progress bars, visually changing the state of graphical components on the screen, or simply showing the immediate result of the action such as when a user scrolls a document. The desktop computer solution is based on continuous and immediate feedback to user's actions. However, because public displays are shared objects and can be interacted with remotely, the desktop solution cannot be generally applied.

The main challenge for public displays is in directing timely and appropriate feedback information to the correct user, on the correct channel, so that users understand the result of their actions. Given that public displays are shared devices that can be used simultaneously by multiple users and applications, it is not obvious what the feedback information should be and how it should be communicated.

Accident

How do I avoid (or recover from) mistakes? The desktop computer answer for this question relies on various mechanisms to deal with mistakes. Previews allow users to inspect the final result before starting the actual, usually lengthy, action. For example, before actually printing a document, a user can ask for a preview to determine if the printed document will look as desired. Undo allows users to correct mistakes after they have been made and take the system back to a previous state. Applications usually provide several levels of undo so that users can correct even mistakes made several actions before. Cancel can be used for lengthy operations so that users can abort the operation during its execution if an error is detected. Cancel can also be used for operations that require users to perform several steps allowing them to terminate the operation in any step. All these mechanisms also require continuous feedback so that the user can determine that a mistake has been made and can then take action to correct it.

The challenge for public displays is finding the right error prevention and correction mechanisms that work in an interaction environment where there are multiple simultaneously interacting users, possibly interacting with different applications, which may not all be visible on the public display at the same time.

1.3 Objectives and Contributions

The general objective of this work is to develop high-level interaction abstractions and tools for public display applications that support public, shared, remote interaction, abstracting the low-level details of various input mechanisms. An important aspect of this research is the focus on *applications for public displays*. Our goal is that the developed abstractions contribute to the emergence of open display networks. Ideally, applications should be as much as possible agnostic of the concrete interaction mechanisms available in a particular display. This general goal can be further refined in the following objectives.

Objective 1: Digital footprints for socially-aware public displays. Our first objective is to analyse how the information that results from interactions with public displays may be used to drive content adaptation behaviour on public displays. This analysis should result in a framework that maps digital footprints and adaptation strategies that can be used to create more relevant displays that are able to automatically adapt to their environment. This mapping can be used by situated display designers to help them choose the interaction mechanisms that a display should support in order to be able to collect a given set of footprints.

Objective 2: Interaction tasks and controls for public displays. Our second objective is to identify and characterise a new set of interaction tasks and controls that are appropriate for public display interaction. This should result in a design space of interaction tasks and controls that can become a tool to structure an interaction system for public display applications. This can also be a valuable tool for allowing application developers to make more informed decisions on the types of controls that they would need, considering for example the application's goal but also the envisioned interaction mechanisms.

Objective 3: A programming toolkit for public display applications. Our third objective is to design, implement, and evaluate an interaction toolkit that programmers can use for incorporating interaction features into public display applications. This toolkit should be framed by the requirements imposed by public display interaction and it should abstract the interaction details of multiple interaction mechanisms into high-level events.

Contributions

The main contribution of this work is a toolkit of interaction abstractions for public displays that can help designers think about the interactive features of public display applications, can help programmers to implement those features by reducing the amount of work they need to accomplish, and can help users by providing a consistent interaction model for public displays. More specifically, pursuing the previously outlined objectives resulted in the following contributions:

1. A framework of digital footprints that designers of public displays can use to map several sensing and interaction features to content adaptation strategies.
2. A design space of interaction tasks and controls that can serve as a basis for programming toolkits that provide high-level interaction widgets.
3. Identification of the fundamental requirements for an interaction abstraction toolkit for public displays.
4. The PuReWidgets software toolkit itself for incorporating interactive features into public display applications
5. An in-breadth evaluation of the PuReWidgets toolkit along several dimensions, including the system's performance and scalability, the API usability, and a real-world deployment.
6. This work has also contributed to the research and programming community with several open-source software projects, allowing anyone to use, modify, and adapt for further research and development. The following open source projects were initiated during the course of this work:
   - `https://code.google.com/p/purewidgets/`
     - The PuReWidgets toolkit itself.
     - Three interactive public display applications (a video player application, a polls application, and a word game application).
   - `https://code.google.com/p/public-display-scheduler/` A Google Chrome extension that serves as an application scheduler for public displays.

Related publications

This work has also originated or contributed to two journal papers, one book chapter, six conference papers, one conference poster, and one conference demo:

Under review

1. Cardoso, J. C. S., & José, R. (2013) Interaction tasks and controls for public display applications. Submitted to the International Journal on Human-Computer Interaction. (under review)
2. José, R., Cardoso, J. C. S., Pinto, H., & Hong, J. (2013) Expressing and interpreting media sharing in a network of public displays. Submitted to the 12th International Conference on Mobile and Ubiquitous Multimedia - MUM '13. (under review)
3. Cardoso, J. C. S., & José, R. (2013) Evaluation of a programming toolkit for interactive public display applications. Submitted to the 12th International Conference on Mobile and Ubiquitous Multimedia - MUM '13. (under review)

Published or accepted for publication

4. Taivan, C., José, R., Elhart, I., & Cardoso, J. C. S. (2013). Design considerations for application selection and control in multi-user public displays. Journal of Universal Computer Science, (Towards Sustainable Computing through Ambient Intelligence). (accepted for publication)

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

6. Cardoso, J. C. S., & José, R. (2012). Creating web-based interactive public display applications with the PuReWidgets toolkit (Demo). Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia - MUM '12 (p. 1). New York, New York, USA: ACM Press. doi:10.1145/2406367.2406434 [Cardoso and José, 2012a]

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

8. Cardoso, J. C. S., & José, R. (2012). The PuReWidgets Toolkit for Interactive Public Display Applications (Poster). Proceedings of the International Symposium on Pervasive Displays (PerDis'12). Porto. [Cardoso and José, 2012c]

9. Cardoso, J. C. S., & José, R. (2011). Assessing Feedback for Indirect Shared Interaction with Public Displays. In R. Meersman, T. Dillon, & P. Herrero (Eds.), On the Move to Meaningful Internet Systems: OTM 2011 Workshops (Vol. 7046, pp. 553–561). Springer Berlin / Heidelberg. [Cardoso and José, 2011]

10. José, R., & Cardoso, J. C. S. (2011). Opportunities and Challenges of Interactive Public Displays as an Advertising Medium. In J. Mueller, F. Alt, & D. E. Michelis (Eds.), Pervasive Advertising (pp. 139–157). Springer-Verlag London Limited. [Jose and Cardoso, 2011]

11. Cardoso, J. C. S., & José, 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. [Cardoso and Jose, 2009]

1.4 Outline of this Document

The rest of this document is structured in the following way:

Chapter 2 – Related Work – presents work related to public displays and interaction abstractions. We describe several existing public display systems and analyse how interaction has been implemented in each of them. We also describe several approaches for software support for interactive application development and analyse how they can contribute to an interaction abstraction solution for public displays.

Chapter 3 – Requirements for Interaction Abstractions for Public Displays – characterises the interaction environment around public displays, and defines a set of requirements for an interaction abstraction for public display applications.

Chapter 4 – Digital Footprints for Socially-Aware Interactive Displays – presents a framework for designing interactive displays that are able to gather interaction footprints that can be used for automatic and dynamic content adaptation.

Chapter 5 – Interaction Tasks and Controls for Public Display Applications – identifies a new set of interaction tasks focused on the specificities of public display interaction, and proposes concrete interaction controls that may enable those interaction tasks to be integrated into applications for public displays.

Chapter 6 – The PuReWidgets Toolkit – A Widget-based Interaction Abstraction for Public Displays – proposes a concrete interaction programming toolkit for public display applications, describing its main concepts, architecture and usage.

Chapter 7 – Evaluating PuReWidgets – evaluates the proposed toolkit on several dimensions: system performance, API flexibility and usability, and resulting system's usability from the end-user perspective.

Chapter 8 – Conclusions – presents some concluding remarks and points to further research opportunities.

Chapter 2

Related Work

Contents

2.1 Introduction

This chapter's objective is to describe and analyse previous work in relevant areas for this thesis and to position our work in the current state of the art. The analysis is structured around two main directions: interaction in existing public display systems, and solutions for providing software support for developing interactive applications.

First, we describe various existing interactive public display systems, and analyse the interaction mechanisms and features they support, along with how interaction affects the display's content adaptation. Second, we describe various types of software support for the development of interactive applications for interactive systems, and analyse how they could contribute to an interaction abstraction solution for public display applications.

2.2 Interaction in Public Display Systems

Interactive public displays have been around for some time now, and many different displays with different purposes and interaction capabilities have been developed and tested. In this section, we analyse various display systems, focusing on the aspects related to the interaction. Each display system is analysed according to three related perspectives: interaction mechanisms, interaction features, and interaction influence on content. The analysis of the interaction mechanisms considers what physical devices are necessary and how they are used to interact with the public display. The analysis of the interaction features considers what actions are available to users and how they are performed using the interaction mechanisms. The analysis of the interaction influence on content considers how the display system takes advantage of the information that results from the various interactions to adapt the displayed content.

At the end of this section, we compare and synthesize the systems according to the interaction aspects we defined previously. To organize the presentation of public display systems considered in this section, we classified them according to the main interaction mechanism used to interact with the display, and grouped them in the following categories:

Desktop-like Display systems that use a mouse and keyboard as the main interaction device to interact with the public display. We include in this category display systems in which users use a mouse and keyboard to interact directly with the public display, or indirectly through desktop applications (including web interfaces).

Touch-screen Interaction is accomplished by directly touching the public display. In this category, we don't distinguish among the various touch or multi-touch technologies.

Mobile device The standard communication features of a mobile device, or a custom mobile application are used to interact with the public display.

Id badge Users wear a personal id badge that the display system uses to detect and identify users.

Device-free Interaction with the display is accomplished by simply passing-by the display or standing in front of it and gesturing, without having to wear or use any device.

These categories are not mutually exclusive, as their frontiers are not clear-cut, and some display systems use more than one interaction mechanism. In the following description however, we have classified each system with a single category.

2.2.1 Desktop-like

In this section we analyse display systems in which interaction follows a Graphical User Interface (GUI) style: a mouse and keyboard are used to interact directly with the interface on a public display, or indirectly through desktop applications or web interfaces.

Opinionizer

The Opinionizer [Rogers and Brignull, 2002] is display system meant to work as an ice-breaking mechanism in social settings such as parties:

> *"The aim was to provide a milling audience with a forum to post their opinions and comments about a particular topic. The intention was that it be lighthearted and lightweight, allowing people to easily 'step in and out' of the limelight when interacting with the system and for others to be able to comment to each other or the 'interactor' with relative ease and, importantly, not to feel embarrassed when doing so."* [Rogers and Brignull, 2002, p. 2]

The Opinionizer is a wall-projected display where people interact by using a standard keyboard and mouse attached to the same computer that drives the projection. The system displays a topic in the form of a question and displays the submitted comments and opinions as speech bubbles coming out of avatars placed in one of the four colour quadrants of the display (see Figure 2.1).

Interaction with the Opinionizer system consists in selecting a cartoon avatar (the available avatars depend on the type of social gathering), selecting a speech bubble type (speaking, shouting or thinking), entering a nickname, writing an opinion, and dragging the avatar to one of the quadrants of the display.

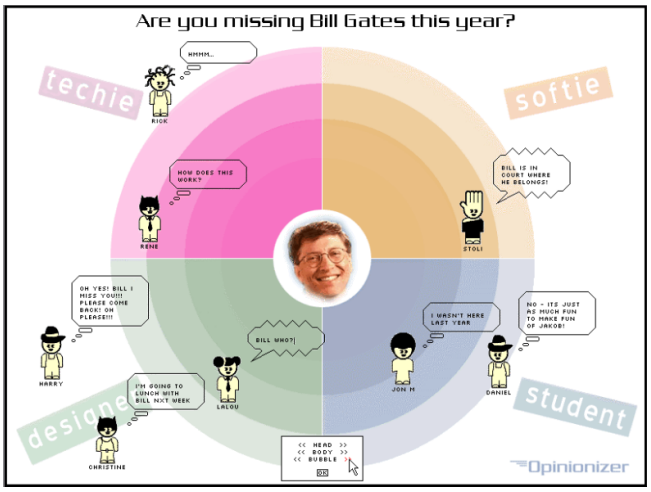


Figure 2.1: The Opinionizer display [Rogers and Brignull, 2002].

Dynamo

The Dynamo display system [Brignull et al., 2004], is a large multi-user interactive display for sharing, exchanging, showing and interacting with multimedia content in a communal room of a high school (see Figure 2.2).

(a) Dynamo surface detail: two carved regions (indicated by the icon of a key).

(b) Display in use in the communal room.

Figure 2.2: The Dynamo display [Brignull et al., 2004].

Dynamo provides a desktop-like GUI accessible from various interaction points (wireless mice and keyboards) that can be placed freely around the space. Each interaction point is represented on the display by a colour-coded mouse cursor. Dynamo allows two levels of user interaction: anonymous and identified interaction. In the first level, anyone can use the display to access public content areas or to access a personal Universal Serial Bus (USB) device and view or place media on the public areas of the display. In the second level, registered users can perform more advanced interactions such as carving a region of the screen for private use of a group of chosen users (see Figure 2.2a). Registered users can also share items with other users and leave notes on media items. Sharing is done using parcels – a bundle of media items that can only be opened by their intended target users. Asynchronous discussions are also possible in Dynamo by using notes – textual information that can be associated with media items on the screen. Content is fully created, managed and presented by the Dynamo users.

IM Here

IM Here [Huang et al., 2004] is a groupware display system that tries to bring the value of instant messaging to other places were people work, beyond the personal desktop:

> *"We developed the IM Here system as an awareness and communication tool that takes advantage of IM for lightweight interactions and large-scale displays for their walk-up-and-use nature. The system uses the large, highly visible form factor to promote group awareness of upcoming events. The large display also emphasizes the fact that its IM capabilities are a shared resource for the workgroup."* [Huang et al., 2004, p. 279]

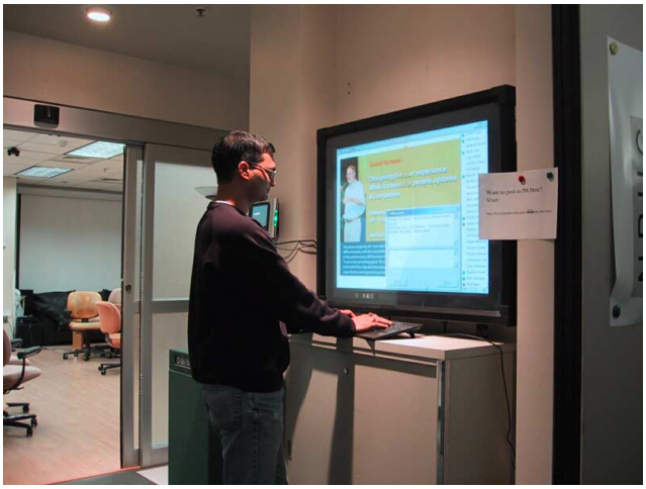


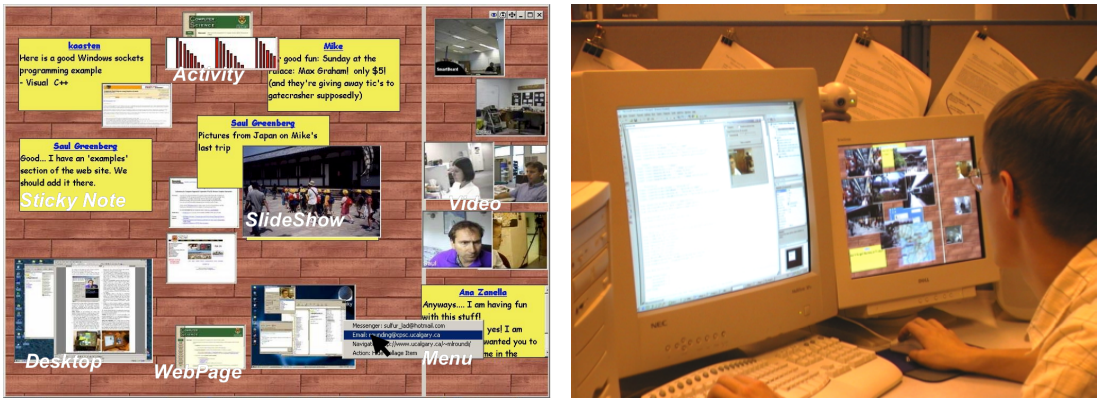
Figure 2.3: The IM Here display [Huang et al., 2004].

The IM Here display system is composed of two components: the IM Here Messaging Client, and the IM Here Event Display. The messaging client provides awareness about workgroup members' presence and availability by displaying their instant

messaging status on the public display, and allows workgroup members to broadcast messages directly from the public display to all workgroup members. Interaction with the IM Here messaging client is done with a standard mouse and keyboard (see Figure 2.3). The IM Here Event Display shows upcoming events and announcements that are created using a web interface accessed on a personal machine. The web interface provides a form that allows users to enter information about the event such as the title, description, date, location, and expiration date. The public display automatically schedules the display of these events based on their expiry dates (it cycles through the current events showing each one for 25 seconds).

Notification Collage

The "*Notification Collage (NC) is a groupware system where distributed and co-located colleagues . . . post media elements onto a real-time collaborative surface that all members can see*" [Greenberg and Rounding, 2001, p. 515]. NC's goal was to provide a platform for supporting interpersonal awareness and interaction in small communities of people and to see how, and what, social practices would emerge around it. The system was designed so that work colleagues could share their work, or topics of interest, increasing awareness of co-workers' activities. It was designed in a bulletin board style that accepts elements such as live video feeds from desktop cameras, sticky notes, activity indicators, slide shows with photos, desktop snapshots and web page thumbnails (see Figure 2.4a).



(a) Public display interface. (b) Personal display.

Figure 2.4: The Notification Collage display [Greenberg and Rounding, 2001].

All interaction with NC is done via a desktop application (see Figure 2.4b) that is mirrored in the public display. The desktop application allows users to post elements and interact with the existing ones: users can respond to the author of the item (by email or instant messaging), visit the homepage of the poster or web thumbnail.

Items are graphically arranged automatically on the public display: new items are posted in the top of the display and gradually move down as they age. However, NC only arranges the left hand side of the display, the positioning of items on the right hand side is manually controlled by its users, using the desktop application.

MessyBoard

MessyBoard [Fass et al., 2002] is similar in functionality to the Notification Collage system. It's a display system meant to provide a shared communication space for a workgroup:

> "*MessyBoard is a large, projected, shared 2D bulletin board that allows users to share pictures and text over the internet*" [Fass et al., 2002, p. 303].

(a) MessyBoard's screen. (b) MessyBoard's projection in context.

Figure 2.5: The MessyBoard display (adapted from [Fass et al., 2002]).

MessyBoard evolved from a Windows desktop replacement application called MessyDesk that allowed Windows users to decorate their desktop with images or text from any application by dragging or pasting from the clipboard (see Figure 2.5a). While MessyDesk was a personal tool, MessyBoard transferred that functionality to a shared tool. MessyBoard runs as a regular application composed of a window that is shared among several users and projected into a wall in a workspace (see Figure 2.5b). Unlike Notification Collage, in MessyBoard the placement of the various items on the screen is completely defined and controlled by its users.

CoCollage

CoCollage by McCarthy et al. [2009] is a situated public display designed to increase the sense of community and place attachment:

> "*The Community Collage is a new place-based social networking application designed to bridge the gaps between people in coffeehouses by bridging the gaps between the richness of their online interests and activities and their physical presence in a potentially 'great, good place' . . . CoCollage links online profiles, machine-readable loyalty cards and a large computer display that shows elements from those profiles when people use their loyalty cards in the café.*" [McCarthy et al., 2009, p. 225].

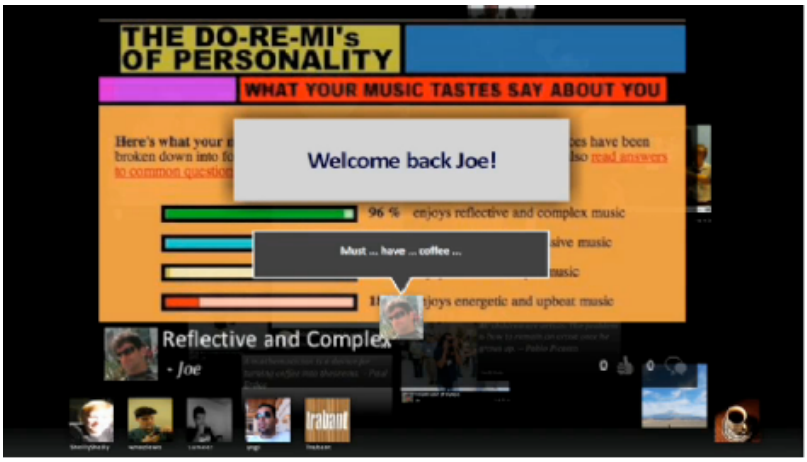


Figure 2.6: The CoCollage display welcoming a user [McCarthy et al., 2009].

Interaction with the CoCollage display is mainly performed through a web page where users can define an online profile with some basic information such as username, email address, birthday, avatar, and a collection of social media content. Social media content can be specified explicitly by uploading a file, or implicitly by Really Simple Syndication (RSS) feeds of other web services such as Flickr,[1] in which case the user needs to specify a username and a set of optional tags to filter the imported media. Users can also use the online web page to vote (thumbs up or down) on any item on the display's stream; to send messages directly to the display (these messages show up as balloons above the user's avatar); or to check in the café (i.e., to indicate that they are present). Check-ins can also be performed through the use of their magnetic loyalty card at the café. The CoCollage display shows a stream of content by periodically selecting an item from the users' profiles. The selection of the next item to show is based on several factors that include when the item was added, the last time it was shown, how many votes it has, how many comments, when it was last commented on or voted, and when the author last checked in. CoCollage also displays the avatar list of the currently present users in the bottom of the screen. Whenever a user checks in, the display shows a welcome message and the user's avatar and username (see Figure 2.6).

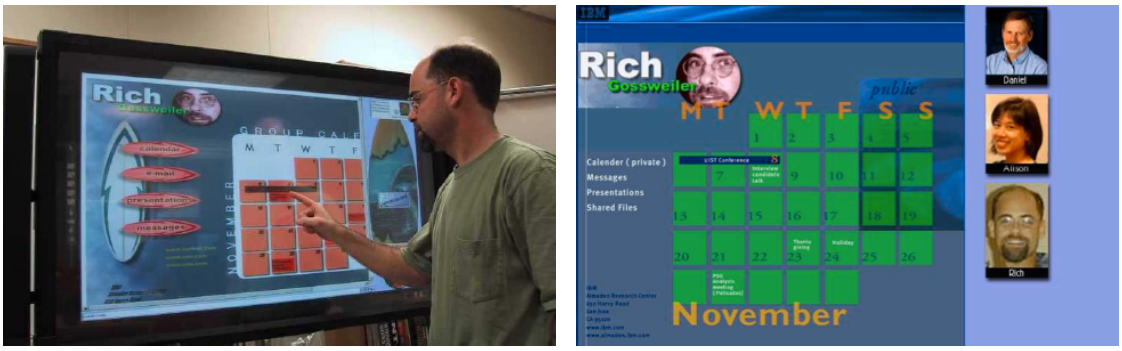
2.2.2 Touch-screen

In this section we analyse display systems in which interaction is accomplished by directly touching the public display. We consider only the cases where the public display itself is touch-enabled; we don't consider interactions that happen on personal touch devices.

[1] `http://flickr.com`

BlueBoard

The BlueBoard display system by Russell and Gossweiler [2001] (see Figure 2.7) was designed as a personal information tool and also as a collaboration and sharing tool for small groups. Blueboard was designed as a kiosk-style, large display, which allows access to personal information (instead of just a pre-defined set of information, as usually happens with public kiosks). It was also meant for simple collaboration tasks such as sharing content between small groups of users. BlueBoard uses a display with a touch-sensitive overlay for interaction and a badge reader for user authentication.



(a) A user's personal calendar. (b) P-Cons.

Figure 2.7: The BlueBoard display [Russell and Gossweiler, 2001].

Once a user authenticates by swiping a personal badge, he has access to his previously configured personal information. This information must be set by the user himself, using BlueBoard's authoring system, which allows users to link web content (calendars, email, files, etc.) to their BlueBoard profile. When a user authenticates, BlueBoard simply loads the Uniform Resource Locator (URL) of his personal area.

Several users can authenticate at the same time and access their personal data. Each user is represented by a *p-con* – an icon on the right side of the display. The p-con provides a target for content sharing by dragging the content into the other user's p-con.

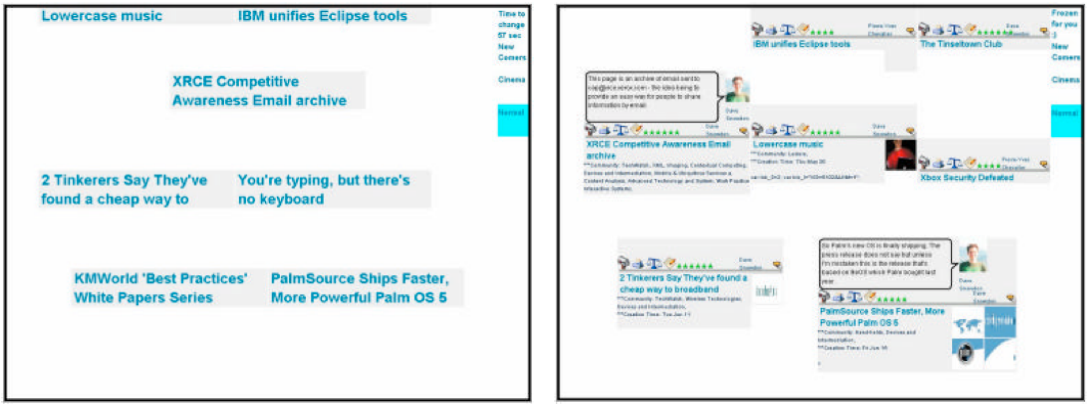
OutCast

OutCast [McCarthy et al., 2001] is an office door display that shows information about the owner (see Figure 2.8). Visitors to the office can access the following information about the owner: biography, calendar, location information, project information, demonstrations, favourites, and a user-defined text message. OutCast works in a passive mode and in an interactive mode. In the passive mode, it cycles through its content; in interactive mode, it allows users to navigate through the display's content, and even leave a message to the owner, using the touch-screen interface of the display.

Figure 2.8: The OutCast display [McCarthy et al., 2001].

Community Wall

The Community Wall (CWall), by Grasso et al. [2003], is a semi-public, interactive large screen that was installed at Xerox Research Centre Europe. The purpose of CWall was to support communication within a community of practice – a kind of spontaneous and informal workgroup that is driven by the common work-related interests and practices – and to "*create an environment that fosters social encounters (conversations) using documents or news and peoples' opinions on them as triggers*" [Grasso et al., 2003, p. 266].



(a) No user looking at the display.

(b) With user looking, the display shows interaction possibilities.

Figure 2.9: The Community Wall display [Grasso et al., 2003].

The display was designed as a bulletin board (see Figure 2.9b) that people can post to. Posting can be done using several methods such as email, web bookmarklets,[2] paper (using Xerox's Flowport(tm) software to scan the document) or a Personal Digital Assistant (PDA) application. Although users are completely responsible for the content database, CWall is responsible for choosing what to display from that database. A number of rules exist to define the priority of each item based on its

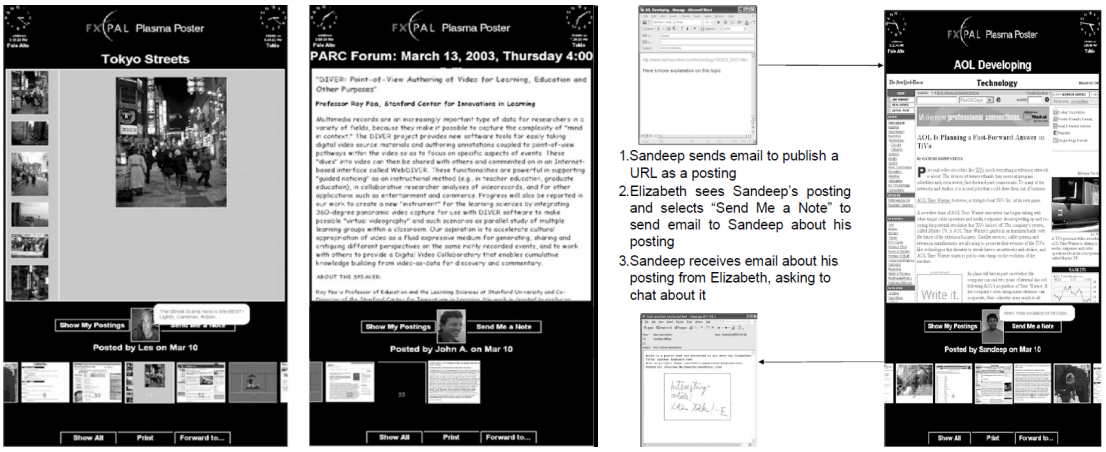
[2]A bookmarklet is a Javascript program that can be stored as a bookmark in a web browser and that can be used to automatically process the current page. In Community Wall the bookmarklet is used to send the current web page to the display system.

type, age, rating, number of comments, time, etc. After applying the rules, the system will present the highest priority items.

CWall employs a presence detection mechanism to change the way items appear on the screen. Presence detection is accomplished with a grid of infrared movement sensors and computer vision techniques for face detection. These techniques are used to: show bigger titles when people are in the room but far away from the display, to attract attention (see Figure 2.9a); show more detailed item information and freeze the content of the display when someone is detected reading so that it doesn't change contents while people are reading (see Figure 2.9b). CWall also provides touchscreen interaction and allows users to read an item in more detail, to rate, email, comment or print it. Rating and emailing requires users to identify themselves, or the target user (in case the item is being emailed), by selecting their image in a list of users presented by the display (users must have been previously registered in the display system).

Plasma Posters

The Plasma Posters [Churchill et al., 2004] is a network of interactive community boards developed for a community of a lab at FXPAL – a software research company, based in Palo Alto, California, USA – to informally share information. It is also a means to foster awareness about the interests of each of the community's members and to provide a visible expression to the lab's identity.



(a) Photo slideshow with author's comment in the text balloon.

(b) Formatted text item.

(c) Sending a note to the author of the post.

Figure 2.10: The Plasma Posters display [Churchill et al., 2004].

A Plasma Posters display consists of a large plasma display with a touch-sensitive overlay that allows people to interact with the items currently displayed. The content presented by the Plasma Posters displays is mostly the result of user submissions but it also automatically collects items from the intranet such as calendars of meetings and announcements of new technical reports. Posted content can be images (see Figure 2.10a), movies, formatted text (see Figure 2.10b) or web pages. The

authors of the posts can also submit an associated comment or highlight passages of text (in the case of web pages). Content can be posted by email or using a dedicated web interface.

A Plasma Posters display cycles through all the available content periodically, providing a peripheral or ambient display (items are removed after two weeks by default, but the duration of an item can be set manually). However, users can interact with it directly, with touch interaction, to browse the available content, read it (by pausing the automatic cycling), scroll and even print it. Users can also send notes to the author of the post (see Figure 2.10c) or forward the posted item to themselves or to another Plasma Poster user. To forward an item to someone, a user needs to select the appropriate button and then select the target user from a list of users registered in the Plasma Posters network.

Digifieds

"*Digifieds is a digital and networked public notice area designed to support passers-by when creating, sharing, and retrieving classified ads on public displays.*" [Alt et al., 2013].

Digifieds allows users to create, publish, and retrieve classified ads using the touch-screen of the display itself, using a mobile application, or using a web application (see Figure 2.11). Classifieds can be created in three ways. Using the touch-screen, a classified can be created by typing on the on-screen keyboard to write the text and then augmented with pictures or videos transferred from a USB memory stick. A mobile application can also be used to create a classified that can later be published on a Digifieds public display. Finally, classifieds can also be created using a web interface. After a classified is created, it has to be published on a public display. Classifieds created using the touch-screen are immediately published and available on the same display. Classifieds created on the mobile or web interface can be published by manually entering an alphanumeric code associated with the classified on the public display, using the on-screen keyboard. A QR code can also be used to publish a classified created on the mobile phone: the phone generates a QR code that a camera in the public display can read to activate the classified. Finally, classifieds created on the phone can also be published using an interaction technique similar to PhoneTouch [Schmidt et al., 2010] where the user touches the public display with the phone at the location he wants the classified to appear. Classifieds can be taken from a public display using one of five alternatives: touch with the phone on the public display (similar to publishing), using a mobile phone to scan the QR code that is displayed next to each classified, writing down the alphanumeric code that is also displayed next to the classified, forwarding the item to an email address using the on-screen keyboard, printing the classified on a printer that is attached to the public display.

Figure 2.11: Digifieds public display [Alt et al., 2013].

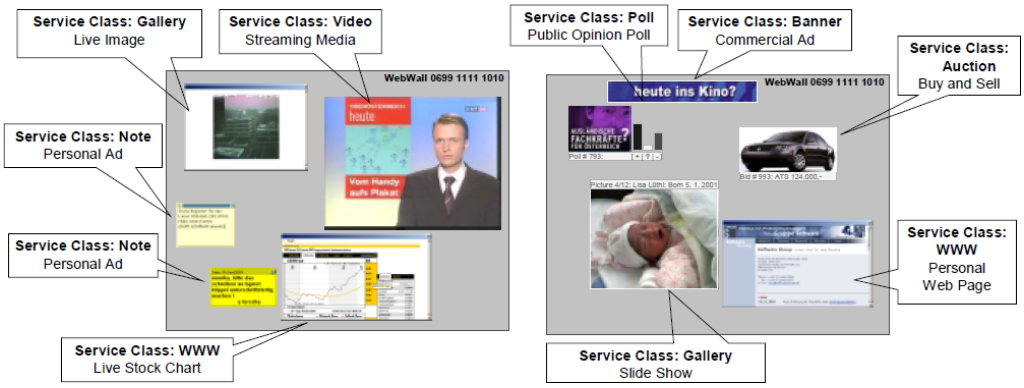
2.2.3 Mobile device

This section considers public display systems in which the interaction is accomplished via a mobile device. We consider the cases where the standard communications features of a mobile device are used for the interaction, e.g. SMS, Bluetooth, and the cases where a custom mobile application is required to interact with the public display.

Web Wall

WebWall [Ferscha et al., 2002] is an example of a display designed as a kind of digital bulletin board. The motivation for its development was to address the potential of ad-hoc communication in public spaces using a wall metaphor. WebWall is an infrastructure that can be used to create large displays that allow users to post various media elements and interact with existing ones.

WebWall supports several types of media elements, called *service classes*, which differ in visual appearance and functionality (see Figure 2.12a). An instance of a service class can be created or interacted with using a number of input mechanisms such as email, Short Message Service (SMS), Wireless Application Protocol (WAP) or using a web interface. To address a particular WebWall, users use the wall id (which is also the phone number for SMS interaction and the user part of the email address for email interaction). When created, an instance is assigned an instance id that uniquely identifies that element in that particular display (see Figure 2.12b). Interaction with WebWall is made using a command language that allows users to specify the service class (when creating a new item) and the properties of the item. For example, to create a note with a blue colour and the text "Hello WebWall!", a user could send the command "!note.blue Hello WebWall!" to the WebWall's assigned phone number (the Wall ID). To reply to an existing note, one would issue a command such as "!347 r Ok", where "347" is the instance id of the note and "r" indicates the reply command.



(a) Service classes.

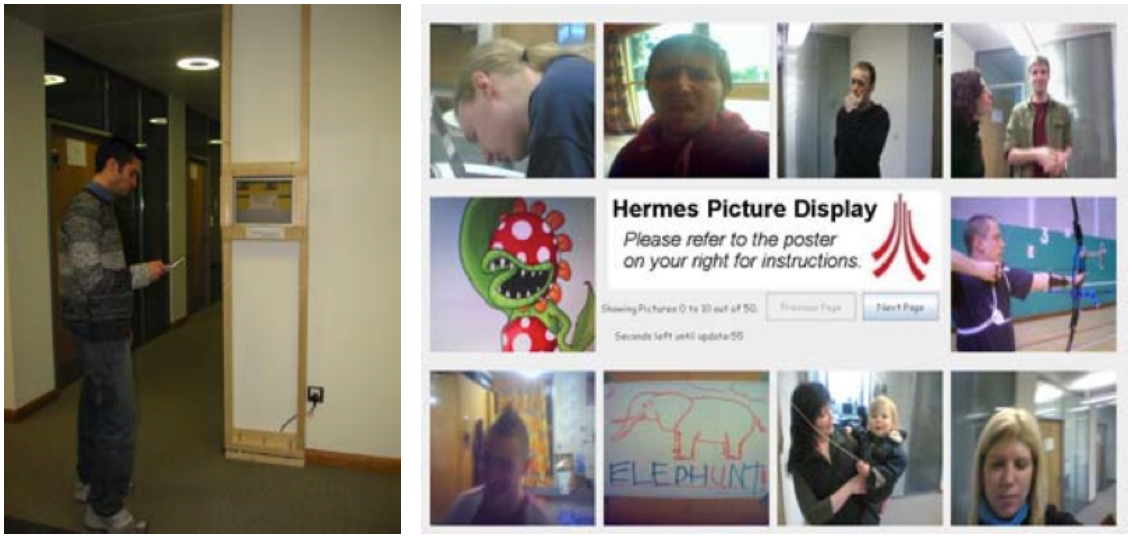
(b) Instance components.

Figure 2.12: The WebWall display [Ferscha et al., 2002].

Hermes Photo Display

The Hermes Photo display [Cheverst et al., 2005] is an extension to the Hermes office doorplate system [Cheverst et al., 2003] that enables users to send, receive and browse photos on the photo display. It consists of several small digital displays deployed near the office doors (see Figure 2.13a) of the Computing Department at Lancaster University, UK. This system was deployed with the objective of understanding the technical feasibility of the idea of interacting via Bluetooth with a photo display and to get insight about user acceptability and potential of such displays for supporting and fostering a sense of community.

The displays are touch-sensitive, so users can browse the current set of photos by simply touching the next or previous page buttons (see Figure 2.13b). Sending and receiving is done through Bluetooth communication. To send a photo, users need to select the photo from their Bluetooth enabled personal mobile device and select the "send via Bluetooth" option; the mobile device will then search for nearby Bluetooth devices; users need to select the one that corresponds to the intended display (for example "PubDisplay(C)", where C stands for the floor level); the photo will then be received by the display. To receive a photo on their personal device, users must select the intended photo on the Hermes display (touching the picture selects it); the display will then begin searching for nearby Bluetooth devices and display a list when finished; users must then select their own device from the list and accept the photo on their personal device.

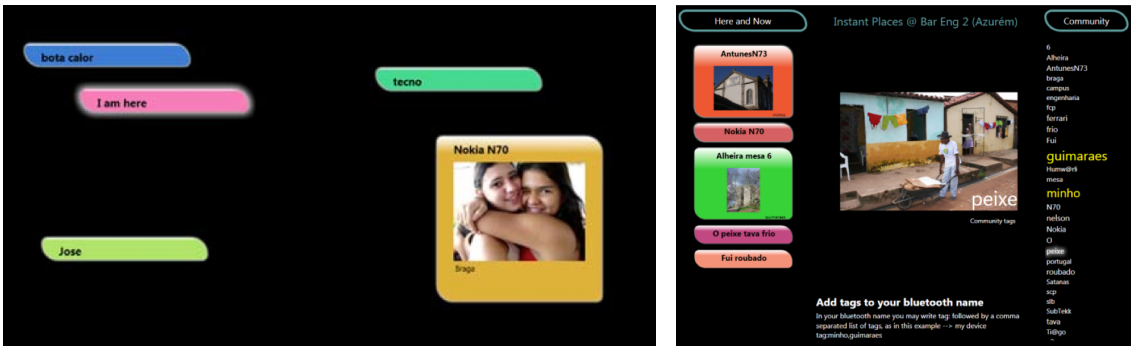


(a) Installation in the building. (b) Browsing photos.

Figure 2.13: The Hermes Photo display [Cheverst et al., 2005].

Bluetooth Instant Places

The Bluetooth Instant Places display system [José et al., 2008] was designed for shared and communal use in public and semi-public settings, using Bluetooth presence and Bluetooth naming as interaction techniques. The system was initially developed to study the suitability of those interaction techniques and the type of social practices that would emerge from their use in a real setting.



(a) Visualisation A. (b) Visualisation B.

Figure 2.14: The Instant Places display [José et al., 2008].

The system uses periodic Bluetooth scanning to generate a flow of presence information (based on the Bluetooth Media Access Control (MAC) address) and introduces a simple Bluetooth naming mechanism that can be used for explicit interaction with the display. The system is able to recognize specific words in the Bluetooth device name that trigger specific behaviour from the display. Users can define a Flickr username by including the string "flk:" followed by the Flickr username, for example: "my device flk:JohnDoe". Users can also associate keywords with their identity by using the "tag:" command, for example: "my device tag:football,fcporto".

Two visualisations where implemented. The first (see Figure 2.14a), displays information about currently present Bluetooth identities. Each identity is displayed as a coloured icon (the colour is generated when the identity is first seen by the system and re-used in subsequent sightings of that identity). When identities remain present for some time, a glow effect is applied to the icon, providing a sense of which identities have just arrived and which ones have been there for a while. For identities with commands in the Bluetooth name, the icon also shows a photo obtained from Flickr, using the command as a seed. The second visualisation (see Figure 2.14b) maintains the identity visualisation of the first, but introduces tag clouds. Identities are represented in the same manner, but occupying only the left-hand side of the display. The rest of the display is used to represent the current tag cloud. Tags are extracted from all words seen in the Bluetooth names, but the system gives a greater emphasis to the words found in the "tag:" command. In order to achieve a balance between an historic view of the place and a more dynamic and responsive system, tags have a popularity property. New tags are given a high popularity but it is decreased in every scanning. The system displays the 25 most popular tags and also uses them to seed periodic searches on Flickr to obtain a photo that is also shown in the centre of the display.

Instant Places

In a more recent version, Instant Places became a "*screen media system that enables people to manage the projection of their identity in public displays. With Instant Places, people can have an identity representation that allows them to explicitly and systematically manage their presence in public displays*" [José et al., 2012].

In Instant Places, people can publish two types of content: pin badges and posters. A pin badge refers to a particular institution, cause, campaign, sports teams, artist or brand that people may identify with. A pin badge is composed by a name, a set of tags, and a set of content sources from which it should be possible to generate screen content, e.g. a YouTube channel, a Flickr photo collection or a blog feed. On the Instant Places web page, users can associate pin badges with their profile. When they later check-in to a place, the information associated with the pin badge can be used to characterise the place and also as a source of content. Applications can query an Instant Places service about which pins are associated with the present users to generate situated content. Figure 2.15 shows an example of an application that shows content related to football club badges. Posters are media items that can be created by individual users, and distributed at specific locations. With posters people can create and publish content they consider relevant for the places they visit (for example, promoting an event, a cause, or an art creation). Posters are authored at the Instant Places web page by uploading an image, a short description, and a schedule for its availability for display.

Using the Instant Places mobile application, people can check-in and distribute posters to local screens when visiting a place. Additionally, the mobile application allows users to see the list of nearby places, and consume content associated with a place such as the list of presences and an activity stream.

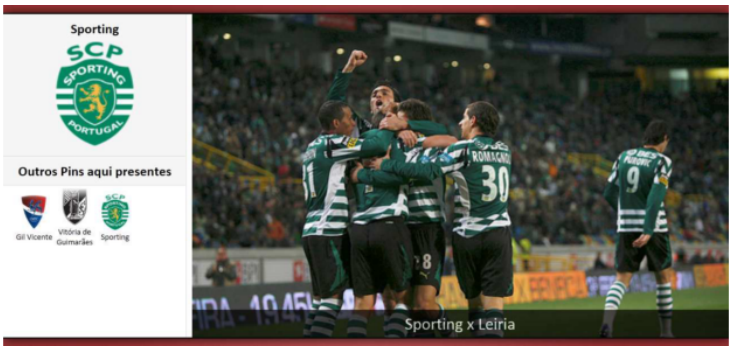


Figure 2.15: An Instant Places application driven by pin badges [José et al., 2012].

e-Campus Bluetooth

"*The e-Campus system is a network of public displays on the campus of Lancaster University in the UK designed to serve a dual role: as an infrastructure and testbed for local researchers and artists, and as a device for improving the experience of students, visitors and staff on campus*" [Davies et al., 2009, p. 153]. e-Campus is based on a software infrastructure that provides services for scheduling content on the existing public displays using a constraints based scheduling mechanism that allows content providers to specify when and on what displays their content should be shown.

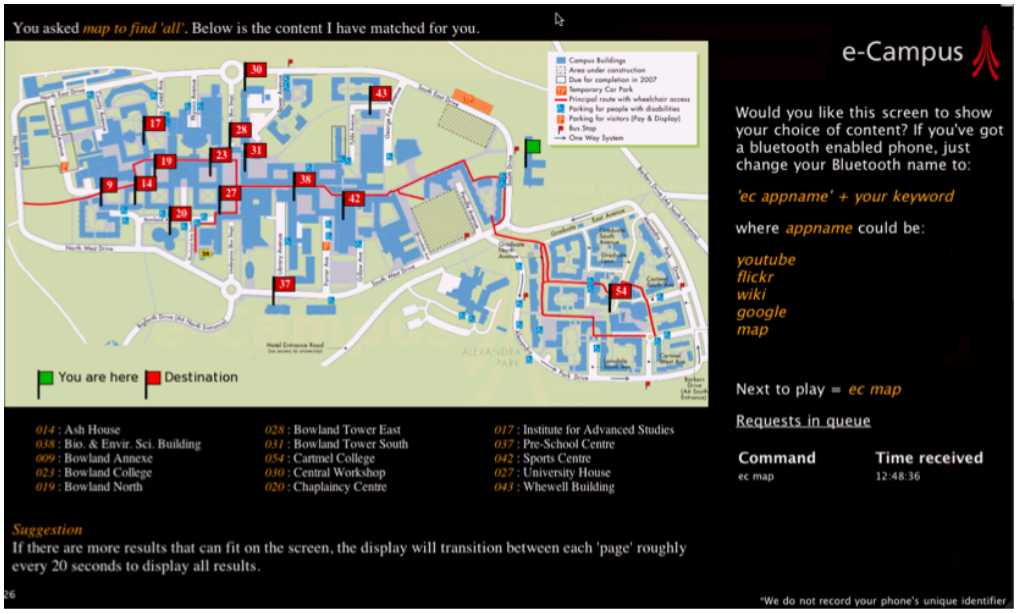


Figure 2.16: The e-Campus display, showing the map service [Davies et al., 2009].

In one particular project, Bluetooth device names were used as the interaction mechanism with the public displays. Bluetooth scanners on each display continually discover Bluetooth devices in the vicinity and send those sightings information to the content scheduler. The scheduler parses the Bluetooth names for valid commands and schedules services to display content, based on the commands detected. The commands to interact with the system take the form `ec <service_name> <params>`. The word "ec" is just a way to indicate that the Bluetooth name is an e-Campus

command, "service name" is one of the various services that were implemented (map – to show a map of the campus; flickr – to show Flickr photos; youtube – to show Youtube videos; google – to show search results; tiny – for generic web access; and juke – to play songs), and "params" are parameters that each service interprets in its own way (most are keywords used for searching photos, videos, or web pages). The system keeps a list of commands received and processes them in order, giving a limited display time to each one. Figure 2.16 shows the display interface that gives instructions to users and shows the selected service.

JoeBlogg

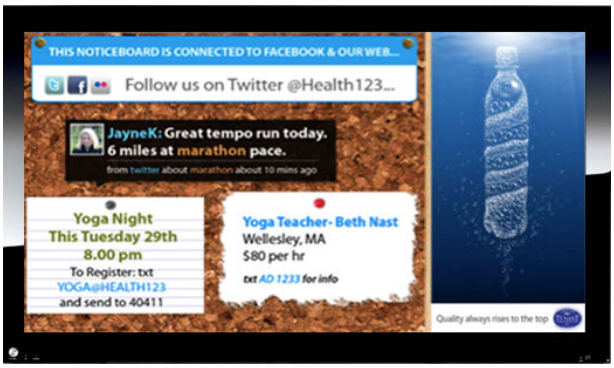
JoeBlogg is a "*socially situated public display for receiving MMS and SMS messages*" [Martin et al., 2006, p. 1079]. JoeBlogg is essentially an artistic project designed to give participants freedom to direct the use of the content of the system. The designers of the display were interested to see if a collective sense of narrative would emerge of the individual contributions to the display. The display was installed in the reception area of the Bartlett School of Architecture of the University College London, UK.

Figure 2.17: JoeBlogg display [Martin et al., 2006].

Essentially, the display receives photos and text sent through Multimedia Message Service (MMS) and displays and mixes them with portions of other pictures already on the display (see Figure 2.17). The display is divided in two rectangular regions that correspond to the left and right-hand sides of the screen. When a picture is received it is displayed in the left-hand side of the screen. If there was already a picture in that side, the older picture is moved to the right-hand side. Fragments of the displayed pictures are displayed in the smaller rectangular regions. These fragments are changed every 30 seconds to ensure that the display remains dynamic even if there are no interactions. If text is sent along with the MMS message it is also shown on two areas of the display (with a darker background).

LocaModa's displays

LocaModa[3] is a company that provides several, place-based social media display applications. Their applications use a variety of interaction mechanisms ranging from SMS to Twitter[4] messages. Figure 2.18 shows three of LocaModa's current applications.



(a) CommunityBoard.

(b) Jumbli.

(c) Wiffiti.

Figure 2.18: LocaModa's displays [LocaModa, 2010].

CommunityBoard, for example, "*allows venue managers, their customers and community members to post events, info, offers, and timely messages to venue screens via SMS Text, RSS, Twitter, and Facebook*" [LocaModa, 2010]. CommunityBoard allows users to place and reply to ads on the board via SMS. For example, the bottom left ad in Figure 2.18a, allows users to register for the yoga night by sending the SMS text "YOGA@HEALTH123" to the number 40411.

Jumbli is a word puzzle game that allows users to form words with the letters presented on the public display. The game is essentially a means to reduce the perceived waiting time for customers by engaging them in a game that also provides a reward program. In order to submit a word users can simply send an SMS message with the word to a pre-defined number (see Figure 2.18b). The same game is also a Facebook and iPhone application.

Wiffiti is an application that combines functionalities from several other applications, allowing users to send text via SMS, upload photos by email, and share Twitter

[3] `http://locamoda.com`
[4] `http://twitter.com`

messages on the public display (see Figure 2.18c).

AgentSalon

AgentSalon is an agent-based public display system which is aimed at facilitating "*face-to-face knowledge exchange and discussion by people having shared interests, in museums, schools, offices, academic conferences, etc.*" [Sumi and Mase, 2001, p. 393]. AgentSalon facilitates conversation between people by displaying their personal agents, using computer-animated characters, on the public display and having those characters interact with one another. The agents are able to carry a conversation using and sharing information from the respective user's personal profile, which includes personal information but also historic information about the exhibitions the user has attended and his comments and ratings for those exhibitions. Figure 2.19 shows two characters (and users) interacting through AgentSalon.

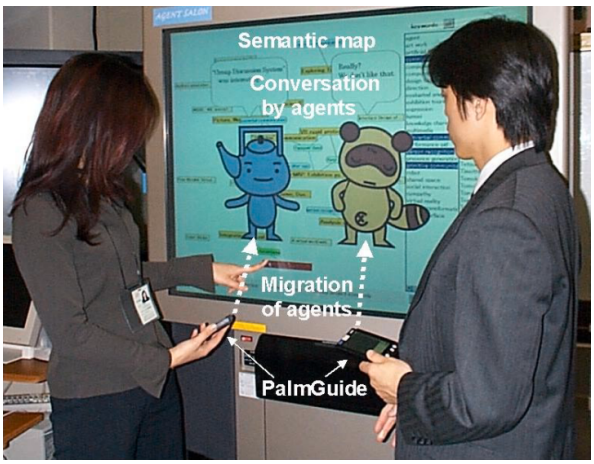


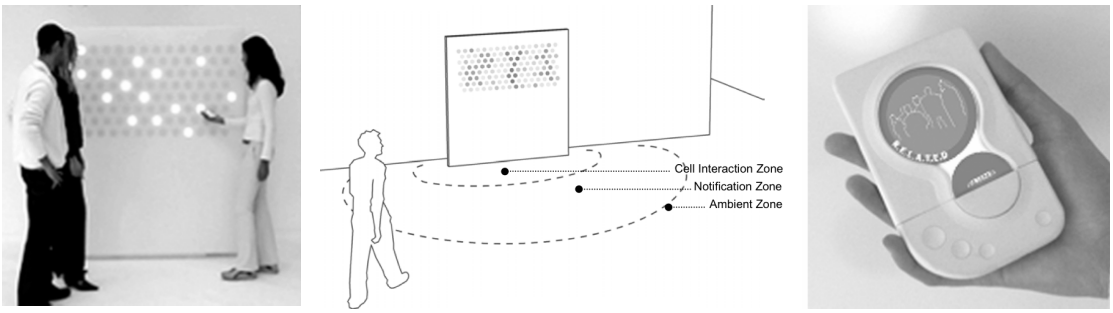
Figure 2.19: Users interacting through the AgentSalon display [Sumi and Mase, 2001].

Agents migrate to the display from the users' personal Palm OS devices when users connect the devices to the public display via infrared communication. The personal profile stored at the device migrates to the display with the agents. By exchanging profiles and searching for shared or (different) interests, agents are able to start a conversation, which is shown on the public display, allowing users to become aware of each other's interests. The display also shows a semantic map with the currently connected users' information about visited exhibits. The map is composed of text and icons that users can touch to display more detailed information about the exhibit.

Hello.Wall

The Hello.Wall [Streitz et al., 2003] is a public interactive ambient display for organisational information. It displays information in a very abstract way, using a matrix of 124 light cells that can be turned on or off, functioning also as informative art

(see Figure 2.20a). It's purpose it not only to display public ambient information, but also to support informal communication within the organization. It does this by combining a private information device – the ViewPort – with the public display. The ViewPort is used to interact with the display, by directly turning cells on or off, leaving messages to others, or to access private information stored in the display.



(a) Hello.Wall. (b) Interaction zones. (c) ViewPort device.

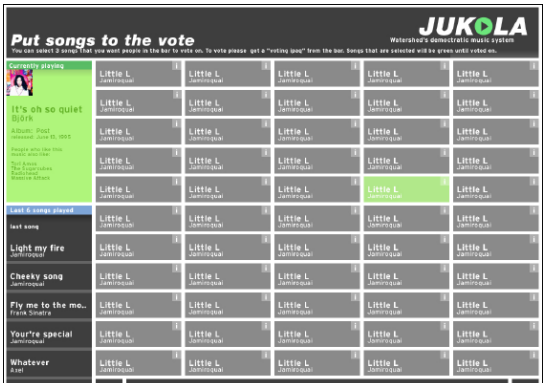
Figure 2.20: The Hello.Wall display [Streitz et al., 2003].

Hello.Wall has three interaction levels that correspond to the distance of people to the display (see Figure 2.20b): ambient zone – the farthest zone, where the display functions simply as an ambient display showing information that is independent of the presence of any particular person (for example, the number of people in the building, the level of activity, etc.); notification zone – an intermediate zone, where the display is capable of identifying the person (through the ViewPort) and display information targeted at that particular person in the public display or in the ViewPort; and cell interaction zone – where people can interact with the individual light cells of the public display, reading or writing information in a particular cell with the ViewPort. These different levels of interactivity are accomplished using long-range Radio-Frequency Identification (RFID) antennas to detect users in the notification zone and short-range RFID transponders in each light cell. The ViewPort is equipped with long range RFID transponder so that it can be identified and short range RFID reader in order to be able to read the id of the light cells. This way the Hello.Wall can detect users carrying the ViewPort in the notification zone and the ViewPort is able to identify each individual light cell in the cell interaction zone. The ViewPort is also equipped with a Wireless Local-Area Network (WLAN) adapter so that it can communicate with the Hello.Wall. The ViewPort is based on a Compaq Ipaq Pocket PC device physically modified to have a more appealing look (Figure 2.20c).

Jukola

Jukola [O'Hara et al., 2004] is an interactive MP3 jukebox system that allows a group of people to democratically choose the music that plays in a given place by voting. There are two interaction points in Jukola: a public display used to nominate songs, and a handheld device to vote for the next song. Jukola was deployed in the café bar of the Watershed - a local arts and digital media centre in Bristol, UK.

(a) The public display for nominating songs.



(b) The interface for nominating songs.

(c) The handheld device.

Figure 2.21: The Jukola display [O'Hara et al., 2004].

The public display (see Figure 2.21a) is a touch-screen display, located in a public part of the bar, that allows costumers to browse the music collection and nominate songs to be played in the bar (see Figure 2.21b). The handheld (iPAQ PDAs) devices are distributed to groups of customers so they can use them while sitting at a table (see Figure 2.21c). In each voting round (while the previous song is still playing) Jukola selects four candidate songs to be played next and displays them as voting choices in the handheld device. These four songs are drawn from the list of nominated songs as well as at random from the music collection. Using the handheld device, customers can register their vote. The device allows one vote for each voting round (but allows the vote to be changed). While the music collection is mainly defined by the owner of the bar, customers can also contribute to it by uploading MP3 files using a web interface. These files are, however, subject to vetting by the staff.

ContentCascade

ContentCascade, by Raj et al. [2004] is a mechanism to interact with public displays to download content from the display to a personal device. One intended scenario for this application is to allow users to download movie trailers from digital movie posters in theatres (see Figure 2.22).

ContentCascade uses a mobile application that communicates with the display via

Figure 2.22: The ContentCascade mechanism for content download [Raj et al., 2004].

Bluetooth and allows users to interact with the display's content both implicitly and explicit. In the implicit mode, ContentCascade detects that a user is standing near the display for some time and automatically starts downloading small pieces of meta-information about the content on the display. In the explicit mode users can browse the available content on the mobile application and select which content they wish to download.

In order to detect the presence of a nearby user and start transferring items automatically, ContentCascade takes advantage of the Bluetooth discovery mechanism, defining a minimum user hover time before starting the download: the system only starts downloading content if a user stands near the display for at least a preconfigured amount of time. To make an efficient use of bandwidth and storage space, ContentCascade's content items are described in various levels of meta-information. Depending on the user's interest and time near the display, ContentCascade may download different levels of information. For example, level 1 may consist simply of an URL and text description, level 2 may include a thumbnail image, and level 3 may include an additional video.

MobiLenin

MobiLenin [Scheible and Ojala, 2005] is a publicly controlled display that allows users to vote for the next music video to be displayed. MobiLenin was developed with an artistic perspective of giving the audience a new way to engage with the live show of a music artist by interacting with a multimedia piece.

The main idea was that each user could use his personal mobile phone to interact with a multi-track music video shown on a public display (Figure 2.23a). Interaction was a matter of voting for the next music video to be shown. To do this, a user would install and launch a mobile application developed for Mobilenin, and select the next video on a menu (Figure 2.23b). The mobile application connects to a server that counts the votes and also controls the public display that shows the voting statistics and then plays the most voted video (Figure 2.23c). To entice users

(a) Mobilenin in the public setting.

(b) Voting menu.

(c) Voting results.

Figure 2.23: The Mobilenin display [Scheible and Ojala, 2005].

to install and use the application, the system used a lottery mechanism that, with a certain probability, gives a prize (a beer or pizza) to the randomly selected "winner" of the current voting round.

Publix

Publix [Ventura et al., 2008] is a interactive advertisement billboard display system composed of a network of billboards and a mobile application. The purpose is to increase the advertisement's efficiency, hence driving more users to the billboards to capture their attention, by providing interactive advertisements.

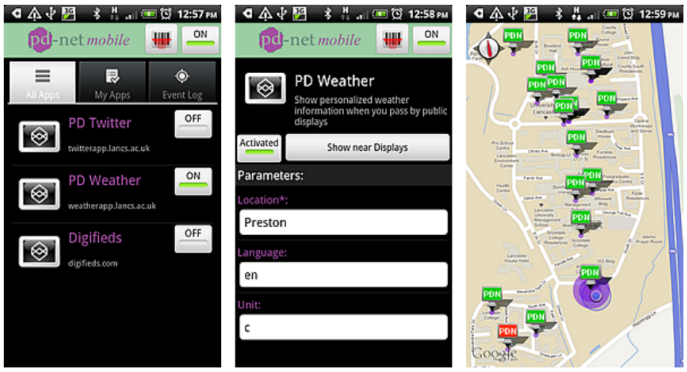
User interaction with the billboards takes place through the use of a mobile application that communicates with the display via Bluetooth (see Figure 2.24). Through this application, users can download ringtones and images, see current promotions, and play games. The billboards also perform proximity marketing by detecting nearby Bluetooth devices and pushing digital flyers (the system is able to recognize if someone already received or rejected the flyer in order not to send it again).

(a) User playing with the display. (b) PublixMobile application. (c) PublixMobile game menu.

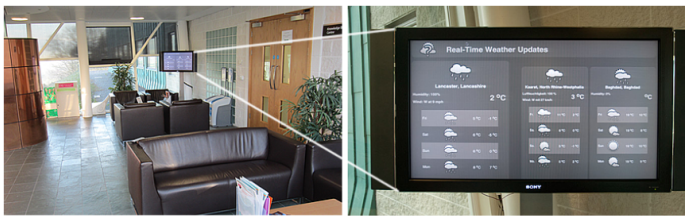
Figure 2.24: The Publix display system [Ventura et al., 2008].

Tacita

Tacita is a system to allow mobile users to express personalisation preferences to nearby public displays [Kubitza et al., 2013]. In contrast to other personalisation systems, Tacita avoids sharing user's location and personalisation data with the display infrastructure, instead using trusted application servers to make the personalisation requests. Using an Android mobile client, a user can discover nearby displays, determine the set of applications available, and trigger personalisation by filling in a set of key-value pairs that each application can define (see Figure 2.25a).



(a) Mobile application.



(b) Public display.

Figure 2.25: The Tacita public display system [Kubitza et al., 2013].

Using the client's knowledge of display locations and capabilities allows the user to see personalised content when within the proximity of a display without revealing their location (see Figure 2.25b). Sending the personalisation parameters directly

to the application to be shown prevents the display from building up a profile about any individual.

2.2.4 Id badge

In this section, we describe public display systems where users wear a personal badge that enables the display system to identify and react to their presence.

Intellibadge

The IntelliBadge [Cox et al., 2003] is an example of the use of displays to enhance the awareness of activities in a conference setting. This project was developed by the organising committee of the IEEE Supercomputing 2002 (SC2002) conference, with the objective of tracking the conference participants and display awareness information about the conference such as most active places and sessions. Conference attendees were initially asked to fill in personal and professional information in a web profile that included name, institution and professional interest categories (from a pre-defined set of categories). At the conference, participants were given RFID badges that were tracked throughout the conference site.

(a) Conference at a glance. (b) How does your conference grow. (c) Responsive map.

Figure 2.26: The Intellibadge display [Cox et al., 2003].

Three visualisations (see Figure 2.26) were created to show the flow of participants through the different conference areas and to show the most active conference sessions. The visualisations were pre-defined at design-time by the authors of the project and the main data that the displays used was simply the location data registered by the RFID readers deployed throughout the conference spaces. Users had no control over what was displayed.

The "conference at a glance" visualisation (see Figure 2.26a) displayed statistical information about the relative number of people from each interest category at each current event. The "how does your conference grow" (see Figure 2.26b) visualisation showed a more poetic view of the statistical data by using a virtual garden metaphor that used flowers as locations in the conference, coloured petals as interest categories and walking ants as rate of people walking through the tracked conference areas.

The "responsive map" visualisation (see Figure 2.26c) reacted more directly to users standing near the display by showing their place of origin on a map along with the name of their institution. This visualisation responded to people near each one of the three displays by sequentially displaying information about each person in a map. A live video feed from each display was also displayed and the video window borders were colour-coded in the same colour used in the map to show the association between the live video and the information on the map.

Proactive displays

In the Proactive Displays project [McDonald et al., 2008], the authors developed and evaluated a set of public display applications designed to augment and extend the social interactions that usually occur in an academic conference and are usually designated by "social networking". The authors developed a set of proactive displays capable of sensing and responding to the physical proximity of the participants. The conference participants were asked to fill in a personal profile on a web form, before attending the conference and these profiles were associated with RFID augmented conference badges. Throughout the conference venue, displays detected participants and showed content from their profiles.

(a) Autospeaker ID screenshot.

(b) Ticket to talk screenshot.

(c) Neighbourhood window screenshot.

Figure 2.27: The Proactive displays [McDonald et al., 2008].

Three different applications were developed and deployed in three different locations in the conference:

AutoSpeakerID – deployed in the session space of the conference, this application displays the name, affiliation and photo of the person asking a question during the question and answer period following a paper or panel presentation (see Figure 2.27a). In this case the display senses the presence of the participant that is in the microphone stand, not near the display itself.

Ticket2Talk – deployed behind the coffee tables, shows a theme (an image and a caption) that participants specified as being willing to talk about during the conference, thus providing a conversation starter for people during the coffee breaks. Content is shown only if the participant is near the display (see Figure 2.27b) and only for five seconds at a time. The bottom of the screen shows a list of thumbnails representing a queue of people that have been detected by the display and whose ticket to talk will be presented soon.

NeighborhoodWindow – deployed in the lounge area, shows keywords taken from the participants web pages and creates a network of connections between people and their interests. The result is that both shared and unique interests are naturally highlighted by the visual representation (see Figure 2.27c).

GroupCast

GroupCast [McCarthy et al., 2001] intended to be a semi-public peripheral display (see Figure 2.28) for fostering interactions and informal conversations between people in a workplace by displaying content that is of interest to at least one of the persons that were detected near the display (using an existing infrared badge infrastructure in the office environment). GroupCast builds on the user profiles created for another display system called UniCast but uses them differently: information is displayed only when the user to whom it refers to is detected near the display. This way it guarantees that what is displayed is relevant at least to one person and potentially creates an opportunity for person–person interaction about the displayed information.

Figure 2.28: The GroupCast display [McCarthy et al., 2001].

2.2.5 Device-free interaction

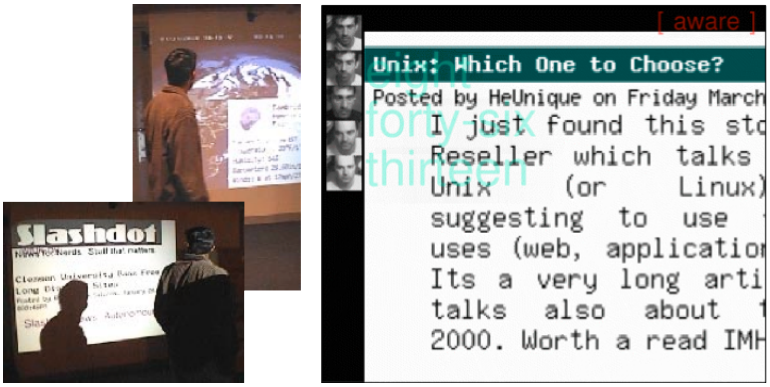
This section presents display systems in which interaction is accomplished by simply passing-by the display or standing in front of it and gesturing, without having to wear or use any device.

Aware Community Portals

MIT Media Lab's Aware Community Portals by Sawhney et al. [2001] is an example of a semi-public display intended to be used in a transitional area of a workplace as a shared peripheral information device. Because the community's interests were well known, the display was designed to use a popular technology-related news site

– Slashdot[5] – as the main content source, although it used several other types of content such as the current time, weather, cartoons and even MP3 audio files. The Aware Community Portals used movement, proximity and glance as interaction mechanisms (see Figure 2.29a):

> "*A phased approach first displays an 'information glance' when new information arrives. When a person is seen walking-by the space, a series of images are shown cycling through, depicting the recent stories in memory. If the person stops to glance at the display, a preview of the current story (news headlines or weather map) is shown for a short duration. If the person then continues to glance, the system assumes she wishes to browse the article in more detail, hence a sequence of related information is shown*" [Sawhney et al., 2001, p. 68].



(a) Glancing. (b) Faces timeline.

Figure 2.29: The Aware Community Portals [Sawhney et al., 2001].

The system is also able to detect faces and show them next to the articles, in a timeline view (see Figure 2.29b). This gives users an historic perspective over who read the article, raising their awareness about other people's interests and also giving an indication of which articles attracted more attention. The system used a web-camera and computer vision techniques to detect movement and faces. Movement is used to infer the proximity of people and face detection is used to infer a person's interest in the displayed content. The image that is being captured by the web-camera is shown back to users, and before faces are recorded, a rectangle around the face begins to grow, giving users the possibility to step back if they don't want their faces to be recorded.

Interactive public ambient displays

[Vogel and Balakrishnan, 2004] created a prototype public display system that uses physical proximity and explicit gestures for interaction. Although they prototyped it using a motion capture system that requires users to wear special markers, the concept was about implement free interaction with a public display.

[5] `http://slashdot.org`

Figure 2.30: Interactive public ambient display [Vogel and Balakrishnan, 2004].

The display uses a four-phase interaction model based on proximity and attention. Ambient display phase: the ambient display phase is a neutral state where the display shows only overall public information. Implicit interaction phase: this phase is triggered when a user passes by the display and appears "to be open to communication" [Vogel and Balakrishnan, 2004, p. 3]. In this phase the display shows an abstract representation of the user and notifies the user if there is an urgent information item that needs attention. Subtle interaction phase: in this phase the user gives a clear indication that he is interested in the display, by pausing and looking at it, for example. In this phase the display can show more detailed information. Personal interaction: the user interacts directly, touching the display to see more details about personal information.

Info-Jukebox

"*The Info-Jukebox is an information kiosk for browsing and searching multimedia archives in public spaces*" [Li et al., 2004, p. 384], that uses a electromagnetic field sensor to determine the coordinates on the screen at which a user is pointing. Objects on the display can be selected by pointing at them during a brief period. The display presents a list of images arranged in a matrix layout, which users can select to play the corresponding video (see Figure 2.31).

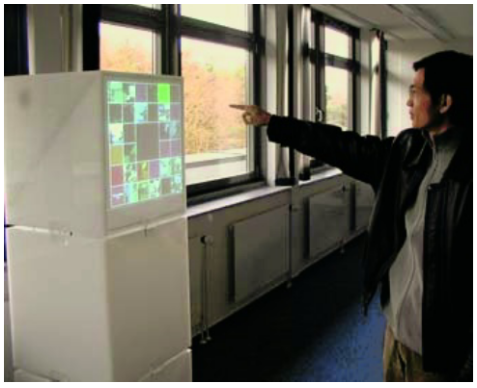


Figure 2.31: Info-jukebox public display [Li et al., 2004].

Proxemic Peddler

The Proxemic Peddler [Wang et al., 2012] is a prototype public display system that proposes a continuous proxemics interaction framework that takes into account users' identity, distance, and orientation relative to the public display. The Peddler Interaction Framework (see Figure 2.32) "*extends the Audience Funnel Framework [Michelis and Müller, 2011] to incorporate continuous proxemics measures including distance and orientation, additional states including digression and loss of interest, and the passer-by's interaction history, all with the goal of pursuing the AIDA [attract Attention, maintain Interest, create Desire, and lead customers to Action] strategy effectively*" [Wang et al., 2012, p. 3]

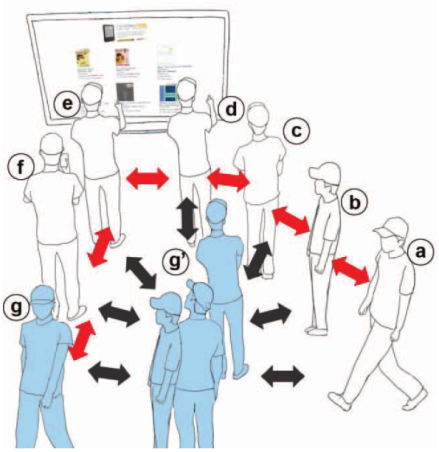


Figure 2.32: Proxemic peddler public display [Wang et al., 2012].

In the prototype, the authors used a Vicon[6] motion capture system that requires users to wear special detection markers, and the content was extracted from the Amazon website. The system uses various techniques to attract and maintain user's interest. A rapid animation of a flow of products is used to attract passers-by. If a user looks at the display, the animation slows down to allow better visibility. If the user moves closer, the display switches to show personalised content, which users can buy by touching the product on the display. If the display detects a possible loss of attention, it shakes the currently shown product to re-gain the user's attention. If the user moves away slowly, the display again tries to capture his attention by showing different products at a large size.

2.2.6 Analysis

How interaction influences content

Interactions with public display systems can affect the content displayed in essentially two ways: directly and indirectly. Directly means that the display system provides explicit interaction features so that users can control various aspects of the

[6] `http://www.vicon.com`

content presented by the display system. Interaction features such as submitting, browsing, removing, playing, graphically positioning, or showing detailed information about content, give users direct control over the content that the public display shows at a given moment. Most of the display systems surveyed provide users with various degrees of direct control over the content. For example, all display systems in the desktop-like and the touch-screen interaction categories, and many of the mobile device category allow users to submit content for the display to show; in fact, in most display systems in these categories, user-submitted content is the only source of information used by the display with the exception of Plasma Posters, Jukola, Hello Wall, and Locamoda's display systems, which also include other sources of information. Some display systems allow users to graphically control the positioning of a content item on the screen (Dynamo, Notification Collage, MessyBoard), browse content items (Plasma Posters, Outcast, Hermes Photo Display, Info-Jukebox, Interactive Public Ambient Displays), show detailed information about an item (AgentSalon), vote on content to appear (Jukola, MobiLenin), search (e-Campus Bluetooth), and play content (e-Campus Bluetooth, Dynamo).

Interactions can also indirectly affect the content selection or scheduling algorithms. The primary function of some features is to control the display's content, but the system gives them a secondary function, often unknown to users, of influencing the content. In CoCollage and Community Wall, the voting/rating, commenting, and check-in (in CoCollage only) features were used by the scheduling algorithm to decide which content items should be displayed next. In Hello Wall, Aware Community Portals, IntelliBadge, Proactive Displays, GroupCast, Interactive Public Ambient Displays, Proxemic Peddler, and Tacita, the user proximity to the display is used to trigger content (personal content in some cases where the display is able to identify the user). In Instant Places, user presence and expressed preferences are used to create a place profile in the form of a tag cloud which drives the content searches and consequently the displayed content items. These indirect uses of the interactions represent strategies that some display systems have used to autonomously adapt the content to their situation.

Table 2.1 shows the comparative analysis of the influence of interactions on the displayed content. It is apparent that very few display systems try to leverage o users' interactions to adapt the display's content. The most common approach is still to give users full control (and burden) over what is displayed, instead of using the information that indirectly results from interactions to learn about the display's social environment. For this to happen, we need a better understanding of how the various types of interactions can contribute to these content adaptation strategies. Our work addresses this issue by creating a framework of digital footprints that result from various types of interactions, and by mapping those footprints to various types of content adaptation processes.

Table 2.1: Influence of interaction on the display's content: comparative analysis.

Category	**D**isplay system	**I**nteraction influence on content
Desktop-like	Opinionizer	Direct (submit)
	Dynamo	Direct (submit, position, play)
	IM Here	Direct (submit, define content lifespan)

...continued.

Interaction category	**D**isplay system	**I**nteraction influence on content
	Notification Collage	Direct (submit, position content)
	MessyBoard	Direct (submit, remove, position content)
	CoCollage	Direct (submit); Indirect (votes, comments, check-ins influence scheduling)
Touch-screen	Community Wall	Direct (submit); Indirect (ratings and comments influence scheduling, presence freezes and displays more detailed content)
	BlueBoard	Direct (submit, remove, display)
	Plasma Posters	Direct (submit, browse)
	OutCast Digifieds	Direct (browse) Direct (create, browse, display)
Mobile device	Web Wall	Direct (submit)
	Jukola	Direct (submit, nominate, vote)
	Hermes Photo Display	Direct (submit, browse)
	AgentSalon	Direct (submit, show detailed information)
	Hello.Wall	Direct (submit); Implicit (presence triggers content)
	ContentCascade	No control over content.
	MobiLenin	Direct (vote on content to appear next)
	Publix	Direct (play)
	Locamoda's	Direct (submit)
	JoeBlogg	Direct (submit)
	Instant Places	Indirect (presence and expressed preferences determine what and when content is displayed)
	e-Campus Tacita	Direct (content searches/requests) Indirect (presence and expressed preferences determine what and when content is displayed)
Id badge	IntelliBadge	Indirect (proximity triggers personal content)
	Proactive displays	Indirect (proximity triggers personal content)
	GroupCast	Indirect (proximity triggers personal content)
Device-free	Aware Community Portals	Indirect (movement triggers generic content, continued presence triggers detailed information)
	Interactive public ambient displays	Direct (browse); Indirect (presence triggers notifications)
	Info Jukebox	Direct (browse)
	Proxemic Peddler	Indirect (presence triggers content)

Interaction mechanisms and features

The comparative analysis of the various display systems reveals a plethora of interaction mechanisms. The reviewed display systems used mechanisms such as keyboard-/mouse, custom desktop, web, and mobile applications, touch-screens, SMS/MMS, WAP, Bluetooth naming, Bluetooth OBEX, email, active badges, RFID badges, movement detection, face detection, gestures. Table 2.2 summarises the interaction mechanisms used in the surveyed public displays and the features they offer to users.

Most display systems are tightly coupled with very specific interaction mechanisms and have graphical interfaces built around these mechanisms. For example, the Aware Community Portals [Sawhney et al., 2001] relied on face detection for expressing and communicating interest in an article, showing the faces of readers in a timeline next to the article. Publix [Ventura et al., 2008] relies on a mobile device with Bluetooth capabilities, not only for proximity marketing, but also for communicating with the display for interaction. Hermes Photo Display [Cheverst et al., 2005] relies on Bluetooth OBEX for file transfers and even includes a printed poster near the display, instructing users on how to accomplish the Bluetooth OBEX file transfer process to send and receive photos; the e-Campus Bluetooth system [Davies et al., 2009] included written instructions on the public display graphical interface on how to interact by changing the Bluetooth name of the personal device; Joe-Blogg' [Martin et al., 2006] also included written instructions about sending MMS or SMS messages to display photos and messages on the public display.

Most display systems that employ more than one interaction mechanism, assign different mechanisms to different interactive features. Interaction mechanisms are usually tightly coupled with the interactive features that the display system supports. For example, on BlueBoard [Russell and Gossweiler, 2001] the touch-screen is used to share content with other users, the web interface is used to submit content to the display, and the id card is used to login. In Jukola [O'Hara et al., 2004] the web interface is used to submit content, the touch-screen to nominate songs, and the mobile application to vote on the next song.

The comparative analysis also reveals that the various interaction mechanisms have been used to support very different sets of interaction features and different interaction models. For example, the web interface in CoCollage McCarthy et al. [2009] was designed as an online mechanism to be used near the display itself: users would sit in a café with their laptops and see (some of) the actions made on the web interface immediately reflected on the public display. In Blueboard [Russell and Gossweiler, 2001], Plasma Posters [Churchill et al., 2004], Web Wall [Ferscha et al., 2002], and Jukola [O'Hara et al., 2004] the web interface was used more as an offline mechanism where users would submit content to view it later on the display. The Bluetooth naming mechanism used in e-Campus Bluetooth [Davies et al., 2009] was meant to be actively used to issue commands that would be immediately carried on by the display system. In Bluetooth Instant Places [José et al., 2008], the Bluetooth naming mechanism was meant to be used in a more passive way where users would define their preferences once and have the display system react to their presence in subsequent visits.

This analysis of the interaction mechanisms and features shows that the surveyed display systems have not taken advantage of abstractions for incorporating their interactive features. These display systems have been developed with specific interaction mechanisms in mind, which makes them hard to adapt to situations where other interaction mechanisms have to be used. Our work tries to address this by analysing and abstracting the interactions that can occur with public displays and proposing high-level controls that are independent of specific interaction mechanisms. These controls should provide public display applications with an abstraction layer that shields them from the low-level details of any interaction mechanism.

Table 2.2: Interaction mechanisms and features in public displays: comparative analysis.

Interaction mechanism	**D**isplay system	**I**nteractive features
Keyboard/ Mouse	Opinionizer	Submit content
	Dynamo	Submit content; browse; control presentation; share content; communicate;
	IM Here	Communication;
Desktop application	Notification Collage	Submit content;
	MessyBoard	Submit content;
Web interface	IM Here	Submit content;
	CoCollage	Submit content; check-in; vote; comment; message;
	Community Wall	Submit content
	BlueBoard	Submit content
	Plasma Posters	Submit content;
	Web Wall	Submit content;
	Jukola Digifieds	Submit content Submit content
Touch-screen	Jukola	Nominate;
	BlueBoard	Share content;
	OutCast	Browse content; communicate
	Community Wall	Rate; comment; forward content;
	Plasma Posters	Browse; comment; forward content;
	Hermes Photo Display	Browse
	AgentSalon	Show detailed information
	Interactive public ambient displays Digifieds	Browse; Browse; submit; forward content
Mobile app	Community Wall	Submit content
	AgentSalon	Submit content
	Hello.Wall	Submit content; receive content; detect users;
	Jukola	Vote;
	ContentCascade	Receive content;
	MobiLenin	Vote;

...continued.

Interaction mechanism	**D**isplay system	**I**nteractive features
	Publix	Receive content; play;
	Digifieds	Submit; receive
	Tacita	Express preferences;
WAP	Web Wall	Comment; submit content; vote;
SMS	Web Wall	Comment; submit content; vote;
	Locamoda's	Submit content; receive content;
MMS/SMS	JoeBlogg	Submit content;
Bluetooth OBEX	Hermes Photo Display	Submit content; receive content;
Bluetooth naming	Instant Places	Express preferences;
	e-Campus	Search/request content;
Email	Community Wall	Submit content
	Plasma Posters	Submit content;
	Web Wall	Comment; submit content; vote;
	Locamoda's	Submit content; receive content;
Id card	CoCollage	Check-in;
	BlueBoard	Login
Computer vision	Community Wall	Pause content
	Aware Community Portals	Trigger content;
Motion capture	Interactive public ambient displays	Trigger content; browse
	Proxemic Peddler	Trigger personalised content;
Electromagnetic field	Info Jukebox	Browse content;
Xerox scan	Community Wall	Submit content
Id badge	IntelliBadge	Trigger personal content; Be detected;
	Proactive displays	Trigger personal content;
	GroupCast	Trigger personal content;

2.3 Software Support for Application Development

This section describes several existing types of software that support the development of interactive applications, focusing on the abstractions they provide. We describe software support for different platforms and paradigms – desktop, web, context-awareness, and ubiquitous computing, which may all serve as inspiration for a solution for public display applications.

In order to facilitate the description, we have grouped the existing solutions into

three categories: widget based, dynamic user interface generation, and data-driven interaction. These categories were chosen because they reflect different approaches for abstracting user input and they result in different programming models and concerns.

2.3.1 Widget based

The widget based development model is perhaps the most used currently, not only for desktop development but also for web and mobile applications. Typically, this model emphasises the graphical look of an application and focuses on giving application developers, and users, a set of self-contained interaction objects that can be used to build an application through composition of individual interactive elements.

The X Toolkit (Xtk) [Swick and Ackerman, 1988; McCormack and Asente, 1988] was one of the first graphical user interface toolkits. Most modern user-interface toolkits follow its concepts[7] and structure so it is instructive to consider these concepts briefly.

A widget is the fundamental entity for the construction of graphical user interfaces. Xtk and all modern toolkits use object-oriented programming techniques to structure the available widgets and provide ways for programmers to develop new widgets based on existing ones. A widget is, thus, represented by a class and the toolkit provides a class hierarchy of widgets in the form of a tree (there is usually a root widget class that provides general bookkeeping functions needed by all widgets). Figure 2.33 shows the set of basic widgets for the creation of user interfaces in Xtk.

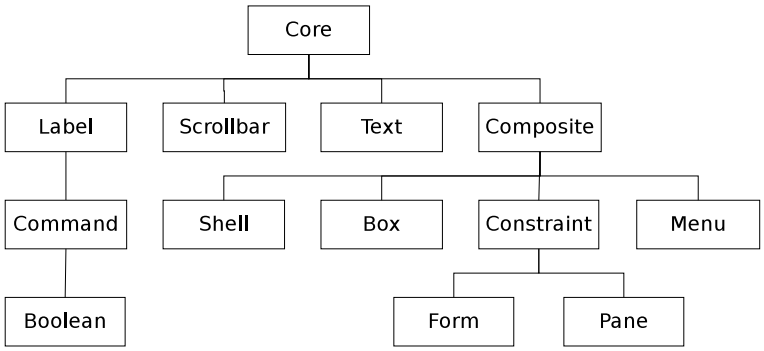
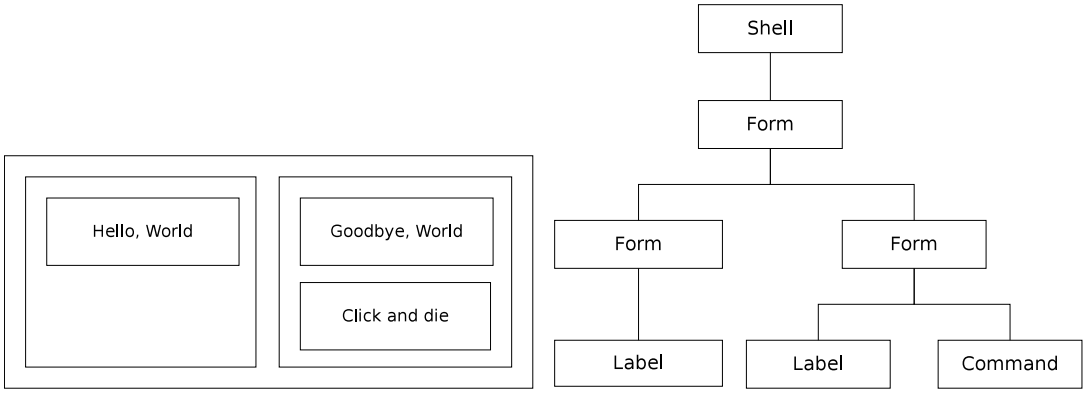


Figure 2.33: X Toolkit: the class tree for a subset of the available widgets.

Generally, widgets "*define input semantics and visual appearance*" [Swick and Ackerman, 1988, p. 222], and they are expected to be self-contained and be able to draw themselves when needed and handle all the input directed at them. Some widgets are just used as containers for other widgets to help lay them out in the graphical user interface. In the Xtk, these are called composite widgets (they are all subclasses of the Composite class).

[7] Swick and Ackerman [1988] were also the first to apply the term *widget* to user interface elements.

The user interface of an application can also be represented by a tree, but in this case, a tree of widget instances. The internal nodes of the user interface are composite widgets that serve only to contain other widgets and help manage their size and position. The leaf nodes are widgets meant to display information and accept and respond to user input. Figure 2.34 shows a sample application and the corresponding widget instance tree. The application has three leaf widgets: a Label with the text "Hello, World" in the left side of the window, another label with the text "Goodbye, World" in the right top side of the window and a button with the text "Click and die" in the right bottom side. To help create this layout, the application uses two forms: one that delimits the left side of the window and another that delimits the right side. To contain these two forms, another form is needed which is contained in the shell widget. (In Xtk, all applications need a special top-level widget, which is called Shell and can contain only a single widget.)



(a) Sample application with two labels (top) and one button (bottom).

(b) Widget instance tree.

Figure 2.34: X Toolkit: instance tree for a sample application.

Applications communicate with widgets by querying its state or invoking the widget's methods (to change the widget's state, set the data to be displayed, etc.). In the inverse direction, the main mechanism for widgets to communicate with the application is via callback methods. If an application is interested in receiving input notifications from a widget it registers a callback method that the widget will call when the corresponding input event is triggered. A widget may define several different events and applications are free to register callbacks to only a subset.

Applications may need a widget that is not provided by the toolkit and, so, programmers may need to develop their own widgets. Toolkits are designed with extensibility in mind so that programmers can easily extend an existing widget and create a new widget with new functionality. If the programmer only needs to constrain the functionality of an existing widget or modify it slightly, he can simply create a subclass of the existing widget's class and add new functionality or constrain the existing one. If a completely new widget is necessary, the programmer can always subclass the root widget (in many toolkits, a specific class is provided for this extension purpose). In other cases, the new widget can be built by composing existing, simpler, ones into a single widget. For this specific case, a "composite" or container widget can be used (or subclassed) to join together several widgets (some toolkits also provide specific composite widget classes for this purpose). Composition is suitable when

the widget has distinct visual components with specific input semantics and when components also make sense as individual widgets *per se*.

Whatever the means to create a new widget, from the application's point of view, the new widget can be used in the same way as the existing ones. Although toolkits usually provide a small set of basic widgets, there is really no limit to the complexity of a widget in terms of its visual appearance and input capabilities. As long as it provides a reusable component for the application and perhaps also useful for other applications, it can be encapsulated in a widget.

Other desktop and web toolkits

We can list various other toolkits that follow the same structure as the X Toolkit, for desktop, mobile, and web application development.

On the desktop, for example, the Java Abstract Window Toolkit (AWT) and Swing are very common toolkits for Java based applications. AWT is actually not just a widget toolkit but a combination of graphics, windowing and widgets toolkits. AWT provides just a thin layer of abstraction over the Operating System (OS) native user interface and the widget components are actually drawn by the OS, which makes Java applications look like native applications. Figure 2.35[8] shows some elements of the AWT toolkit. The Swing toolkit changes this by introducing its own widgets and drawing them independently of the OS (in reality, Swing provides an option to use the native look and feel), so that a Java application looks the same, regardless of the OS it's running on.

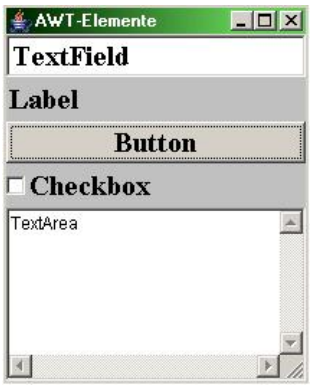


Figure 2.35: Java AWT toolkit: some components.

More recently, there have also appeared several libraries and toolkits for web application development. Although programming for the web is very different from programming for native desktop computer applications, the existing web toolkits have a very similar structure to those for desktop. jQuery [JQuery, 2013], Yahoo! User Interface (YUI) [Miraglia, 2006; Yahoo!, 2013], and Google Web Toolkit (GWT) [Google, 2011b] are widely used web toolkits for building rich, interactive, web applications taking advantage of techniques such Asynchronous JavaScript and

[8]Picture taken from `http://en.wikipedia.org/wiki/File:Easy_Java_AWT_example.jpg`.

XML (AJAX), and Dynamic HyperText Markup Language (HTML) (DHTML). They provide utilities for abstracting some differences in Javascript implementations between different browsers, easier manipulation to the HTML's Document Object Model (DOM) structure, custom events and graphical user interface widgets. jQuery and YUI are very similar in the way they are used for web application development: the programmer manually includes one or more Javascript file in the web page via the $< script >$ HTML element and can then take advantage of the respective toolkit's functions in the Javascript of the application, including instantiating widgets and adding them to the HTML DOM. GWT works in a very different manner: applications are written mostly in Java, not Javascript. GWT's libraries are based on the Java language, reusing a part of the core Java's libraries and introducing new ones specific to web development. Applications are written in Java but are then compiled into Javascript by the GWT compiler, which creates optimised versions of code for different browsers automatically. Another difference is that GWT is also prepared to work with Google's Appengine [Google, 2011a] – a server based application framework – which allow developers to more easily create full (server and client) web applications using the same language on both the server and client.

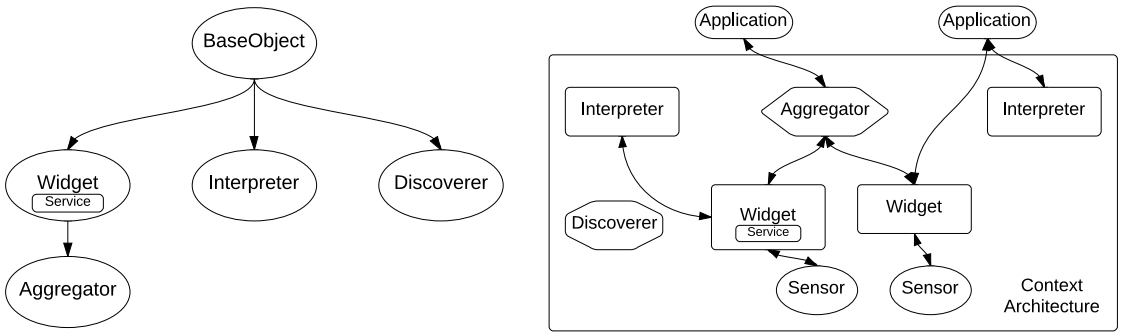
Widgets for context-aware applications

The widget abstraction concept has also been applied in application areas such as context-aware computing, which demonstrates their usefulness beyond the GUI interaction paradigm. In this section we describe how widgets are used in the Context Toolkit [Salber et al., 1999; Dey, 2000].

The Context Toolkit results from the recognised difficulties in developing context-aware applications, specifically in creating applications that are easy to develop and evolve and that can take advantage of a great variety of sensors and types of context. The design of the Context Toolkit was informed by the lessons learned from graphical user interfaces toolkits, namely the concept of widget. In the Context Toolkit, the main building block for applications is the context widget which collects information from the environment (through hardware or software-based sensors). Context widgets, just like traditional GUI ones, abstract low-level details of sensor input and processing, providing applications with ready-to-use high-level abstractions.

One example of a context widget is the IdentityPresence widget, which reports the arrival and departure of people at the location to which it is associated. An application can query the widget for the location where it is installed and the identity of the last person sensed. Applications can also request to be notified whenever a user arrives or leaves the location. Applications need not be concerned about the implementation details of this widget or the type of sensors it uses, only what kind of information it provides. Context widgets also provide reusable building blocks and can also be composed of simpler widgets. For example, a Meeting widget could be built on top of the IdentityPresence widget and determine that a meeting was taking place in a given location whenever two or more users where detected simultaneously.

Context widgets differ from GUI widgets because they must operate in a completely different architecture. Whereas GUI widgets run on a single, local computer, context widgets must run on a distributed system because sensors need to be physically distributed. Another major difference is that, contrary to GUI widgets, which execute only when the application that uses it is executing, context widgets need to collect environmental information continuously and, so, they are independent from applications.



(a) Component class hierarchy.

(b) Component interaction.

Figure 2.36: Context Toolkit conponents (adapted from [Dey, 2000]).

Besides context widgets, the Context Toolkit includes other components that applications can use (see Figure 2.36a). These components may be physically distributed and they can run on different computers. The toolkit provides peer-to-peer communication between components and facilities to discover components of interest to an application. Figure 2.36b shows a typical interaction between components and applications in the Context Toolkit. The main components of the toolkit are:

- BaseObject: provides the communication infrastructure needed by all components. This component is the root of the class hierarchy, so all other components inherit its functionality directly or indirectly. Components communicate using the HyperText Transfer Protocol (HTTP) protocol and data is encoded using an eXtensible Markup Language (XML) format. Applications instantiate this class to be able to communicate with the context infrastructure.

- Discoverer: provides resource discovery services. The Discoverer component allows applications to dynamically discover the existence and location (on the network) of other components. Applications specify what components they are interested in by specifying which attributes they require (location, username, in/out status, timestamp, etc.) and possibly defining some attributes' values (for example location="building X", if the application is interested in context information about building X only). The Discoverer knows about other components because they register themselves with a known Discoverer when they start, and they unregister when stopping. Applications may request notifications from a Discoverer to be notified, for example, when a specific component becomes online.

- Widget: acquires context information directly from sensors and provide applications with a uniform interface to access this information. Applications

can request to be notified about changes in the widget's context information or can simply query its current state to get the context information. Widgets also store the context information they acquire so applications can use historical context information. Widgets will typically run on the same computers to which the widgets' sensors are connected. Widgets can also provide services to applications, enabling them to actuate on the environment.

- Interpreter: performs inferences on context information or maps between different representations. For example, an Interpreter can translate Global Positioning System (GPS) data into addresses or combine context available in a conference room to determine if a meeting is occurring. Interpreters may be used to generate context information at a higher level than context widgets, although, in some cases, the two may be interchangeable.

- Aggregator: aggregates context about a particular entity. Aggregators collect context information about a particular person, place or object, from different widgets. An Aggregator can be seen as the union of a set of widgets, for a particular entity.

Analysis

Widget toolkits, such as the X Toolkit and derived ones for desktop, web, and mobile applications are very oriented towards graphical tasks – moving graphical objects, drawing, clicking on buttons and menus, dragging windows, etc. They abstract low-level mouse pointer and keyboard actions into higher-level actions such as button presses and text-entry. Also, they rely on very specific input devices: a pointing device and a keyboard for desktop and web toolkits, and a touch-enabled screen for mobile toolkits. In addition, they are targeted at single-user and local interaction environments where both the application and input devices are on the same computer system. Although some public display systems have successfully used these toolkits in their implementation, they did so for a very specific usage scenario. As a general-purpose interaction abstraction for public displays, these toolkits are not directly applicable but they still provide a source of abstraction and inspiration.

The Context-Aware Toolkit is obviously not suited for interaction with public display applications because it is targeted at context acquisition, not at interaction. However, it demonstrates that the concept of widget, widely familiar to most programmers, can also be applied to a completely different platform, maintaining its fundamental properties of abstraction.

Our work also leveraged on the familiarity of the widget concept as an abstraction for programming the user interface of an application. We took advantage of all the concepts and features we could from these toolkits, but public display applications have specificities that require a different approach. The toolkit developed in this thesis uses the concept of widget to represent high-level controls that abstract input from several sources and provide interaction events to public display applications. Unlike widgets for desktop programming, our widgets for public displays are not

dependent on local input devices such as mice and keyboards but can receive input from both local and remote devices. Additionally, they support multiple simultaneous users interacting with the same widget.

2.3.2 Dynamic user interface generation

The dynamic user interface generation approach is another possibility for providing interaction abstraction for applications and so it is important to analyse its main properties and variants. This approach works on a different level than widgets and both can even be used together.

The dynamic user interface generation approach is more inclined towards ubiquitous computing environments and smart spaces where there are services associated with places, which can be accessed through many different devices. Developers focus more on how to describe the service interface in an abstract manner than to create a specific graphical interface for a specific device. This approach favours multimodality because the user interface can be dynamically generated or fetched by a user's device, which may render it using different modalities.

In this approach, the application logic and user interface are distributed: the user interface runs in a controller device that connects remotely to the application logic through some kind of remote method invocation mechanism. Service developers focus on the implementation of the internal logic and on what functions the service should provide to the outside world. These functions represent the programmatic interface that the service exposes to other system components. The actual user interface runs on a controller device, which translates user input into remote method calls according to the service's interface. There are usually four main components in this distributed architecture:

Services services can be devices or applications that provide useful functions to end-users.

Controller devices controllers are the (usually mobile) devices that users will use to interact with the existing services.

Communication network controllers communicate with the service to invoke the necessary functions in response to user actions, through some communication network.

Service discovery controllers need to be able to discover the existing services in a given environment, so there is usually some kind of service discovery facility where services are registered and controllers can ask for existing services.

There are several ways to make the user interface appear in the controller device and allow a user to control the service. Whatever the concrete means, the end goal is always to facilitate the use of multiple, heterogeneous mobile devices, to interact with several services available in a given environment.

Downloadable user interface code

One approach for creating the user interface is to have the controller download the User Interface (UI) code from the service itself and then execute it locally. To support controllers with different capabilities and interaction modalities, services may have different user interface implementations and descriptions associated. Controllers wishing to access the service can inspect the associated user interfaces and choose the best fitting one.

This is the approach followed by the Jini Service UI [Venners, 2005], which is based on Jini's service architecture. In Jini, services announce themselves by registering in a lookup service available in the network. This registration involves placing a service proxy object for the service in the lookup service. This service proxy is a runnable Java object that clients use to interact with the service. Clients use the lookup service to find a service of interest and download the associated service proxy. To invoke a function on the service, clients call a local method on the proxy. The proxy object takes care of communicating over the network to the corresponding service, using whatever protocol the service developer chose. Jini Service UI extends the base Jini architecture and defines how services can attach different user interfaces to the service proxy and how clients can select and run an appropriate user interface for the device. This is accomplished by using UI descriptors, which include a set of fields (attributes – i.e., generic attributes to help search for suitable UIs; toolkit – e.g., AWT or Swing; and role – e.g. MainUi, AdminUI) that describe each user interface, and that clients can use to select the most appropriate one. The Jini Service UI architecture is centred on the Java programming language, but the same approach could be used for supporting other programming languages, even simultaneously.

Abstract user interface description

A different approach for showing the user interface in the controller device is to have a service interface description that controllers can read to dynamically generate a suitable user interface. The same service interface description may result in completely different user interfaces in different controller devices because controllers can generate the most appropriate user interface according to their own interaction modalities (graphical, speech, gesture, command line, etc.) and capabilities. The service interface description is usually done in an abstract form to guarantee device independence and interaction modality independence so that devices are able to generate a suitable user interface whatever the concrete type of device and interaction modality.

This approach has been realised in many systems before, with slight variations among them. For example, Roman et al. [2000] developed a device independent, XML based, language for representation of services and the respective dynamic interface generation for mobile devices. The service description language allows the definition of a list of methods with parameters, which can be invoked by the client. The service description also includes a description of GUI elements that should be used in the user interface to represent the parameters and actions that trigger

the invocation of methods. Clients obtain the service description and employ a eXtensible Stylesheet Language (XSL) transformation particular to the device to generate the HTML user interface, which is then presented in a native browser (the client includes a custom HTTP proxy that intercepts HTTP requests and transforms them to service calls and responses when necessary).

XWeb [Olsen et al., 2000] is another system that uses dynamic user interface generation based on an abstract service description but, in this case, with a much higher focus on data. XWeb provides an architecture similar to the World Wide Web (composed of XWeb servers and clients that communicate through the XWeb Transport Protocol – XTP), but with specific mechanisms for interaction. XWeb data is represented in XML and servers view it as a tree of objects (tags) with attributes. Clients can access and edit parts of the data tree with XWeb URLs. A service exposes its data through a Data URL so that clients can manipulate it but, in order to provide a user interface, services also specify a View URL that points to an XView. The XView specifies the interaction that can be accomplished with the data. Basically, an XView defines the possible data types and transformations from internal representations to user-friendly representations. In XWeb these are called interactors and there are atomic interactors (numbers, dates, times, enumerations, text and links) and aggregate interactors (groups and lists). Interactors can be decorated with informational resources such as icons and text descriptions. An XWeb client uses the XView to generate a user interface appropriate to the interaction modality of the device. Because XViews are independent of any concrete interaction modality, devices are able to generate user interfaces for graphical and speech-based interaction.

Another system that resorts to dynamic user interface generation is the Personal Universal Controller (PUC) [Nichols et al., 2002]. PUC uses a peer-to-peer communication architecture between client devices and services. PUC uses a service specification language, based on XML, which models data (state variables) and actions (commands). A service is modelled as a group tree of elements (variables and commands); this structure allows service developers to group similar elements together providing cues for the interface generators. The specification language also allows the definition of dependencies between elements. For example, it is possible to specify that a command or state will be disabled, depending on the values of other state variables. The specification language also allows the inclusion of label information in the form of dictionaries; a label dictionary may have different types of labels that can be used by different devices (text labels of different sizes for graphical devices with different screen sizes, speech labels for speech devices). The combination of the group tree and dependencies is used by the graphical user interface generator to organize the interface components into logical groupings or by a speech interface generator to allow a user to navigate through the groups and activate commands or change state variables.

Hybrid approaches

One of the problems of downloadable user interfaces is that it still requires developers to fully design the user interface to their services. The problem is aggravated by the existence of various heterogeneous devices, with different interaction modalities, which must be explicitly considered by the developers. With abstract user interface descriptions there is no development effort for the user interface, but the resulting user interface can be unappealing. To address the shortcomings of these two approaches some systems adopt a hybrid solution: custom-designed user interfaces can be associated with services for some types of devices but if a suitable user interface for a particular device does not exist, the device can dynamically generate a user interface from the service description.

[Hodes and Katz, 1999], for example, propose an XML based Interface Specification Language (ISL) that allows the specification of methods that can be invoked on the service and also of user interfaces for that service, available to be downloaded and executed in the device. ISL includes a <ui> tag to specify a user interface for the service, which can be the name of a component that the device somehow knows about or a network address where the component can be found; the user interface specification also includes a language parameter that indicates in what programming language the component is written. If no user interface is specified for a service, or if no suitable one for that particular device is found, the device can use the ISL to dynamically create a user interface that allows users to interact with the exposed methods of the service.

iCrafter [Ponnekanti et al., 2001] also uses a hybrid approach but introduces one main difference to previous systems: the interface generation is done in the infrastructure, not in the controller device. In iCrafter, services register in an Interface Manager and send it a service description in an XML-based language called Service Description Language (SDL) – which lists the operations supported by the service, in a similar way to ISL. Clients obtain a list of available services from the Interface Manager and can ask for the user interface of a specific service (or a combination of services). When asked for a user interface, the Interface Manager will search for a suitable interface generator: it first searches for a generator for that specific service, then for a generator for that service interface, and finally for the service-independent generator. This allows the creation of custom user interfaces for a service, if the developer chooses to, but guarantees that a suitable user interface can always be presented to the user. The interface generator uses a template to generate code in a user interface language supported by the controller device (iCrafter supports HTML, VoiceXML [VoiceXML, n.a.], SUIML [Montiel-Hernandez and Cuayahuitl, 2004] and MoDAL [MoDAL, 2011]), so controller devices are assumed to be capable of running a user interface interpreter that can then render the received user interface code.

There has also been some industry effort in the development of standards for universal remote consoles that follow a hybrid approach. The Universal Remote Console (URC) [Vanderheiden and Zimmermann, 2005] is such an example and it consists of a family of industry standards aimed primarily at the remote control of

home appliances. The URC defines a way for appliances to describe their functions through a user interface socket and to provide an User Interface Implementation Description (UIID). A UIID can be an abstract or a very concrete description of the user interface. Concrete UIIDs can assume particular controller devices and provide a very customized user interface; generic UIIDs don't make any assumptions about the controllers. The URC standards assume that controllers are URC enabled (able to communicate to appliances and interpret their interface socket description and UIIDs), but a middleware system called Universal Control HUB (UCH) exists to provide URC connection points for non-URC devices, translating between the devices' specific protocols, such as Scalable Vector Graphics (SVG)/HTTP, DHTML/HTTP, Flash, VoiceXML, etc., and URC.

Analysis

The dynamic user interface generation approach targets smart mobile devices that have the capability of running or generating user interfaces on-the-fly. It does not consider interactions that occur through other interaction mechanisms such SMS, MMS, email, touch-screens, QR codes, and many others, which have been used in public display systems. However, considering the number of public display systems that use mobile applications and the foreseeable usage that these applications will have in the future, this approach may provide an important part of the solution to the problem of public display interaction. In particular, the abstract user interface description and the hybrid approaches could allow developers to focus on the public display interface, while automatically providing mobile device graphical interfaces. For developers without the resources to develop custom mobile applications, a solution in which the display system automatically generates a mobile interface for a display application would be valuable.

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

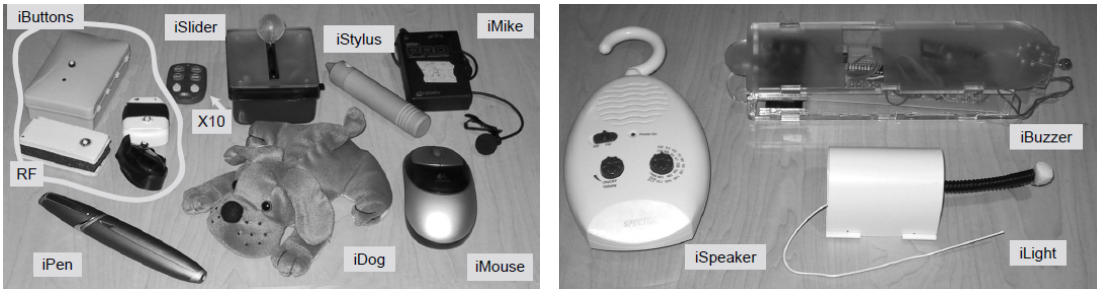
2.3.3 Data-driven abstractions

The data-driven approach is also geared towards ubiquitous computing environments, but its focus is more on the case where a variety of dumb input devices can be used to interact with a given application. This approach focuses more on the data types generated by input devices and the data types that the application supports rather than on the graphical user interface.

iStuff

The iStuff toolkit [Ballagas et al., 2003] is a toolkit of wireless input/output devices with software proxies and a dynamic routing software component that re-targets input events to applications, in run-time. iStuff was created to allow the use of different physical input and output devices (see Figure 2.37) to interact with room-sized environments consisting of displays of different sizes, and to facilitate the development of applications that support input from different physical devices:

> "*Our domain is explicit interaction with a room-sized environment consisting of displays of many sizes, plus support for wireless technology of various types, integrated using a common middleware. Our goal is to allow multiple, colocated users to fluidly interact with any of the displays and applications in augmented environments such as the Stanford iRoom, using for input and output any devices conveniently at hand.*" [Ballagas et al., 2003, p. 537].



(a) Input. (b) Output.

Figure 2.37: iStuff devices [Ballagas et al., 2003].

iStuff architecture is composed of four main components: iStuff components, the PatchPanel, the EventHeap [Johanson and Fox, 2002], and applications (see Figure 2.38):

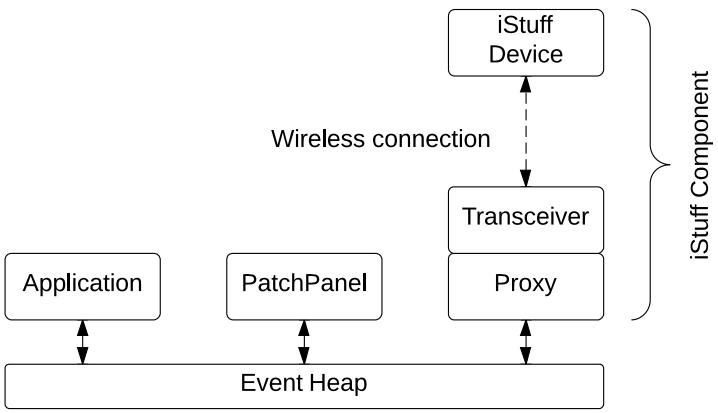


Figure 2.38: iStuff architecture (adapted from [Ballagas et al., 2003]).

iStuff component An iStuff component includes the physical input device (iStuff device), the appropriate transceiver that receives the wireless signals from the device, and the software proxy that translates the input signals into input events.

EventHeap An event-based communication infrastructure. Events are messages or tuples that contain a type and an optional number of fields in the form of key-value pairs. Producers post events and consumers register their interest in being notified when certain events are posted (by specifying the event type or other matching criteria based on the content of the event's fields).

PatchPanel A software component that translates events from iStuff components into application events. The PatchPanel listens for events that it has been instructed to translate and, whenever a matching event is posted, it generates a new translated event (leaving the original event untouched).

Application A workspace application that needs input from physical devices. Applications define their own high-level events.

The PatchPanel is the most important component, from the input handling perspective, because it introduces an abstraction layer between the physical devices and the environment's applications. Both input devices and applications define their own static, non-compatible, event types, but the PatchPanel connects these two separate components allowing them to communicate. It also provides a flexible way to connect any input device to any application (provided their event types can be translated). An example from Ballagas et al. [2003] illustrates this. An application called iPong (modeled after the classic Pong game) defines an event named "MovePaddle". To control this application with an iSlider device (a physical slider device) the PatchPanel can be instructed to map "iSlider" events into "MovePaddle" events. If one wishes to use a different device to control the iPong, for example a wand device, the PatchPanel could be instructed to now map the "Wand" events into "MovePaddle" events. No modification would be necessary in the iPong application. This, of course, also allows a single physical device to be used to control different applications.

The PatchPanel itself is made of two software components: a PatchPanel intermediary and a PatchPanel GUI. The intermediary is an EventHeap client that does the event translation based on its current configuration. The intermediary's configuration itself is updated by sending events to the EventHeap which means that any program can dynamically change the current mappings. Event translations can be as simple as changing the event type or as complex as creating new fields and changing field values by applying arithmetic expressions to the values of the received event. The PatchPanel GUI is a graphical tool for manually creating event translations. This tool sends standard PatchPanel configuration events so it is independent from the intermediary.

I/O Modules

[Paek et al., 2004] proposed a platform for supporting input from multiple devices. The platform follows a modular architecture composed of input devices that send data to I/O modules, each of which is specifically designed to understand data from a single mode of communication. Examples of modes of communication include email, SMS, instant messaging, keyboard, and mouse/joystick. The I/O module parses the received data into discrete message units that are passed to the translation module. The translation modules converts the message units into commands that share a common syntax that are passed to an application module, that decides how to use and react to the input data. Figure 2.39 shows the I/O modules architecture.

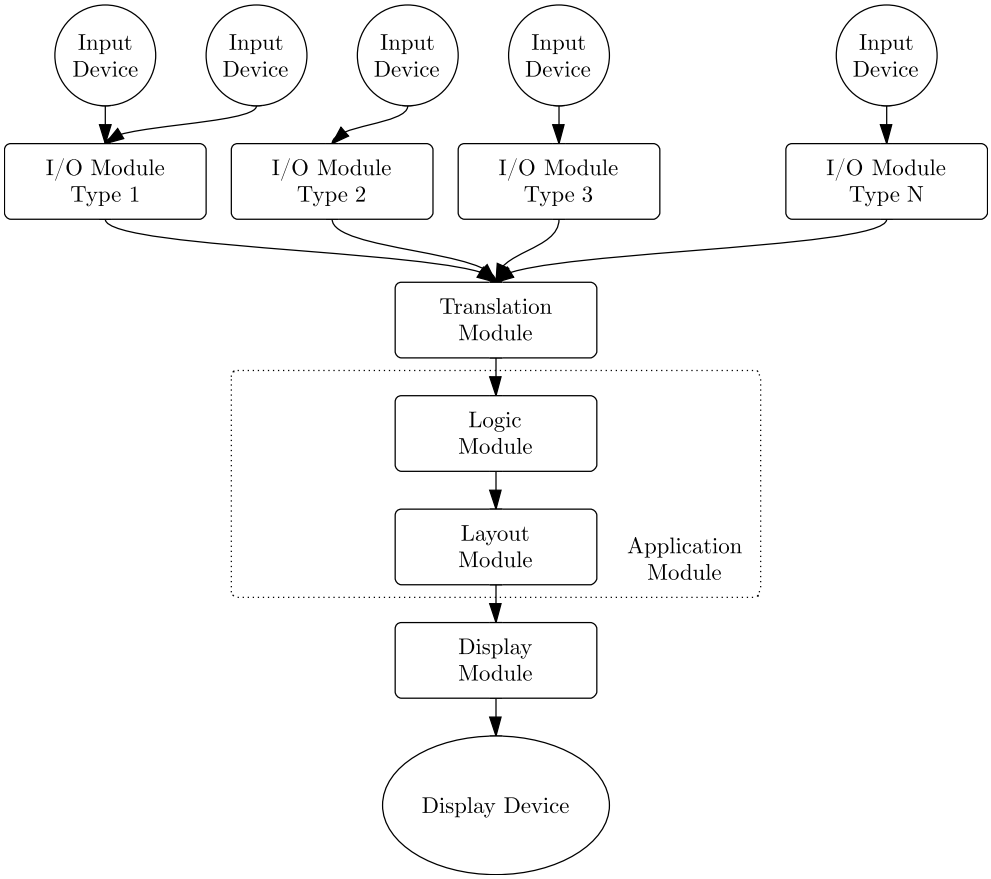


Figure 2.39: I/O Modules input platform (adapted from [Paek et al., 2004]).

Analysis

The data-driven abstractions represent an intermediate abstraction level between low-level device input and high-level application controls. They don't aim at providing high-level controls as, for example, the widget toolkits for desktop environments do. However, a solution such as the I/O Modules by Paek et al. [2004] could be used to support command-style interaction from disparate input mechanisms such SMS, email, Bluetooth naming, instant messaging, in an uniform way.

Our solution for an interaction toolkit takes advantage of an architecture similar to

I/O Modules, building higher-level abstractions on top of it. Our toolkit provides a command language for text-based input devices that allows users to address specific widgets in the public display application.

2.4 Conclusion

The examples of concrete public display systems described in section 2.2 – Interaction in Public Display Systems – attest the diversity of interactive public displays. We have described public displays with different application areas such as entertainment, advertising, collaborative work, and social interaction; with support for very different interaction mechanisms such as SMS, Bluetooth, touch, gestures; and with different usage settings, such as schools, conferences, museums, cafés, streets. These examples also demonstrate the heterogeneity of interaction mechanisms and models. A user accustomed to interacting with one display system would have difficulty in knowing how to interact with the next one. Implicitly, these examples show that these are ad-hoc systems where the developers had to build their own solution to provide interaction, and think about and implement all low-level aspects of the interaction. Table 2.1 also shows that the most common approach to content is still to give users full control over what is displayed.

It is also clear from the examples presented in section 2.3 – Software Support for Application Development – that there are currently many solutions to provide high-level abstractions for desktop systems, but also abstractions for other less standard computing platforms. In this work, we have looked at these existing solutions, their focus, and their programming models, and used them as a starting point to create a solution that answers the requirements of interactive public display applications.

Our work contributes to both these lines of action: it helps designers of situated public displays understand the kind of information that can explicitly or implicitly be gathered from interactions and how that information can drive content adaptation models; and it provides developers with software interaction abstractions that allow them to focus on high-level interactive features, while supporting various kinds of interaction mechanisms.

Chapter 3

Requirements for Interaction Abstractions for Public Displays

Contents

3.1 Introduction

Public display interaction occurs in a setting that is very different from desktop interaction – the one most people are familiar with. It is thus necessary to characterise the main aspects of the interaction environment for public displays, and the requirements they impose on a possible interaction abstraction system. This chapter starts by analysing some fundamental assumptions and concepts about the interaction environment for public displays. It then looks into the implications of these assumptions and how they can represent requirements for interaction abstractions for public displays.

3.2 Assumptions

3.2.1 Interaction environment

Terrenghi et al. [2009] have used the concept of "*ecosystem of displays*" to refer to "*the complete system of displays, people and the space in which they are placed*" [p. 583] and characterised these ecosystems in terms of their scale and nature of the social interaction. The scale of the ecosystem depends on, and is usually greater than, the size of the largest display and can be an inch, foot, yard, perch, or chain size ecosystem.[1] The nature of the social interaction can be one-one, one-few, few-few, one/few-many, or many-many. One-one interaction is the simplest case where, for example, two people exchange files; one-few interaction is when there is one presenter using a personal computer to show slides on a large display or projection, for example; few-few interaction happens when groups of people collaborate; one/few-many happens with public performances where one person or a very small group controls what a large audience is seeing on the display; and many-many interaction when there is not a single person controlling the display but instead many people can participate simultaneously.

Our focus is on yard or bigger size multi-person-display ecosystems for many-many interaction, composed of displays of various sizes. Additionally, our focus is on coupled or loosely coupled display ecosystems, where the various displays are used together to accomplish some interaction task. The perch/yard size public displays can function as the main information outlets, visible to everybody (as in Dynamo – page 19); smaller, yard/foot size public displays can be dedicated to particular uses such as being used solely for input (as in the touch display on the wall in the Jukola jukebox system); and small foot/inch size displays are most likely personal devices used for input and output (as in Hermes for downloading and uploading pictures, MobiLenin for voting, Instant Places for checking-in and publishing posters, etc.)

[1] Terrenghi et al. [2009] extend Weiser's classification of inch (2.54 centimetres), foot (about 30 centimetres), and yard (a bit more than 91 centimetres) displays with two other imperial measurements: the perch, which is 5.5 yards (a bit more than 5 meters); and the chain, which is 22 yards (a bit more than 20 meters).

Such an environment can be found in small spaces likes cafés and waiting rooms but also on larger spaces such as university departments or even an entire campus. Although there can be many kinds of social interaction in these spaces, we are focusing essentially on many-many interactions where there is not a single person or small group that "owns" the information of a display. Public displays may be privately owned, but the owner chooses to share the display, even if with some restrictions, with the aim of creating a shared information space where everyone can have the same opportunities to interact and where the different displays offer different views to the information or different possibilities to interact with it. Although the different sized displays afford different types of interaction, they can function in an integrated way in the ecosystem. To further describe the interaction environment we envision, consider the following scenarios:

Train station *John has just arrived at the train station and bought a ticket to his hometown – he's been away on military service. His train is due in 25 minutes so he sits down in the waiting room. A display is showing useful information such as train and bus timetables and also advertisements. One advertisement seems to have been made on purpose for him: it's from a flower shop offering a digital discount coupon to a Red Passion Bouquet – the perfect gift to offer his girlfriend. To get the coupon he needs to download it to his phone. The instructions on the display describe various alternatives to download the coupon. One of the alternatives is to simply access a website, but his mobile phone is old and does not have internet access necessary for this. He also does not have credit on his mobile carrier account to send an SMS to receive the coupon. Fortunately, as the display informs, there are alternative touch displays spread around the station where he can get the coupon via Bluetooth. John locates the nearest one and approaches it. The display shows a list of items and he scans it until he sees an "Adverts" item. He navigates to the flower shop name, and the screen says he can download the coupon, and alternatives to do it. He chooses the Bluetooth option, and follows the on-screen instructions. Searching the phone settings he is finally able to make it discoverable and receive the coupon. He still has 15 minutes before his train arrives – plenty of time to buy his gift!*

Multiple interaction mechanisms.

Displays are used together to accomplish a task

University news *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!*

Recognizable affordances.

Recognizable feedback.

Coffee shop *Sarah and George took a break from work to grab a snack at the coffee 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 snacking 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 application 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 on his smartphone thanking and telling him that he can check the result 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!*

Asynchronous interactions.

Waiting time *George is at the waiting room of the dental clinic for his 5 o'clock appointment. There's a large display in the room showing the waiting times for the various doctors. It is also displaying a word game that shows random letters and asks users to form words with them. George notices that someone is playing with it because from time to time it shows a pop-up screen congratulating a player. George decides to play a bit to pass the time. The instructions say he can SMS words to a phone number or use a mobile web page to play: he decides to use the web version in his smartphone. After he logs in the web page using his Google account, he sees there are other applications he can use, but he is focused on the word game so he chooses it and begins thinking of words to form with the current letters on the big display. "xerox" is the first word he gets correctly! He gets extra points for the 'x' letter and his name appears at the bottom of the high-scores table. He still doesn't know who else in the room is playing but they are getting some healthy competition!*

Awareness of other people's interactions.

Shared, concurrent interactions.

3.2.2 Open display network

Davies et al. [2012] have described their vision for an open display network, which points further possibilities and characteristics for this interaction environment. Generally, the vision of open display networks "*includes large-scale networks of pervasive public displays and associated sensors that are open to applications and content from many sources*" [Davies et al., 2012, p. 58].

In their analysis, Davies et al. ground their observations in five building blocks:

- open architectures that support application development and allow programmers to define application scheduling, adaptation to display geometries, display coordination, application distribution, and security.

- situated information and applications that users can actively influence, increasing participation and resulting in more interesting, relevant, and adapted content for the current situation of each display.

- privacy-compliant personalisation and control that allows users to remain in control of their personal data, and in control of what information they disclose in each display.

- engaging and efficient user interaction models that are simple and understandable to users.

- viable business models that make displays economically attractive to display owners and content producers.

An open display network will thus be open to many independent entities so that: display owners can easily join and add public displays, sensors and interaction mechanisms; display owners and users can find and use available applications (or, more generally, content) in their displays; application developers can create and distribute content and applications for display owners to use; and users can easily interact with the available displays and applications. A network like this will become an important target for developers that will want to develop interactive applications for public displays, and distribute them globally to run on any display.

On an open network of public displays, one will expect that each display will most likely be used as a general information appliance, showing content from various independent applications. This raises various questions about what applications are and how they might behave, which have implications for and interaction abstraction for this new platform. For example, on the web and desktop platforms and in some closed display networks, users are expected to be registered in the application or system before being able to use it. On an open display network, with hundreds of displays, and where users may constantly encounter new applications, interaction must be very lightweight and easy to accomplish without lengthy registration forms. An open display network demands support for heterogeneous interaction mechanisms so that each display owner may choose the interaction infrastructure that best serves the needs of a place within his cost limits. An interaction abstraction must cope with this heterogeneous environment ideally shielding application developers from it.

3.2.3 Interaction abstractions

In section 2.3 – Software Support for Application Development – we described several approaches to providing interaction abstractions, including widgets. In this section, we start by revisiting the widget concept as an example of what might be a successful abstraction for public displays. We then highlight the particular aspects of public displays that make it difficult to directly apply the abstraction concepts from the desktop platform.

Revisiting desktop abstractions

In the early days of graphical user interfaces applications developers faced a similar problem to the one faced today by public display application developers: there was no consistent way to integrate interactive features into applications. This problem was addressed with the emergence of various conceptual frameworks for interaction.

Mackinlay et al. [1990] proposed a design space of input devices, using a human-machine communication approach. In their design space, they consider the human, the input device, and the application: the human action is mapped into parameters of an application via mappings inherent in the device. "*Simple input devices are described in terms of semantic mappings from the transducers of physical properties into the parameters of the applications*" [Mackinlay et al., 1990, p. 145]. A device is described as a six-tuple composed of: a manipulation operator, input domain of possible values, a current state, an output domain, and additional device properties. This six-tuple can be represented diagrammatically, and this graphical representation of the design space has been used extensively to characterise and compare different input devices.

Foley et al. [1980] produced a taxonomy which organises interaction techniques around the interaction tasks they are capable of performing. The interaction tasks represent high-level abstractions that essentially define the kind of information that applications receive in result of a user performing the task. They form the building blocks from which more complex interactions, and in turn complete interaction dialogues, can be assembled. They are user-oriented, in that they are the primitive action units performed by a user. Foley's tasks were based on the work by Deecker and Penny [1977] which identified six common input information types for desktop graphical user interfaces: position, orient, select, path, quantify, and text entry. Foley also identified various interaction techniques that can be used for a given task and discussed the merit of each technique in relation to the interaction task.

Myers [1990] proposed interactor objects as a model for handling input from the mouse and keyboard. An interactor can be thought of as an intermediary abstraction between Foley's taxonomy and concrete Graphical User Interface (GUI) widgets. Interactors support the graphical subtasks, but abstract the concrete graphics system, hide the input handling details of the window manager, and provide multiple behaviors, such as different types of graphical feedback, that can be attached to user interface objects. Myers defined six interactors: menu-interactor, move-grow-interactor, new-point-interactor, angle-interactor, text-interactor, trace-interactor. The same interactor can be used to implement various concrete GUI widgets.

This type of research led to the now widely used concept of user interface widget (also known as "interaction object" or "control"): an abstraction that hides the low-level details of the interaction with the operator, transforming the low-level events performed by the operator into higher level events [Bass and Coutaz, 1991]. Widgets provide support for the three main stages of the human action cycle [Norman, 2002]: goal formation, execution, and evaluation. Their graphical representations and feedback support mainly the goal formation and evaluation stages. Widgets

have a graphical representation that application developers use to compose the GUI of the application, supporting users in the goal formation stage by allowing them to see the available features of an application. Widgets also support the evaluation stage by providing immediate graphical feedback about their state. For example, a textbox widget echoes the typed characters to show what users have already written and shows a blinking text cursor to indicate that it can accept more input. The internal behavior of a widget supports the execution stage and insulates application from low-level input events transforming them into high-level events. For example, an application that needs users to input a text string does not need to handle individual key presses; it can use a textbox widget that does this low-level handling and passes back to the application the complete text string.

The kind of information carried by the interaction event defines what interaction task is being accomplished [Deecker and Penny, 1977; Foley et al., 1980; Ohlson, 1978]. From an informational perspective, multiple types of widgets could be used to accomplish a desired task: for example, an application that requires users to input a number could use a number type-in widget, a slider, or a spinner.

Interaction in public displays

While it seems reasonable to apply successful lessons from the desktop world, there are significant differences that need to be accounted for when considering the adaptation of those principles to the specifics of the interaction environment around public displays.

The different sized displays afford different types of interaction but they can function in an integrated way in the ecosystem, offering different synergies and opportunities. For example, Dix and Sas [2010] examined several synergies and opportunities between personal mobile devices and public displays, addressing issues such as the physical size of the situated display, the use and purpose of the mobile devices, the level of integration of the public and personal devices, the movement and physical contact within the interaction, the spatial context of the situated display, and the social context. They analyse two main types of conflict that occur between the interacting users of the public display audience: conflicts of content, and conflicts of pace. Conflicts of content, i.e., what is seen, can occur for various reasons: "(1) conflict between the use of the screen for displaying content and for displaying interactive feedback (menus, etc.); (2) conflict between different users wanting different specific content (3) conflict between the particular requirements of an individual and maintaining a content stream that is intelligible, useful and engaging for bystanders". Conflicts of pace, i.e., when it is seen, includes two types: "(1) users cannot always have things when they want due to other users requests (c.f. content conflict), the playing of media, etc. (2) users cannot speed-up, slow-down, stop or replay the flow of information because of the audience." Resolving these conflicts is a challenge for public displays, particularly for multi-user, multi-application systems. In the "dual display" approach, Kaviani et al. [2009] explore interaction concepts that take advantage of both input and output capabilities of interactive public displays and personal mobile devices. Similarly to Dix and Sas [2010], they consider

two types of conflicts when multiple users attempt to interact and manipulate the same content simultaneously: conflicts in space and conflicts of pace/flow. Conflicts in space mainly originate from the limited screen space to provide visual feedback when executing a sequence of actions or providing information about a new system state. Conflicts of pace/flow usually occur when users do not have any feedback on the system behaviour. In order to reduce these conflicts the authors defined four design strategies: localized interactions, distributed system state, providing display focus, and cause summary, which differ on the type of feedback presented in each display. This integration of different kinds of displays creates a very different setting for applications, which desktop interaction abstractions were not meant to address.

Unlike desktop systems, which usually rely on a very small set of input mechanisms – most often just a keyboard and mouse – public display interaction can take advantage of very different input mechanisms, as we have seen in previous chapters. This variety of mechanisms creates an additional challenge for public display interaction abstractions. Often, public display systems use mobile devices as interaction mechanisms. The breadth of mobile interaction mechanisms has already motivated research that tries to systematise the cumulative knowledge around mobile techniques for interaction. Building on Foley's graphical interaction tasks, Ballagas et al. [2008] developed a design space for comparing how different mobile device based input techniques could a support a given interaction task. The techniques can be compared along various dimensions such as the number of physical dimensions (1d, 2d, 3d), the interaction style supported, the type of feedback provided, and whether the technique provides absolute or relative values. As stated by the authors, their design space is "*an important tool for helping designers . . . select the most appropriate input technique for their interaction scenarios*" [Ballagas et al., 2008, p. 387]. The work by Ballagas et al. [2008] provides a valuable design space for reasoning about the multiple types of interaction with public displays using mobile devices, but it does not provide a directly applicable solution for application programmers.

3.3 Design Requirements

In this section, we describe the requirements that an interaction abstraction toolkit for public displays should support. These requirements stem from the assumptions and characteristics of the interaction environment, the vision of open display networks, and the prior knowledge about desktop abstractions. We now describe these requirements.

3.3.1 Public display interaction controls

An interaction abstraction for public displays must provide support for interaction tasks and controls suitable for public display interaction.

The controls that programmers have available from desktop computers are generally not suitable or not at the right level of abstraction for creating public display applications. In the train station scenario, for example, how would a programmer implement the download coupon feature using the desktop controls, without having to worry about the concrete mechanism (Short Message Service (SMS), Bluetooth, mobile application) the user would choose to carry out the action? The interaction environment around public displays is very different from other platforms, making users' interaction goals when interacting with a public display very different from their goals when interacting with a desktop computer. This can easily be attested by looking at the various uses of the display systems described in section 2.2. Application developers should be able to choose from a variety of interaction controls that support the kind of data their applications need to exchange with users. This certainly requires support for a different set of interaction controls from the ones supported in desktop computers. These controls should provide to public display application a similar abstraction level that current desktop widgets provide to desktop applications, allowing developers to focus on the kind of data the application needs to receive.

3.3.2 Standard graphical representations

An interaction abstraction for public displays must provide consistent graphical representations for actions available on the display, and for the presentation of feedback about users' actions.

A recurring challenge in public displays is how to communicate their interactivity so that people who see a display know that it is interactive and may decide to interact with it. Dix and Sas [2010] divide the audience of a public display into five categories:

participant actively engaged with the system doing some form of input/interaction;

unwitting participant triggers sensors to have some effect, but does not know it;

witting bystander sees the screen and realises interaction is occurring;

unwitting bystander sees the screen but does not realise interaction is occurring;

passer-by may know screen is there, but does not watch or interact with it;

Individuals move from one role to another, if given enough information and understanding (see Figure 3.1). In order for an individual to move from the unwitting bystander to a witting bystander it is necessary first that he realises that the display is interactive (comprehension). To move from a witting bystander to a participant, it is necessary that he understands how to interact and that he finds the display compelling enough in order to become an active participant (involvement). (This this is similar to crossing the thresholds from peripheral to focal awareness, and then

to participation, in the public interaction flow framework by Brignull and Rogers [2003]).

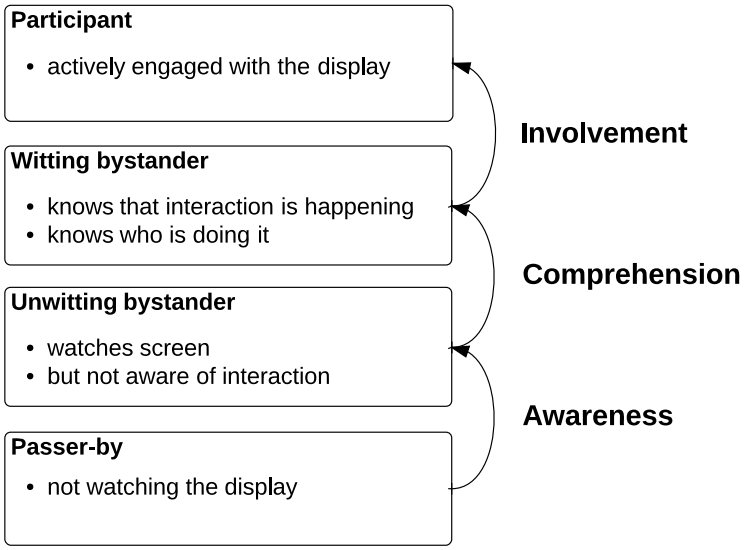


Figure 3.1: Changing user roles in public display interaction. Adapted from Dix and Sas [2010].

Public displays typically employ a number of strategies to let users know that they are interactive. Müller et al. [2012] identified six strategies for communicating interactivity in public displays: call-to-action, attract sequences, nearby analogue signage, the honey-pot, inviting passers-by, and prior knowledge. Prior knowledge refers to the fact that users already know that a device is interactive (because they have used it, seen people use it, or any other reason). In the university news scenario, Sophia realized she could interact with the display because there was a familiar graphical cue – the like button – that she recognised from a different environment. Only after recognising this button, she decided to interact. In the waiting time scenario, the congratulations popup confirmed George that someone was interacting with the display, and so he could too. An interaction abstraction can help in establishing prior knowledge by providing consistent graphical representations for actions available on the display, and for the presentation of feedback about users' actions.

3.3.3 Concurrent and shared interaction

An interaction abstraction for public displays must support concurrent, shared interaction by multiple users.

The many-many nature of the social interaction with public displays means that they must support multiple, concurrently interacting users. Also, most applications will need, at least, to be able to distinguish between different users in order to provide a shared environment where users know who's affecting what. For example, Dynamo [Brignull et al., 2004] allowed users to divide the screen space into areas with restricted access to different groups of users. Jumbli [LocaModa, 2010] allows various users to submit words at the same time, and keeps, and displays, users' high-

scores. In Proactive Displays [McDonald et al., 2008] users' personal information was used in various ways throughout the conference. All these are examples of systems that support concurrent interaction where various users can interact with the same application at the same time.

Additionally, the interaction abstraction should support shared interaction, allowing several users to interact with the same interaction feature simultaneously. Some systems support concurrent interactions but, either by design or physical constraints, only allow one user at a time to interact with a given part, or feature, of an application. For example, in multi-touch tables or walls, interactive features are bound to a specific spatial region and users cannot physically press or gesture at the same region simultaneously. Consequently, interaction toolkits for multi-touch platforms usually don't address the issue of having simultaneous input on a button, for example. Other systems, such as some web Content Management Systems (CMSs) require administrators to get exclusive access to a specific content item before editing it, preventing other administrators from editing the same item. Our target interaction environment demands a shared interaction model, where these constraints don't exist (at least in general, if a particular display only provides touch-based interaction, the physical constraints will naturally be present). For example, in the waiting time scenario, George and other users were simultaneously interacting with the same feature of the same application.

3.3.4 Multiple interaction mechanisms

An interaction abstraction for public displays must support and abstract various interaction mechanisms.

The public display systems surveyed in section 2.2 reveal a plethora of interactions mechanisms ranging from SMS [Ferscha et al., 2002], email and IM [Paek et al., 2004], Bluetooth naming [José et al., 2008], Twitter [LocaModa, 2010], Radio-Frequency Identification (RFID) [McDonald et al., 2008], body movement [Sawhney et al., 2001], gestures [Vogel and Balakrishnan, 2004], face detection [Grasso et al., 2003], mobile applications [Scheible and Ojala, 2005], and more.

An interaction abstraction for public displays must support various interaction mechanisms in order to support the various types of existing display systems, and work in an open, flexible environment where each display owner can choose which resources will be available. An important aspect of this support, however, is abstracting the concrete input mechanism so that application developers don't need to worry about which mechanism is available at a given place. For example, in the train station scenario John had several options for downloading the coupon, but in order to implement the coupon application one should not need to know which concrete mechanisms would be available. The coupon application should work seamlessly in a place where there was only email as the download alternative. An application developed for a public display should work transparently in places with support for different sets of interaction mechanisms.

3.3.5 Ubiquitous interaction

An interaction abstraction for public displays must support a ubiquitous interaction environment where applications are always available for interaction.

On desktop computers users decide when the application should run, when it should be in the foreground receiving input, and when it should be terminated. In an open public display ecosystem, there may be different levels of control over which application shows content on the display at a particular moment. For some displays, the display owner may have full control over the scheduling of the various applications; in other displays, users may have some degree of control and ask for a specific application to be displayed at a given moment. Whichever model is used, public display applications should generally be available for interaction independently of whether they are currently on-screen, or not. For example, in the coffee shop scenario the public display may not have been displaying the menu comments application at the time that Sarah and George came in, but that did not prevent them from being able to scan the QR code and enter their comment about their snack.

Contrary to what happens on the desktop, where applications need to be on the foreground of the display to receive input, an interaction abstraction for public displays should support interactions independently of the on-screen state of the application. This kind of ubiquitous interaction environment can help mitigate the "conflict of pace" mentioned by Dix and Sas [2008], which happens because users are not in full control of the public display. A ubiquitous interaction environment guarantees that, at least, the display's scheduling does not impose the pace for the interaction with an application.

3.4 Conclusion

The public display environment in which we are focused on for the purposes of the current work consists of an ecosystem of open public displays, sensors, and input mechanisms that support a many-many social interaction. In this kind of environment for public displays, an ideal interaction abstraction must support a number of requirements, which we summarise in Table 3.1.

Table 3.1: Requirements for an interaction abstraction for public display applications.

Requirement	**D**escription
Public display interaction controls	The interaction abstraction should provide developers with various types of controls specifically designed for public display interaction.
Standard graphical representations	The interaction abstraction should provide graphical cues so that users easily understand that a public display is interactive and what features it has.
Concurrent and shared interaction	The interaction abstraction should support multiple, concurrent, shared interactions from different users.
Multiple interaction mechanisms	The interaction abstraction should support and abstract several interaction mechanisms.
Ubiquitous interaction	The interaction abstraction should allow users to interact with an application at any time, regardless of whether the application is currently showing content on the public display.

Chapter 4

Digital Footprints for Socially-Aware Interactive Displays

Contents

4.1 Introduction

An important aspect of public displays is their situatedness – the capability of presenting content that is relevant and interesting to the audience in a particular location. To do this, they must continually adapt to their location and to the social environment that surrounds them. The data that results from the various user interactions with the public display may provide a relevant source of data on which to build adaptation strategies. However, we need a better understanding of what that data might mean and how to use it. Abstracting the multiple types of interactions that may occur with public displays into high-level information is an important step towards the emergence of generic situated display applications that are able to adapt to a particular display location.

In this first approximation to understanding interaction abstractions, we analyse multiple interaction alternatives from the perspective of the information they generate. Rather than considering the specific affordances or semantics of the interactive features offered by the display, we focused on the type of digital trace they generate about the audience, or user. We use the concept of digital footprint to refer to the digital traces generated as a side effect of implicit or explicit interactions with the display. We propose a framework for designing socially-aware interactive public displays. Our goal is to create a tool for informing designers of situated displays about the relation between the supported interaction types, the type of digital footprints they can generate, and the type of adaptation processes they may support.

4.2 Digital Footprints

Making sense of the myriad of sensing and interaction events that can occur across a display network requires higher-level abstractions that are independent from the specific affordances or the semantics of the interaction offered by particular display applications. We propose a framework created around the concept of digital footprint as an abstraction for representing the traces left behind when people implicitly or explicitly interact with public displays. A digital footprint focuses on the nature of the information generated about the interaction itself and abstracts away from the particular interaction modality being used or the particular application semantics in which the interaction occurred. They aim to represent preferences, characteristics and behaviour of the audience and users of the public display, regardless of how they are expressed.

These footprints were created from the analysis of various existing public display systems, described in the research literature. Based on their key properties, we aggregated those digital footprints according to four main categories: presence sensing, self-exposure, user-generated content, and actionables, as described in Table 4.1, providing a mapping between multiple interaction alternatives and their contribution to the generation of local digital footprints. These categories represent not only different types of interaction, but also increasingly higher levels of engagement with

the display system. We then analyse the types of adaptation processes that can be associated with each of those digital footprints, thus providing a mapping from footprints into context-aware adaptation processes. Overall, these mappings provide the framework for reflecting on context-aware behaviours without being caught up by the specificities of any particular interaction or sensing mechanism, thus providing a path for generic context-aware mechanisms.

We will now describe these footprints in more detail, analysing how they can be generated and how they can contribute to the adaptation process.

Table 4.1: Digital footprints from interaction with public displays.

Digital footprint	**D**escription
Presence sensing (Detection, Characterisation, Identification)	Being physically there
Self-exposure	Explicitly managing exposed identity
User-generated content	Pushing content to the system
Actionables (Interactive, External)	Responding to the system

4.3 Presence Sensing

The ability to implicitly collect information about the presence of nearby people is an important element for characterising situations. Despite its stronger potential for generating rich information about usage, explicit interaction events are necessarily sparser than presence events, as there will normally be many more people present than those actively engaged with the display at any given moment. Presence sensing is thus critical to enable content adaptation to occur even when there is no one interacting explicitly. Presence sensing may involve increasingly complex levels of presence information, more specifically: the ability to simply detect presences (detection), the ability to characterise those presences (characterisation) and the ability to associate presences with unique entities across multiple sessions (identification).

4.3.1 Presence detection

Presence detection is the most basic level of presence information in which the system is simply able to detect whether or not there is someone nearby, and possibly at what distance. Multiple off-the-shelf sensors can be used for this purpose. Distance to the display can be determined by combining presence sensors with different sensing ranges or using distance sensors that report distance. Computer vision techniques,

such as frame differencing, can also be used to sense movement, which can be used as an indication of presence.

Information about the presence of someone near a display, even without knowing who or how many, can be used as a trigger for presenting specific content, and particularly as part of an attraction loop designed to entice people to interact when passing-by, or to prevent content from changing while someone is near the display. Grasso et al. [2003], in their Community Wall display system, used frame differencing as part of the presence detection system to prevent the display from changing content while someone was reading from it. Sawhney et al. [2001], in the Community Aware Portals system, used presence detection for attracting people to look at the display:

> "*When a person is seen walking-by the space, a series of images are shown cycling through, depicting the recent stories in memory. If the person stops to glance at the display, a preview of the current story (news headlines or weather map) is shown for a short duration. If the person then continues to glance, the system assumes she wishes to browse the article in more detail*" [Sawhney et al., 2001, p. 68].

Knowing at what distance someone is from the display can be important for the display system because there is normally a strong correlation between distance and the awareness level that people may have about the display. If people are close, it is much more likely that the display is currently the focus of their attention. In the Range whiteboard [Ju et al., 2008], for example, an infrared distance sensor was used to determine the distance of a user from a whiteboard, to distinguish between several interaction zones (intimate, personal, social, and public zones). This allowed the whiteboard application to behave differently according to the proximity of its users.

The digital footprint generated by a presence detection mechanism is essentially a presence/absence or distance pattern. Analysed over time, however, this may provide more a complex characterisation of the place in terms of people flow.

4.3.2 Presence characterisation

Presence characterisation is the ability to count and possible determine particular characteristics about the presences detected near the display, which broadly corresponds to the concept of audience measurement. In an attempt to set industry standards for audience measurement, the Digital Place-based Advertising Association (DPAA) (formerly Out-of-home Video Advertising Bureau (OVAB)) has produced a guidelines document that describes the Average Unit Audience as "*the number and type of people exposed to the media vehicle with an opportunity to see a unit of time equal to the typical advertising unit*" [Spaeth et al., 2008, p. 4]. These guidelines have a clear focus on what is normally called the Opportunity to See (OTS) and on three qualifying characteristics associated with that opportunity:

presence, notice, and dwell time, but they also include recommended demographics such as age and gender. A vast range of audience measurement technologies have emerged to estimate, not only the number of people in front of a display, but also to determine their attention span or inferring some type of characteristic about them, such as age or gender [Quividi, 2013; TruMedia, 2013]. These tools usually use computer vision techniques that are able to detect people's faces and estimate their gender [Verschae et al., 2007] and age [Kwon, 1999], and normally involves placing a video camera on the display, typically on the top and facing the audience, and processing the images to generate reports about the number, attention span, and characteristics of viewers.

The most typical example of content adaptation using presence characterisation is targeting adverts according to the characteristics of the current audience as, for example, in the case of the Eye Flavour that selects adverts based on age and gender [NEC, 2009]. Presence characterisation can, however, be used in general content adaptation processes outside advertising. It is easy to imagine public displays that select news content, for example, according to some characteristic of the majority of the audience, or that adapt how fast content changes according to the attention level of the audience.

The digital footprint that results from presence characterisation is an aggregated presence pattern that can include, among other factors, the number of people, age, gender, and attention span.

4.3.3 Presence identification

Presence identification is the ability to detect unique identities within presences and recognise different visits by the same person. Determining who is present, in the sense that the display system is able to determine that the same person is present in different occasions, gives the display system not only the possibility to determine how many people are present, but also to establish a correlation between different people or groups of people. This may be achieved through face recognition techniques, but the most common approach is by far the use of some personal device (with Bluetooth or Radio-Frequency Identification (RFID) capabilities, for example) as a proxy for the person.

Bluetooth has been used extensively as presence detection mechanism as it is widely available and enables many mobile phones to be discovered and uniquely identified through their Bluetooth address. The BluScreen system [Sharifi et al., 2006], for example, uses Bluetooth detection to maximise the exposure of people to new adverts. The Cityware project [Kostakos and O'Neill, 2008a] studied online social networks alongside their real-world counterparts by merging users' online social data from Facebook with mobility traces captured via Bluetooth scanning. Cityware also included a set of in situ visualisations about Bluetooth presences [Kostakos and O'Neill, 2008b] that provided people with information about current or recent Bluetooth encounters.

RFID tags can also be used for presence identification. They are small and can easily be incorporated into many existing artefacts as in the IntelliBadge project [Cox et al., 2003], where users participating in a conference were given RFID augmented badges that were used to track them through the conference rooms. A display at the conference cycled through several visualizations of the resulting data. A similar approach was used by McDonald et al. [2008] in the Proactive Displays to display personal information about the conference participants in public displays placed at different locations in the conference venue.

Presence identification raises considerable privacy issues. It should only occur as part of a process in which people are informed and value the extra functionality that the existence of this information may be providing. The ability to enable or disable presence identification according to the circumstances would be equally important. This is a point in which Bluetooth, which allows discovery to be easily disabled, takes considerable advantage over RFID tags.

The digital footprints resulting from presence identification correspond to individual presence information and open many new opportunities for the content adaptation process. They may serve to build individual profiles that characterise people according to their presence patterns, e.g. regular visitors vs first-time visitors. They can be used to optimise content exposure by selecting content that has not yet been shown to the people that are currently present. If a network of displays is involved, presence identification may also serve to show some variant of the same content to the same person at different locations, thus reinforcing the message, or to select content based on where the person has been before. The existence of multiple sensing points would also enable new forms of impact assessment. For example, it may be possible to measure how many people who have been shown an advert about a nearby store will then go to that store, or simply how the number of first-time visitors has increased as the result of a campaign.

4.4 Self-exposure

Self-exposure is the ability of the display system to collect information that people have explicitly created to represent their characteristics or preferences. This personal information can take many forms, such as a list of personal tags, a reference to a user's personal web page, the user id in a Social Networking Site (SNS) or a purposely created profile. Self-exposure should also be seen as a way for people to take control over how their identity is projected in the public display. It should be part of a process whereby people expose something about themselves because they understand and value the implicit response of the display to that self-exposure. Moreover, the intended level of self-exposure may be strongly influenced by a broad range of circumstances. Therefore, a key feature for self-exposure mechanisms is some type of control on how that exposure should occur at any given moment. This means not just the ability to enable or disable presence detection, but also the ability to dynamically manage self-exposure in a way that is easily controlled and understandable. Self-exposure differs from presence in the sense that it is an explicit

act of identity management, while with presence we assume that all the information was implicitly generated. It differs from other forms of interaction in the sense that there is no pre-established connection between self-exposure and a specific type of reactive behaviour by the display system. Potentially, different display systems may react very differently to the same form of self-exposure, or even the same system may also react differently under different circumstances.

Self-exposure may be achieved by combining presence identification with an a priori definition of a user profile that is associated with the identity. In Proactive Displays [McDonald et al., 2008], users attending a conference registered their affiliation, interests and personal web page before the conference day and were given RFID augmented conference badges at the conference site. Throughout the conference, several displays reacted to the nearby participants showing and creating associations between their profiles and creating opportunities to socialise. A limitation of this approach is that people could not easily change their profile, and had no control over what information to expose at any given moment. Another alternative is the use of a personal information device running a custom application to manage and expose a profile. This application can connect automatically, or on demand, to the display system and communicate users' preferences. This can be exemplified by the Hello.Wall [Prante et al., 2003], a system in which people would carry a ViewPort device – a modified Personal Digital Assistant (PDA) – to communicate with an ambient display. Similarly, in the Mobile Service Explorer system [Toye et al., 2004], a custom mobile application could be configured with personal information that is made automatically available to the display system when a user interacts with it. One advantage of these approaches is that the information is always available to be updated by its owner and it may be possible to have greater control on how to manage self-exposure, but they have the disadvantage of requiring a dedicated mobile application. The use of Bluetooth names to express self-exposure, as explored by José et al. [2008] and Davies et al. [2009], is an opportunistic alternative that is easily available on almost any device. It allows people to enter predefined commands in the Bluetooth name of their mobile phone. When that person approaches a display, these commands can be obtained and interpreted as part of the person's preferences. For example, Ribeiro and José [2010] have shown how tags exposed this way by multiple people could be used to select content feeds from the Internet for presentation on a local display. The downside is that many users may have difficulty finding the Bluetooth settings, and some more recent mobile phones automatically turn off Bluetooth discoverability after a few seconds.

In all the previous examples, there was the assumption that information would be specifically created for the purpose of influencing a display system. However, a very promising alternative is to explore connections with the many SNS where people already have extensive descriptions about themselves and their preferences. WhozThat [Beach et al., 2008] uses the SNS profiles of people nearby to create context information that can then be used to support spontaneous interactions or drive the music selection. People are expected to use a mobile phone running an identity sharing protocol that will advertise their online identities to the other nearby devices. This system does not consider the use of public displays or any explicit selection of which information to share, but it is an example of using SNS profiles as a sort of personal data aura that can be used to mediate digital self-exposure. Bohmer

and Muller [2010] conducted a study on the exhibition of SNS profiles in public settings. Using mock-up images they asked people about their willingness to expose profile information in two types of what they called social signs. The first was a personal social sign projected around the person and showing parts of the respective profile. The second was an interpersonal sign, projected in such a way to link two people and representing some type of connection between them, such as having a mutual friend or sharing an interest. The study provides an interesting example of the type of identity projections that can be generated by these connections with SNS.

The information associated with self-exposure footprints can be very varied. In its simplest form, the result may be just a list of keywords generated from multiple forms of preferences expression. However, with access to profiles, much more structured information can be obtained. The potential of self-exposure in the content adaptation process may be particularly relevant for targeted content adaptation, including advertising. Depending on the type of information collected, it may support multiple forms of targeting, such as demographic, contextual or behavioural.

4.5 User-generated Content

User-generated content is the ability of the system to accept content originating from the users of the display. This is achieved by allowing people to post their own content for publishing at the display, either directly or indirectly through a reference to the content (an Uniform Resource Locator (URL), for example).

Many displays have been created that support some variant of this feature. WebWall [Vogl, 2002], for example, allows people to submit content (ads, polls, pictures, videos) using Short Message Service (SMS), email or a web interface. The Plasma Poster [Churchill et al., 2003a] system supports content (photos, text, web pages) submission through two interfaces: email and a web form. Multimedia Message Service (MMS) has also been used as an input interface for display systems. For example, the Joe Blogg project [Martin et al., 2006] includes a display designed in the form of an interactive artwork where people can send pictures and text messages through MMS or SMS. Bluetooth can also be used to push content to a display system using either standard OBject EXchange (OBEX) exchanges or custom mobile applications. Both Hermes [Cheverst et al., 2005] and Snap and Grab [Maunder et al., 2007] use the OBEX feature to enable users to send pictures and other media to a display. In both cases, the user just selects the content on his mobile phone, chooses the "send via Bluetooth" command and selects the Bluetooth device name associated with the target display. An example of a custom Bluetooth application for sending content is Touch & Interact [Hardy and Rukzio, 2008], which allows users to choose a picture in their mobile phone and touch the display in the position they want to place that picture. The main advantage of a custom application is that it can be built to interact specifically with a given display, thus allowing a richer and more convenient interaction. For example, the application may be able to automatically determine the Bluetooth address of the display system, alleviating

the user from the task of manually searching for nearby devices and selecting the correct device name. The application may also be able to determine beforehand what type of content is acceptable by the display system, preventing the user from sending content that will be rejected. OBEX on the other hand, has the obvious advantage of not requiring the installation of any additional software in the mobile device, and thus enable more opportunistic interactions. Also, from a developer's perspective, OBEX is more attractive because, given the multitude of different devices, it is often the case that multiple versions of the application have to be created to deal with the hardware and software variants in devices. Also, these applications must be somehow distributed to users which may be yet another barrier to usage.

When generating content to a display, users are implicitly associating that content with that particular place. The nature of the digital footprints generated by such process depends on the type of content. However, the analysis of the content and its meta-data should produce a relevant characterisation of that place and its local practices. This is already common on the web, where a vast and sophisticated range of techniques has been emerging to analyse user-generated content. For example, tags in Flickr images have been used to extract place semantics [Rattenbury and Naaman, 2009], and tweets have been used to determine mood and emotions [Bollen et al., 2009]. If displays were able to generate enough content from users, similar techniques could also be applied to generate footprints that would be very relevant for content adaptation processes.

4.6 Actionables

Content on a public display is often some type of actionable: a message or an interactive feature intended to cause people to act [Müller et al., 2007]. Actionables provide a specific demand for action, such as downloading a content item, selecting one of several available options or reading a 2D code from the display. An actionable footprint represents a reaction to one of those actionables.

Generating this type of footprint requires some mechanism for tracking the actions taken and also some type of system-wide reference for the actionables, e.g. an URL or some other unique id. The existence of a unique reference that can be tracked system-wide separates actionables from the myriad interactive features that may exist in a display system and which in many cases will not be meaningful outside the specific application in which they are available.

Actionables may provide one of the most promising paths for content adaptation in public displays because they enable several forms of automated impact assessment and advert valuation, ultimately leading to something similar to the pay-per-click concept. When considering how to generate actionable footprints, we need to take into account two major types of actionable: interactive actionables in which the reaction occurs within the system and can thus be automatically detected by the system; and external actionables in which the action occurs outside the context of the display system and cannot be tracked automatically by the system itself.

4.6.1 Interactive actionables

Interactive actionables are invitations to interaction that can be interpreted by the system. They involve giving users some type of control and being able to track their options. Allowing people to pull content from the display system is a common example of an interactive actionable. Knowing who has downloaded which items enables the system to infer interest on the content items that are being offered. Giving people some type of control over the system behaviour may also be used to generate interactive actionables. For example, Jukola [O'Hara et al., 2004], which allows people in a bar to vote on the next music to be played by selecting music from a list, is an example of a system that could generate this type of footprint. Rating content items allows people to explicitly say they like or dislike an item and gives the display system information about the popularity of content items. It is also possible to conceive a Games With A Purpose (GWAP) approach [von Ahn and Dabbish, 2008] in which all sorts of polls, quizzes, questionnaires, or games are designed to provide engaging experiences, while allowing the display to collect users' preferences and interests. These actionable footprints may be anonymous or identifiable depending on the interaction mechanism provided, but the key point is that the meaning of the reactions can be interpreted system-wide and not just within the scope of a particular application.

4.6.2 External actionables

With external actionables, the intended action is to occur outside the scope of the display system, and therefore, the system is unable to automatically assess their efficiency. At best, their efficiency can be assessed through some type of off-line mechanisms, such as the collection of redeemed coupons.

MobiDiC [Müller and Krüger, 2009] is an example of the use of digital coupons in public display advertisement. The public display shows adverts with coupons that people can photograph with their mobile phone and redeem at the associated shop to receive the announced item. The time and location where the ad was displayed is encoded in the coupon. When the coupon is redeemed and the shop owner enters the code back to the system, the specific advert and display are identified and the aggregated results can tell how successful the advert was.

4.7 Mapping Footprints to Adaptation Models

Our goal with this framework was to inform the process of mapping multiple types of sensing or interactive features in public displays into specific content adaptation models. The digital footprints that have been presented support the first part of that mapping. By focusing on the essence of the information generated from sensing and interaction features, they provide a framework for reflecting on the meanings of the interaction without being caught up by the specificities of any particular interaction

mechanisms. In this final analysis, we discuss the second part of that mapping, which is to describe what types of digital footprints may be needed to support specific content adaptation goals. We hope that, by making the relationship between content adaptation goals and footprints more explicit, this framework may contribute to focus the design of public displays on the data generation objectives, while avoiding the pitfalls of focusing too much on a particular application or interaction technique.

Overall, the entire set of digital footprints constitutes a collection of data which can be used to characterise a place profile, enabling the display system to adapt its behaviour to that particular social setting. Regardless of their generic contribution to this broad adaptation, specific types of footprints can support specific types of adaptive behaviour.

Based on this framework, context-aware public display solutions may start by setting their specific goals, consider the digital footprints that the display systems may need to generate, and finally conceive sensing or interaction features that are capable of generating the appropriate information while providing an adequate user experience. This last step should also consider that these different digital footprints are not completely independent from each other. They are more like a stack of increasingly richer information about the audience of a public display. For example, availability of presence characterisation also generates presence detection. Presence identification is implicit in self-exposure and identifiable interactive actionables. The correct choice of the interactive features may optimise the generation of the most appropriate set of digital footprints.

Table 4.2 presents a summary of how to map multiple content adaptation goals to the available digital footprints. Several types of content adaptation goals are listed and associated with the specific types of digital footprint that could support them. Finally, given the myriad of sensor technologies and interaction modalities that can be used with public displays, it is important to have a mapping between the digital footprint and the possible sensor or interaction modalities that can be used to generate that footprint. This mapping can be used by situated display designers to help them choose the interaction mechanisms that a display should support in order to be able to collect a given set of footprints. Table 4.3 shows this mapping and, although it is not meant to be exhaustive, it lists the most common interaction modalities for public display interaction, making it a valuable tool for designing part of the interaction with a public display.

Table 4.2: Mapping between content adaptation goals and digital footprints.

Adaptation goals	**D**igital footprints
Attraction loops	Presence detection can trigger specific content to get people's attention.
Audience measurement	Presence characterisation may provide sophisticated reports about viewers (e.g., gender, age) and their attention.
Demographic Targeting	Presence characterisation may infer basic demographic information from presences, but self-exposure and specific actionables may generate even richer information, given that they involve people explicitly saying something about themselves.
Contextual Targeting	Self-exposure and specific actionables may generate keyword sets and other types of aggregate descriptions that will help to dynamically characterise the context of a public display in ways that are relevant for contextual targeting, such as hot keywords, social situations or ongoing local activities.
Behavioural Targeting	Presence identification and identifiable actionables generate rich information about people's behaviour and enable multiple types of behavioural targeting. Presence identification will mainly revolve around presence patterns, possibly at multiple locations, whereas actionables will revolve around expressed preferences inferred from actions.
Optimise individual exposure	Presence identification supports several types of optimisations related with specific individuals, from avoiding repetitions to purposely generating periodic repetitions for reinforcement.
Impact Assessment	Actionable footprints provide the most appropriate measure for engagement and therefore for really understanding the reaction of people to the message.

Table 4.3: Mapping between digital footprints and supporting interaction modalities.

Footprint	**I**nteraction Mechanism
Presence detection	· Movement sensor · Distance sensor · Computer-vision techniques
Presence characterisation	· People counter · Audience measurement tools · Computer-vision techniques for age, gender or attention classification
Presence identification	· Bluetooth · RFID
Self-exposure	· Any presence identification mechanism (plus custom profile or SNS profile) · Custom mobile application with integrated profile · Bluetooth naming
User-generated content	· Email/IM · SMS/MMS · Bluetooth OBEX · Bluetooth naming
Actionables	· Touch screen (Standard GUI controls) · Email/IM (Text commands) · SMS/MMS (Text commands) · Bluetooth (Text commands, e.g. BT naming; Standard GUI mobile application) · RFID (Proximity activation, e.g. Touch & Interact)

4.8 Conclusion

Situated displays cannot rely solely on a static pre-characterisation of the place they were designed to. They must adapt themselves to their changing environment by collecting digital footprints that will help in characterising the social context in which the display is embedded. In order to be efficient, digital displays need to target their audience's needs, expectations and tastes. By collecting digital footprints of people's interactions, displays can take a step in this direction. Interaction abstractions should also consider how implicit and explicit interactions might contribute towards a generic characterisation of a place and its situated displays. We have presented a framework that defines a mapping between interaction mechanisms and their contribution to the generation of digital footprints with relevance for the characterisation of a place. Each footprint may be used in isolation or in conjunction with other footprints by digital displays to target specific aspects of their audience. These digital footprints can be viewed as very high-level interaction abstractions that designers of public display applications can use to frame the general behaviour of an application.

Chapter 5

Interaction Tasks and Controls for Public Display Applications

Contents

5.1 Introduction

The interaction footprints presented in the previous chapter must be implemented in applications as interaction features. An interaction feature such as commenting on a specific content item can be supported by many different interaction mechanisms (emailing a given address, sending an Short Message Service (SMS) message to a given number, using an on-screen keyboard, etc.), depending on the resources of a specific place. Ideally an interaction feature should be implemented using standard, abstract, high-level interaction controls so that application developers don't need to consider all the possible interaction mechanisms. The commenting feature, for example, could be implemented by a text entry control that provides applications with text data, regardless of how the text is sent to the display system. Currently, however, high-level controls for public display applications do not exist and developers have to delve into low-level details, and create their own approach for dealing with a particular interaction feature using a particular interaction modality, leading to extra development effort outside of the core application functionality. In addition, this effort is replicated by each developer, potentially leading to poor designs and representing wasted effort.

Desktop computer programmers had to make a similar effort to support their interaction with users in the early days of graphical user interfaces. The problem was addressed with the emergence of reusable high-level interaction abstractions that shielded application developers from low-level interaction details, and provided consistent interaction experiences to users. Currently, desktop application developers can focus on the high-level interactive features of their applications, and abstract away from low-level issues, such as receiving mouse pointer events, recognising a click on a specific button or changing the visual state of a button that has just been clicked. These low-level input events are encapsulated by user interface widgets that provide developers with high-level abstractions, thus facilitating the task of creating an application. From the usability perspective, widgets also enforce consistency of the interface, allowing users to learn to interpret their affordances in a way that enables them to tackle new interfaces and programs by building on previous experience.

This type of abstractions may now also provide an important inspiration for addressing the similar problem being faced by public displays, where the transition to a new era of generalised interaction support will also require a step up in the abstraction scale. As described by Mackinlay et al. [1990, p. 147]: "*to achieve a systematic framework for input devices, toolkits need to be supported by technical abstractions about the nature of the task a input device is performing*". The proliferation of input devices and techniques for public displays reached a point at which it is both possible and fundamental to systematise the knowledge that may support the design of interaction toolkits for public display systems and ultimately enable interaction to become a common element of any display application in open display networks.

In this chapter, our objective is to take a first step in that direction by uncovering interaction tasks that may lead to the emergence of interaction controls for appli-

cations in public displays. Interaction tasks [Foley et al., 1980] represent high-level abstractions that essentially define the kind of information that applications receive in result of a user performing an interaction. By focusing on the interaction tasks we focus on the essence of the interaction that differentiates public display interaction from interaction with other computer systems.

5.2 Procedure

To uncover interactive tasks for public displays, we have made a comprehensive study of 52 publications about interactive public display systems, and coded the description of their interaction features. This approach aimed to go beyond specific interaction techniques and allow common interaction patterns to emerge from the assumptions and approaches applied across a broad range of interactive display systems. Our research followed an approach based on the grounded theory methodology [Glaser and Strauss, 1967], borrowing many of its phases: open, selective, and theoretical coding; memoing; and sorting.

We started with an initial set of 12 papers and did a first phase of open coding, in which we produced our first set of codes corresponding to specific attributes of the respective interactions. We then analysed these codes to aggregate some of them and remove others that were deemed not relevant from the interaction point of view. This much smaller set of relevant codes was used as the starting point in a second coding phase, where we coded 40 additional papers. Simultaneously, we started a third theoretical coding phase, identifying relationships between the existing codes, and producing new codes to reflect these relationships. In this phase, we started organising the existing codes into categories of interactions, along with their properties and concrete values associated with those properties. We adopt the definitions of categories and properties from Glaser and Strauss [1967, p. 36]: "*A category stands by itself as a conceptual element of a theory. A property, in turn, is a conceptual aspect or element of a category*".

To identify and distinguish categories, we analysed the interaction features that were being described, based on the underlying types of information that had to be exchanged between the user and the display system. These second and third phases were highly iterative and intermixed: we recoded previously coded papers more than once to make sure their coding was up-to-date with the latest categories and properties. For example, if we identified a new property while coding a paper, we would go back to previous publications and make sure we coded that property, in case we had missed it originally. The complete process originated a total of 87 codes that referenced 448 text segments in the 55 papers (the complete list of coded papers is available in appendix A).

Memoing, i.e., writing ideas associated with codes, was also an important part of the methodology and this went in parallel with all the coding phases. We used memos to start relating our codes together and forming a structured view (of categories, properties, and values) of all the interaction tasks that were emerging. We also

used memos to note possible missing properties and values that we needed to search in additional publications to make sure our categories were saturated. The memos associated with the categories became the first raw descriptions of our interaction tasks in the final description and analysis, after we sorted them to chain the ideas that emerged during the coding phases and turn them into a more logical narrative.

The categories that resulted from the coding process correspond to the interaction tasks that define the general information that the application needs to specify and the information that the application receives in the interaction events. The interaction tasks have properties that can take different concrete values and restrict the information or the behaviour associated with the task. These properties and values of the interaction tasks are mapped directly from the properties and values that resulted from the coding process. For example, the passage "CoCollage users who are connected to the web site in the café may also send messages directly to CoCollage via a textbox near the upper right of any page" is describing an interaction feature that allows users to send a text message to the display. In the third coding phase, this feature was coded with "data entry" (category), "bounds" (property), and "text" (value).

We then matched these interaction tasks against the concrete interaction mechanisms identified in the literature. We plotted the various implementations found on the literature in a spatial layout of a design space that extends previous work by Ballagas et al. [2008], which provides a valuable design space for reasoning about the multiple types of interaction with public displays using mobile devices. We thus used this as a starting point for our own work and extended it in two ways: by considering not just the smart-phone, but also other interaction mechanisms; and by considering the existence of new interaction tasks, beyond the ones defined by Foley, which may give a broader and more specific view of the interaction space with public displays.

Finally, we explored different combination of properties and values associated with the interaction tasks, and outlined a set of concrete interaction controls that can provide a starting point for the development of interaction toolkits for interactive public display applications.

5.3 Interaction Tasks for Public Displays

A key result of this work is a list of interaction tasks, properties, and values that overall characterise the major types of interactions with public displays. For each interaction task, we characterise it in terms of the information exchanged, the respective properties and the possible values for those properties, using examples from the surveyed display systems to illustrate the different properties. Table 5.1 summarises the tasks identified in this process.

Table 5.1: Interaction tasks for public displays: properties, and values.

Task	**P**roperty	**V**alues
Select	Type of selection	[Action, Option]
Data entry	Bounds	[Unbounded, Bounded]
Upload	Media type	[Text, Image, Video, Audio, etc.]
	Media location type	[Personal device, Public location]
Download	Media type	[Text, Image, Video, Audio, etc.]
	Media location type	[Display system, Public location]
	Target device	[Smartphone, Email, USB stick, Print]
	Target person	[Self, Other]
Signal presence	Location disclosure	[Automatic, Manual]
	Location verification	[Verified, Unverified]
Dynamic manipulation	Type of manipulation	[Cursor, Joystick, Keyboard, Skeleton/silhouette]

5.3.1 Select

The select task is equivalent to the select task of Foley et al. [1980], allowing users to trigger actions or select options in an application. It requires applications to specify the complete set of options or actions they wish to provide to users. The interaction event triggered by the display system will include the action or option identification, so that the application can determine which one was selected.

Type of selection

The type of selection property refers to what users are selecting: an action to be triggered immediately by the application, or an object from a list of possible objects. Using the terminology of Cooper et al. [2007], in action selection users input a verb (what action the application should perform), and the noun (the object on which to act) is usually implicit. In object selection, users input a noun, and later a verb (or the verb is implicit). These two types of selection are traditionally represented on Graphical User Interfaces (GUIs) in a variety of different forms; for example, on desktop systems programmers usually have at their disposal different sets of widgets for triggering actions (menus, toolbars, buttons), and for selecting objects (listboxes, dropdowns).

In regard to triggering actions, Vogel and Balakrishnan [2004, p. 142] in the Interactive Public Ambient Displays system provide an example using hand gestures: "*Two complimentary hand postures are used to hide and show the display of a user's*

own proxy bar. The hide action is performed with a palm away posture consisting of an open hand pointing up with palm facing the display . . ., analogous to the commonly seen 'stop' gesture used for traffic signaling in real life." QR codes are also a common alternative to provide users with a visual representation for an action, whether in a live public display, or printed on paper. In the Mobile Service Toolkit/Mobile Service Explorer (MST/MSE) by Toye et al. [2005, p. 64] for example, users could scan a visual code to have access to various actions: "*'Sally, can click on the tag using her MSE-enabled phone to establish a Bluetooth connection with the service. As soon as the phone connects with the service, her phone displays a message containing the current queuing time and asks whether she'd like to join the queue.*" Another example is the Bluetone system [Dearman and Truong, 2009, p. 99], where users can use their phone's keypad to issue commands: "*a user is able to watch a particular YouTube video, but also has the added ability of controlling audio/video playback. The user presses '5' on their mobile phone to pause the video . . .*"

Selecting an object or item from a set of related items is also a frequently used feature, as the following examples show. The e-Campus system Davies et al. [2009, p. 155] provided a Bluetooth naming based interaction mechanism for selecting a song to play: "*By subsequently changing their device name to 'ec juke <song id>' the selected music track will be added to the queue of songs to be played.*" In this case users explicitly enter the action to be performed (i.e., 'juke') and the item on which the action should take place (i.e., the song id). More often, the action is implicit and users just need to select the item to be acted on from a list presented by the public display, as in this Plasma Poster Churchill et al. [2004, p. 9] example, which used a touch-screen interface: "*... this was the last item posted to the Plasma Poster Network, and the display cycle is about to begin again. Readers can select any thumbnail to be displayed by pressing it.*" SMS is also frequently used for this purpose, as in Locamoda's Polls [LocaModa, 2010] application: "*Submit votes via text message for a poll of two (or more) 'choices.' Results are tallied in real-time and displayed on the screen both as a percentage and a bar graph.*"

5.3.2 Data entry

The data entry task allows users to input simple data (text, numeric data, or other formatted values) into a public display. Applications need to specify which type of data they wish to receive (text, numeric, dates, etc.), and possible bounds, or patterns, on the values they can accept. The interaction event that the application receives carries the user-submitted data. The data entry task is equivalent to the combination of the "quantify" and "text entry" tasks defined by Foley et al. [1980]. We chose to combine them because, when we abstract the interaction paradigm (instead of focusing on graphical manipulation interfaces), and consider the information exchange between user and application quantifying and entering text are essentially the same: users input values to the application. Cooper et al. [2007] also group quantify and text entry into data entry controls in their classification of desktop application controls.

Bounds

The bounds property of the data entry task refers to whether the application accepts free text from the user, or whether it imposes some pre-defined format to the data. For example, an integer number within a limit, or text that corresponds to a valid email address.

Unbounded text entry corresponds to Foley's text entry task, in which users are allowed to submit a string of text that does not need to conform to any specific rule. Text entry can be used to send messages, comments, and keywords, to the public display. In CoCollage, for example, users could send messages to the display by entering text in a web page: "*CoCollage users who are connected to the web site in the café may also send messages directly to CoCollage via a textbox near the upper right of any page. These messages become part of the history – and may thus be commented or voted on*" [McCarthy et al., 2009, p. 227]. Entering search terms is also a common use of text entry feature in public displays. The e-Campus system, for example, provided a Flickr search application: "*Users can access photos on Flickr by changing their* [Bluetooth] *device name to 'ec flickr <search term>'. For example 'ec flickr oranges' would cause photos retrieved using the search term 'oranges' to be displayed*" [Davies et al., 2009, p. 155]. SMS is also frequently used to allow user input. Locamoda's Jumbli application [LocaModa, 2010], for example, allows users to play a word game by texting their words. Touch interfaces can also be used for these interactions, supporting the traditional desktop entry controls such as sliders and dials, but also text-entry via onscreen keyboards as in the Digifieds system: "*Finally, if displays are touch-enabled the client provides an on-screen keyboard that allows users to create and send posts without using additional devices*" [Alt et al., 2011, p. 168].

Bounded data entry restricts the type, pattern, and range of the values that are entered. For example, in Visual Code Widgets, Rohs [2005, p. 511] described how visual widgets could be used with a camera phone: "*Unlike free-form input widgets, which provide 'unbounded' input, sliders are 'bounded' data entry widgets. The slider can be moved across a certain range, the selected value being proportional to the current slider position. . . . there are horizontal and vertical sliders. Input can either be continuous or discrete.*" Rating is another example of a bounded entry control, which usually allows users to enter a 1-5 value for an item. In CWall users could rate the content items presented by the public display: "*users can . . . touch the balance item to record a numeric rating*" [Grasso et al., 2003, p. 271]. Bluetone also allowed users to input bounded numeric values, in this case, using the mobile phone's keyboard: "*For example, the user will press '3' to increase the volume*" [Dearman and Truong, 2009, p. 99] .

5.3.3 Upload media

The upload task allows applications to receive media files sent by users. Applications should be able to specify the type of media they are interested in, but other

parameters such as the maximum file size, or maximum media duration (for video and audio) could also be of interest. The interaction event received just needs to specify the URL of the uploaded file.

Media type

The media type property of the upload task indicates the type of media file being uploaded: image, video, audio, html, and many other types of office documents. In JoeBlogg [Martin et al., 2006] for example, images were used to create an artistic composition on the public display. In other cases, images were used as free-hand comments to existing content, creating a discussion thread, as in the Digital Graffiti project [Carter et al., 2004, p. 1207]: "*we are experimenting with a system that allows individuals to attach digital graffiti annotations to publicly posted content.*" Audio and video are also often used media types. In the Dynamo for example, students could upload a variety of media files into the surface, including video and music files: "*During the two-week deployment, the use of Dynamo varied considerably: students displayed and exchanged photos, video and music, which they had created themselves or brought in from home*" [Brignull et al., 2004, p. 52].

Media location type

The media location type property of the upload task refers to the original location of the media. In many cases, the public display system accepts content that is stored in a personal device such as a mobile phone or even an USB pen drive. In these cases content is sent directly to the public display by attaching the pen drive or by transferring the file via Bluetooth OBEX or via a custom mobile application. In the Hermes Photo Display, for example users could transfer photos from their mobile phones to the display using OBject EXchange (OBEX): "*This version of the Hermes Photo Display also enables a user to . . . use her mobile phone's built-in 'picture' application in order to send a picture to the photo display over Bluetooth*" [Cheverst et al., 2005, p. 48]. In JoeBlogg, users would send personal pictures stored in their mobile phones via MMS. In Dynamo, users would simply attach their USB pen drives to the display to copy the media files into a shared space. Email has also been explored for uploading in the Plasma Posters display system: "*we discovered that providing an email interface for sending content to the posters resulted in a significant increase in postings. Posted content can be images and movies (sent in email as attachments), formatted text and URLs*" [Churchill et al., 2004, p. 8]. In other cases, however, users don't actually have a copy of the content in a personal device, but know the respective address. In these cases, the display receives a reference to the content, instead of the content itself. WebWall, for example, accepted Uniform Resource Locators (URLs) of media files to play in the public display: "*there are other service classes that are better defined first over the Web-client: Video and picture galleries (service class Gallery) can be used to display multimedia content by composing URLs of the media to display*" [Ferscha et al., 2002, p. 3].

5.3.4 Download media

The download task allows users to receive a content item from the display and store it in a personal device or account for later viewing or reference. The interaction event received by the application can simply be an acknowledgement that the file was, or is about to be, downloaded.

Media type

The media type property is analogous to the media type property of the upload task. Just as in upload task, various media types may be provided by a display system and made available for users to download. The Hermes Photo Display, for example, allows users to "*use the interface on the Photo Display to select a picture and then receive this picture onto her phone via Bluetooth*" [Cheverst et al., 2005, p. 48]. Videos are also a common media type that users may want to download. In ContentCascade, for example: "*The display is playing trailers of upcoming movies. Bob sees the Shrek movie and decides 'I like that!' and wants to download the movie clip. He pulls out his Bluetooth enabled cell phone*" [Raj et al., 2004, p. 375].

Media location type

The media location type property is analogous to its counterpart in the upload task: content to be downloaded can either be already publicly available and the display system just provides the address on the web, or it can be content stored internally at the display system that is transferred to the user. For example, in ContentCascade users could also receive URLs in their mobile device: "*However, since he had accepted to receive small meta-information about the upcoming movies, the ContentCascade mechanism downloaded a URL from where he can later get more information about Antz*"[Raj et al., 2004, p. 375]. In Hermes Photo Display, however, the photos were stored internally in the display system and downloading involved establishing a Bluetooth connection between the display and user's mobile device to transfer the photo.

Target device

The target device property refers to where the downloaded content is transferred as a result of the interaction. Downloaded media can be received in a variety of destination devices or personal accounts, using various communication protocols. Content can be downloaded to a personal mobile device, for example, using SMS as in Locamoda's Community Board application [LocaModa, 2010]. OBEX is another protocol that can be used for receiving media files as in the Hermes Photo Display. There are also examples of display systems that use custom mobile applications and communication protocols for receiving files on the mobile device. Touch & Interact

for example consists of a public display and a mobile application in which "*the user interacts with a picture board by touching the picture with the phone and in response, the picture moves from the dynamic display to the phone*" [Hardy and Rukzio, 2008, p. 246]. Users can also receive files in a USB pen drive, as in the Dynamo system [Brignull et al., 2004], or download to their mobile device by scanning a QR code as in Digifieds: "*Digifieds can also be taken . . . by scanning a QR-code with the phone*" [Alt et al., 2011, p. 168]. Mobile devices, however, are not the only possibility for receiving media files. A popular approach is to allow users to receive the content in their email. In the Digital Graffiti project [Carter et al., 2004, p. 1209] for Plasma Posters for example: "*Later, Jane is passing by the Plasma Poster and sees all the annotations that have been posted over her original content. She is amused to discover her post has caused so much response and debate and forwards the recommended URL to her home email so she can read it later.*" Finally, a less common but also possible solution for specific media types it to allow users to print the content. Also in the Plasma Posters project, users could print a displayed item directly from the public display: "*along the bottom of the new interface there are three buttons: 'Show All' button to show a list of items in the presentation sequence, 'Print' to print the currently displayed posting*" [Churchill et al., 2004, p. 10].

Target person

The target person property refers to whether the content is transferred to the interacting user, or to another person. Often, users want to download content for themselves, in order to get an offline copy of the content or as a reference to view later. However, there are also cases where a user wants to download a content item and forward it to another person. Plasma Posters, for example, allows content to be forwarded to others: "*Items can be forwarded to others, or to oneself for reading later at a personal computer*" [Churchill et al., 2004, p. 9] The same could be done with Digifieds, which allowed users to send content via email: "*Digifieds can also be taken away . . . by sending them to an email address*" [Alt et al., 2011, p. 168].

5.3.5 Signal presence

The signal presence task allows the application to be notified about events regarding the presence of users in the vicinity. Although all interactions with a display system can be used to determine the presence of users (if a button was pressed in a touch screen, it means that there was someone there), in this section, we are considering only those interactions specifically designed for determining the presence of users.

Location disclosure

The location disclosure property refers to whether the user manually sets his presence, or whether it is sensed automatically by the display system. The manual form

corresponds to a check-in interaction where users decide when they would like to announce their presence to the public display. A check-in can be accomplished through a number of different ways, for example using hardware that reads a personal identity card, or a personal mobile device. Magnetic card readers, Radio-Frequency Identification (RFID) readers, or even Bluetooth detection can be used to accomplish this type of check-in. Russell and Gossweiler [2001] for example, used personal cards that users could swipe on a card reader in the BlueBoard display to access their personal data (in this case, the feature worked as a login because it allowed access to personal information, but it could also be used for check-in): "*The net effect is that a user can 'log in' by simply swiping their badge at the display, getting rapid access to their content*" [p. 357]. Check-in can also be accomplished solely through software, for example through a mobile application or web page. CoCollage provides a web check-in to its users: "*The presence of users is established via an explicit 'check-in' through the use of . . . a button on a web page that is enabled only when the user's computer is connected to the wireless Internet router in the café*" [McCarthy et al., 2009, p. 226]. The automatic sensing of users around the display can itself be subdivided into three forms according to the level of information sensed, as we have defined in the previous chapter: presence detection, characterisation, and identification. Presence detection corresponds to an on/off detection where the display either detects someone (but not who, or how many) in its vicinity, or detects no one. This can be used to trigger a change in the display's mode from an ambient mode to a more interactive mode, as in the Aware Community Portals: "*a weather map triggered by the user walking by vs. a news article shown when the user lingers to browse*" [Sawhney et al., 2001, p. 68]. Presence characterisation corresponds to a more rich detection, where the display is able to sense more information about the people in the vicinity, such as how many, their position, where they are gazing, the estimated age, etc. The CWall display system used computer vision techniques to infer if people were standing in front of the display, and looking at it: "*In the case of the CWall we decided to see if we could improve the utility of the display by differentiating between people passing by and people standing in front of the display actively reading the content*" [Grasso et al., 2003, p. 274]. In presence identification, the display is able to identify users and, possibly, associating personal information. This can be used to provide personalised content on a public display as in the Proactive Displays: "*When attendees are near a proactive display, content from their profiles can be shown*" [McDonald et al., 2008, p. 3].

Location verification

The location verification property indicates whether the system can verify that the user is really where he says he is. In the automatic presence sensing, the system can have stronger guarantees that users, or at least their devices, are in the vicinity of the display. Sensors are assumed to be located near the display, and they usually have a limited detection range. The same happens in the manual presence sensing that makes use of personal cards or other physical items that are detected by a card reader or other sensor near the display system. Even if the check-in is accomplished via software, the user's location can still be verified. CoCollage, for example, uses the local Wi-Fi network to verify the user's location: "*The presence of users is*

established via an explicit 'check-in' through the use of . . . a web page that is enabled only when the user's computer is connected to the wireless Internet router in the café" [McCarthy et al., 2009, p. 226].

In many cases however, the user's location is not verified by the system. Most location based social networks such as Foursquare, Google Latitude, and Facebook Places provide mobile applications that allow to check-in in any place, without any system verification about the real location of the user. This is something that is normally accepted by people as part of the semantics of presence through these check-in procedures. Some public display systems take advantage of these existing location based networks. Locamoda's Check-in application [LocaModa, 2010], for example, "*leverages widely adopted location based applications such as Facebook Places and Foursquare to display relevant venue Check-In activity on venue digital displays.*" The Instant Places display network provides its own mobile client with similar check-in semantics: "*Explicit session activation can be accomplished through a check-in mechanism available in our instant place mobile app*" [José et al., 2012, p. 3].

5.3.6 Dynamic manipulation

The dynamic manipulation task corresponds to continuous interactions were users manipulate graphical objects in the application's interface. Dynamic manipulation represents tasks in which it is fundamental to provide a direct-manipulation style, particularly "*rapid, incremental, reversible operations whose impact on the object of interest is immediately visible*" [Shneiderman, 1983, p. 64]. In this task, the application receives a continuous, timely, flow of information, which it can then map to various graphical objects.

Type of manipulation

The type of manipulation property refers to the type of action performed by the user and the information received by the application. We defined four values for this property: cursor, joystick, keyboard, and skeleton/silhouette input. Although their names may suggest physical devices, these types of input may be generated by highly diverse mechanisms (for example, joystick input can be generated by a physical joystick button, but also by specially arranged keyboard keys, or even by a virtual multi-touch joystick).

Cursor events carry information about the position and velocity of multiple cursors on a 2D or 3D environment, and can be used for mouse, multi-touch, or even 3D interactions. For example, Dynamo, allows users to "carve" rectangular regions on the display to appropriate them for individual use. This is done by simply "drawing" a rectangle using the mouse: "*Carves can be created by a mouse drag gesture to create privately owned areas in which only the user and their chosen members can interact*" [Brignull et al., 2004, p. 50]. In CityWall, users use multi-touch gestures to move,

scale, and rotate photos: "*Moving, scaling and rotation of content . . . follows direct manipulation principles: a user can grab an image by putting a hand on it. The photo follows the hand movements when the user shifts her hand. Rotation and scaling are possible by grabbing the photo at more than two points (e.g. by two hands or two fingers of the same hand) and then either rotating the two points around each other or altering their distance*" [Peltonen et al., 2008, p. 1287].

Joystick events carry information about the angle and state of joystick/gamepad buttons. In Point & Shoot users could use their camera phone as a mouse or joystick and select, rotate and move jigsaw puzzle pieces: "*The phone display is used to aim at a puzzle piece on a large display. . . . Pressing the joystick indicates selection and a visual code grid flashes on the large display to compute the target coordinates*" [Ballagas et al., 2005, p. 1202]. In the Vodafone Cube [Ydreams, 2003], users could dial a phone number and control various games, including a car racing game, using the phone's keyboard as a joystick.

Keyboard events carry information about a succession of key presses in a physical or emulated keyboard. In MST/MSE for example, the mobile client supported keyboard input: "*transmits all keypress events from the phone's keypad back to the MST server in real time*" [Toye et al., 2005, p. 63]. Remote Commander is another example were keyboard input was important: "*This allows . . . the PalmPilot . . . input to emulate the PC's keyboard input. The important point is that this works with all existing PC applications, without requiring any modifications to the applications themselves*" [Myers et al., 1998, p. 287]. It should be noted that keyboard events do not necessarily mean that the application is interested in receiving text data (a keyboard could be used to play music, for example).

Skeleton/silhouette events carry information about the position of the user's body joints and/or about the user's silhouette. This type of input has recently gained wide exposure due to the Kinect depth camera controller, but it can also be accomplished with other sensor technologies such as body suits, stereo cameras, or motion capture systems. This kind of input has been mostly explored in artistic interactive projects, but it has also been applied successfully in public display systems. Müller et al. [2012] in project Looking Glass, used a Kinect to extract user's silhouettes and provide a gaming experience in a public display of a shop window, by allowing users to wave their arms to push balls on the display.

5.4 Design Space for Interaction Controls and Mechanisms

Based on the interaction tasks described in the previous section it is possible to frame a new design space for interaction with public displays around those tasks. In this section, we analyse how the interaction tasks could be mapped to interaction mechanisms and what interaction controls can be derived from them.

5.4.1 Mapping between interaction tasks and mechanisms

The first step in our analysis is to explore the relationship between interaction mechanisms and the set of interaction tasks. This mapping provides a comprehensive view of how different mechanisms can be used to support a given interaction task and also of how the various interaction tasks are represented in the various concrete system implementations from the research literature.

To facilitate the mapping, we have created a spatial layout that shows how the different interaction tasks can be implemented with various interaction mechanisms. This mapping is inspired by the spatial layout from Ballagas et al. [2008], but we omitted the attributes dimensionality and relative vs. absolute, which were not relevant for our analysis, and we added a new interaction distance attribute. The resulting layout, depicted in Tables 5.2, 5.3, 5.4, and 5.5, represents how the interactive displays from the literature are distributed between the interaction tasks and the mechanisms that support those tasks. We have plotted each interaction mechanism that appeared in the surveyed interaction public display systems.

The reference to each interactive display system is complemented with a classification of the interaction along three secondary dimensions: interaction style, feedback, and interaction distance. The interaction style can be direct, or indirect: "*in direct interactions, the input actions are physically coupled with the user-perceivable entity being manipulated, appearing as if there was no mediation, translation, or adaptation between input and output. In indirect interactions, user activity and feedback occur in disjoint spaces (e.g., using a mouse to control an on-screen cursor)*" [Ballagas et al., 2008]. Feedback can be continuous, or discrete: "*continuous interactions describe a closed-loop feedback, where the user continuously gets informed of the interaction progress as the subtask is being performed. Discrete interactions describe an open-loop feedback, where the user is only informed of the interaction progress after the subtask is complete*" [Ballagas et al., 2008]. For the purpose of this analysis, we are only considering shared feedback shown on the public display itself, and not the individual feedback that may be generated on the mobile device for example, which may be considerably more flexible. In the interaction distance we distinguish between close-up and remote interaction. Close-up interaction requires users to touch the display with their body (often fingers and hands) or with a handheld device, whereas in remote interaction users can interact at a distance. This dimension has implications on the physical placement of the public display (close-up interaction require displays that are at arms reach), or on which interaction mechanisms are suitable for an already deployed public display. Each entry in the table is labelled with an ordered set of letters corresponding to the possible values for the three dimensions: **D**irect/**I**ndirect, **C**ontinuous/**D**iscrete, **C**lose-up/**R**emote.

We now use Tables 5.2 through 5.5 to analyse how four common categories of interaction mechanisms – touch-screen based public displays, interaction via mobile devices, device-free interaction, and desktop-like interaction – can be used to support the various interaction tasks, using concrete examples from the design space. The references for the various display systems can be found in appendix A.

Interaction based on touch-screens

Table 5.2: Mapping between interaction tasks and touch-screen based interaction mechanisms.

Interaction mechanism	Interaction task					
	Select	**Entry**	**Upload**	**Download**	**Presence**	**Dyn. Manip.**
Touch screen	DDC:Plasma Posters; Hermes Photo Display; OutCast; Blueboard; Jukola; AgentSalon; Digifieds; Spalendar; iSchool; FizzyVis; Semi-public displays; Cwall; UBI-hotspot; Vista	DCC: Plasma Posters; Digifieds; Spalendar; Cwall		DDC: Plasma Posters; iSchool; Cwall	DDC: UBI-hotspot	DCC: Plasma Posters; Blueboard; FizzyVis; CityWall; Semi-public displays
Touch screen + Bluetooth OBEX			IDC: Hermes Photo Display	IDC: Hermes Photo Display		
Touch screen + Mobile application			IDC: Digifieds	IDC: Digifieds		
Touch screen + printer				DDC: Plasma Posters		

Touch-screens can be used without the need for any other device so they are a good solution for walk-up-and-use, close-up interaction displays, provided that they can be placed in a location that allows users to directly touch it. Touch-screens can be used to support most of the interaction tasks for public displays. Select, entry, and dynamic manipulation tasks are obviously well supported. Download media can be accomplished in a limited way by forwarding the content to a personal email address entered using a virtual keyboard, or by selecting a username from a list in case the display system has registered users. Signalling presence can be supported in a manual way as in the Ubi-hotspot system were users would touch the display to cause a transition to an interactive mode. None of the public display systems we surveyed used a touch-screen (without any other device) for uploading media, although one could conceive that it could be used for uploading by entering the public address of a file using a virtual keyboard.

However, touch-screens in conjunction with other devices can provide richer interactive experiences and better support for the full range of interaction tasks. The download and upload tasks in particular can take advantage of personal mobile devices for an easier transfer of media files by using an approach similar to the one used by the Hermes Photo Display with Bluetooth OBEX transfers, or the Digifieds approach with visual and textual codes. Signalling presence can also be made more flexible by incorporating personal card readers into the display as in the BlueBoard or Ubi-hotspot display systems.

Interaction based on personal mobile devices

Table 5.3: Mapping between interaction tasks and interaction mechanisms based on personal mobile devices.

Interaction mechanism	Interaction task					
	Select	**Entry**	**Upload**	**Download**	**Presence**	**Dyn. Manip.**
Bluetooth detection					IDR: BluScreen	
Bluetooth naming	IDR: e-Campus	IDR: e-Campus; Instant Places; Bluemusic			IDR: Instant Places	
Custom mobile personal device	IDR: Pendle ICR: VisionWand	ICR: VisionWand			IDR: Pendle; AgentSalon	ICR:VisionWand
DTMF						ICR: Vodafone Cube
DTMF + Bluetooth mobile phone	IDR: Bluetone	ICR: Bluetone				ICR: Bluetone
MMS			IDR: JoeBlogg			
Mobile application	DDR: Jukola DCR:C-Blink IDR: Mobilenin	IDR: Digital graffitti; Hello.Wall; Cwall; Mobile Service Toolkit	DDR: C-Blink IDR: Digital graffitti; Cwall	DDR: C-Blink IDR: Hello.Wall; Mobile Service Toolkit	IDR: Hello.Wall; Mobile Service Toolkit	ICR:Digital graffitti; Remote Commander; Mobile Service Toolkit
Mobile application + Bluetooth mobile phone			IDR: Publix	IDR: ContentCascade; Publix	IDR: Publix	ICR:Publix
Mobile application + Camera phone	ICR: Sweep					DCR: Sweep ICR:Jeon et al.
Mobile application + Camera phone + visual codes	DDR: Point & Shoot IDR: Visual code widgets; Mobile Service Toolkit	IDR: Visual code widgets		DDR: Digifieds		ICR:Jeon et al.
Mobile application + NFC phone + NFC display	DDC:Touch & Interact; Hello.Wall DDR: Hello.Wall		DDC: Touch & Interact	DDC: Touch & Interact	IDC: Touch & Interact	DCC: Touch & Interact
Personal id card					DDC: Blueboard; UBI-hotspot IDR: CoCollage; Proactive displays; GroupCast	
SMS	IDR: Locamoda	IDR: Webwall; Locamoda	IDR: Locamoda			

Remote interaction can be accomplished through many interaction mechanisms. A popular approach is to provide a custom mobile application (usually for smartphones) for interacting with the display. Some mobile applications require specific mobile hardware to function properly, such as having a camera, Bluetooth, Infrared, Near Field Communication (NFC); other mobile applications require the display to be able to generate visual codes. Most of these mobile applications provide an indirect interaction style with the public display where the user's focus is on the mobile device interface. Some however, turn the mobile device into a tracked object as in C-Blink and Point & Shoot, or into a viewport into the public display interface, as in the Visual Code Widgets, which provides a direct interaction style but also requires users to stand closer to the display and hold the device in front of it. These solutions cover the complete set of interaction tasks for public displays, allowing users to have a rich interaction experience with a public display, remotely.

Another frequent alternative is to use the standard processing and communication features of mobile devices, without the need to install additional applications. Bluetooth detection, Bluetooth naming, SMS, Multimedia Message Service (MMS), and Dual-Tone Multi-Frequency (DTMF), have been used to support different interactive features. Although these interaction mechanisms do not support all the interaction tasks, they may still be a viable solution for specific interactions. Bluetooth has the advantage of being widely supported by mobile devices and cost-free for the user. Bluetooth detection, i.e., scanning the area near the public display for Bluetooth enabled devices and reading their ids, can be used to estimate the number of people that are present and to determine which devices have been near the display and when, as in the BluScreen system. SMS and Bluetooth naming, i.e., interpreting the Bluetooth name of the device as commands to the display system, can be used for selection and data entry, even if in a simple way, as in the e-Campus, and Instant Places systems. MMS can be used to upload or download pictures and other media files. The downside of both SMS and MMS is that require users or display system to incur in costs (which can be considerable for MMS) when sending the messages. Finally, DTMF can be used to support selection and data entry tasks as in the Bluetone system, and dynamic manipulation as in the Vodafone Cube. DTMF also has costs for users, unless it is done over Bluetooth as in Bluetone.

Device-free interaction

Table 5.4: Mapping between interaction tasks and device-free interaction mechanisms.

Interaction mechanism	**I**nteraction task					
	Select	**Entry**	**Upload**	**Download**	**Presence**	**Dyn. Manip.**
Camera	IDR: Aware Community Portals	ICR: Beye & Meier			IDR: Aware Community Portals; ReflectiveSigns; Cwall; Cwall; UBI-hotspot; SmartKiosk	

...continued.

Interaction mechanism	Interaction task					
	Select	**Entry**	**Upload**	**Download**	**Presence**	**Dyn. Manip.**
Camera (Kinect)	DCR:MAID				IDR: Code Space	DCR: Looking Glass; Code Space; MAID
Camera (MoCap system)	IDR: Interactive Public Ambient Displays; Spalendar	ICR: Spalendar; Spalendar			IDR: Interactive Public Ambient Displays; Spalendar	DCR: Spalendar ICR: Interactive Public Ambient Displays; Spalendar
Electro-magnetic sensor	ICR: Gesture Frame					ICR:Gesture Frame
Sound (finger click)	IDR: Gesture Frame					

Device-free interaction with public displays can be accomplished with cameras (standard web cameras, or depth sensing cameras such as the Kinect) and computer vision techniques. Device-free interaction has the advantage of providing a walk-up-and-use interaction and not requiring users to directly touch the display, allowing it to be positioned in a way that allows multiple users to see and interact with it simultaneously. With devices such as the Kinect, it can be a viable solution in scenarios such as shop windows where it can also be used to detect and attract passers-by. Selection, data entry, presence, and dynamic manipulation tasks can be accomplished with these interaction mechanisms. Although device-free interaction by itself does not support download and upload tasks, it is possible to use additional devices for this purpose as in Bragdon et al. [2011].

Desktop-like interaction

Table 5.5: Mapping between interaction tasks and desktop-like mechanisms.

Interaction mechanism	Interaction task					
	Select	**Entry**	**Upload**	**Download**	**Presence**	**Dyn. Manip.**
Desktop application	ICR: Notification Collage	ICR: Notification Collage		ICR: Notification Collage		ICR: Notification Collage
Email	IDR: WebGlance	IDR: Locamoda; WebGlance	IDR: Plasma Posters; Cwall	IDR: Digifieds		
Instant messaging	IDR: WebGlance	IDR: WebGlance				
Mouse & Keyboard	ICR: Dynamo; Opinionizer	ICR: Dynamo; Opinionizer				ICR:Dynamo

...continued.

Interaction mechanism	Interaction task					
	Select	**Entry**	**Upload**	**Download**	**Presence**	**Dyn. Manip.**
Mouse & Keyboard + USB stick			ICR: Dynamo	ICR: Dynamo		
Web application	ICR: CoCollage	IDR: CoCollage	IDR: CoCollage; Webwall; Digifieds	IDR: Digifieds	IDR: CoCollage	

It is also possible to support all the interaction tasks through desktop-like interfaces. One possibility is to provide a custom native or web application that enables users to interact with the public display. All the interaction tasks can easily be supported in this manner. For example, Notification Collage, CoCollage, and Digifieds, provide applications that mediate the interaction with the public display itself. It is also possible to provide a desktop-like interaction where the public display application itself behaves in a similar manner to a desktop application as in the Dynamo display where users simply pick up a mouse and keyboard to interact with the display. As in the case of mobile devices, it is also possible to use standard desktop applications such as email or instant messaging to interact with a public display system as in Plasma Posters, CWall, WebGlance, and other systems. Although it is not possible to support all the interaction tasks (for example, dynamic manipulation is not possible with email or instant messaging), it can still be a plausible solution in some cases, as it leverages on existing applications thus obviating the need to install additional software.

5.4.2 Interaction Controls

Interaction controls provide the next element that is needed to enable applications to benefit from the interaction tasks that we have identified. The high level of abstraction that is associated with the interaction tasks needs to be instantiated into specific controls that can be integrated into applications to support interaction. A control can still maintain independence from the concrete interaction mechanism, but it refines the specific information being exchanged, defines additional optional and mandatory parameters, and can manage input in a specific way before triggering the interaction event. Just as we have several types of data entry controls for desktop applications, public display applications also need different controls for the same interaction task. These controls will form the main components that applications will use to provide their interaction features.

As part of our analysis of the interaction tasks, we sought to identify a representative set of controls that could illustrate how the various tasks could be instantiated. To define the set of controls we have considered the need to include all the interaction tasks, the key variations within each task and also what seemed to be the most

common forms of interaction in the research literature. Still, this is not meant to be an exhaustive listing (as Foley et al. [1980, p. 20] put it "*their number is limited only by one's imagination*"), but it provides a good overview and comparison of the possibilities for implementing the various tasks for public display interaction and it should provide a relevant starting point for designing interaction systems for public display applications. The relevance of these specific controls will ultimately depend on their real world usage, which may lead to the emergence of totally new controls, changes to existing ones and the disappearance of others.

In this description, we focus on the interaction events and information processing associated with the controls. We leave out the graphical representation and feedback aspects usually associated with widgets in desktop systems, as these would be very dependent on the specific implementation of the interaction system. Table 5.6 provides a list of possible controls for the various tasks.

Table 5.6: List of possible controls for supporting the various interaction tasks.

Task	**C**ontrol	**D**escription
Select	Action	A generic action control, which causes the application to execute an action; similar to a desktop button. Triggers an event that identifies the action.
	Option List	A generic list control, which presents several options and allows users to select one (or more). Triggers an event with the selected option when a user makes a selection.
	Vote	Time based action control with a list of alternatives that waits for interactions during a pre-defined period of time. Triggers an event with the most voted alternative, when the time expires.
Data entry	Unbounded text	Allows users to input any string of text. Triggers an event with the input string.
	Bounded text	Supports various text patterns (such email addresses, phone numbers, dates, etc.). Triggers events with input string that conform to the specified pattern.
	Numeric entry	Generic numeric entry control allows users to input numbers, possibly with lower and upper limits, integer or floating point. Triggers event with the input number.
	Rate	Allows users to rate content. May support various formats such as different scales, discrete/continuous rating scale. Triggers event with the input rating.
Upload media	Generic upload	An upload control that accepts any media file, possibly with a parameter to limit the total file size. Triggers an event with the location of the uploaded file.
	Video upload	Accepts only video files. Allows applications to specify the maximum duration of the video, and supported video formats. Automatically converts between unsupported video formats to supported ones, for example, or simply does not allow unsupported formats.
	Image upload	Accepts only images. Allows applications to specify the maximum/minimum image size, and supported image formats. Automatically converts images that do not conform to the specified size and format restrictions.
	Audio upload	Accepts audio files. Allows applications to specify the supported formats and maximum audio duration.
Downl. media	Download	Allows application to specify the media type and location of a content item that users can download. Triggers an event that identifies the downloaded file.

...continued.

Task	**C**ontrol	**D**escription
	Share	Allows users to share a content item with other people. Triggers an event that identifies the shared file.
Signal Pres.	Check-in	Allows users to explicitly signal their presence near a display. Optionally, the interaction event can carry the location verification status, allowing the display system to give more weight to check-ins with verified locations, for example.
	Presence	Signals the presence of users obtained implicitly from sensors. The information carried on the interaction event may vary, depending on the concrete types of sensors available, but we can generally categorize it according to the levels of information that are sensed: presence detection, characterization, and identification.
Dyn. Manip.	Cursor	Allows users to dynamically interact via (multiple) cursor positions. The interaction event is a continuous flow of cursor positions.
	Joystick	Provides joystick information (direction, gamepad button states), for gaming purposes. The interaction event is a continuous flow of direction and button states.
	Keyboard	Provides keystroke events. The interaction event is a continuous flow of key presses.
	Skeleton	Provides positioning of body joints, and/or user's silhouette information (full or partial body parts). The interaction event is a continuous flow of skeleton/silhouette data.

Together, the mapping between the interaction mechanisms and interaction tasks, and the characterisation of the controls that support those tasks, forms a design space for interaction abstractions for public displays that can be used in several ways. A designer of an interaction toolkit for public display applications can use the design space to understand the kinds of high-level controls that the toolkit should provide to application developers and which interaction mechanisms can support those controls. For someone deploying a public display system, the design space can be used help make informed choices regarding the interaction mechanisms that should be deployed in order to support a specific set of interaction tasks. It can also be used to determine the interaction characteristics those interaction mechanisms impose. Application developers can use the design space to understand which controls can be supported by public display systems and decide how the interaction features of their applications can be implemented using those controls. Additionally, the various concrete examples of display systems listed in the design space can also be used as reference, or design patterns, for the implementation of the various controls in the interaction toolkit.

5.5 Conclusion

We have presented a study about interaction tasks and controls for public display applications, grounded on the existing descriptions of concrete interactive display

systems available in scholarly publications. We have characterised six high-level interaction tasks focused on the specificities of public display interaction, more specifically select, data entry, upload, download, signal presence, and dynamic manipulation. These tasks represent a classification of the major types of interaction between users and public displays; we have also identified various types of concrete interaction controls that may enable those interaction tasks to be integrated into applications for public displays. These controls constitute a first step towards a list of controls that may compose future interaction toolkits for public displays; we have also organised the various interaction mechanism for public displays in a design space adapted from Ballagas et al. [2008] that sketches a mapping between the high-level abstractions provided by the interaction tasks that have been identified and the concrete interaction mechanisms that can be implemented by those displays.

With the interaction tasks, the mapping between tasks and mechanisms, and the interaction controls, we have a tool to structure an interaction system for public display applications. This is a valuable tool for allowing application developers to make more informed decisions on the types of controls that they would need, considering for example the applications goal but also the envisioned interaction modalities.

Chapter 6

The PuReWidgets Toolkit – A Widget-based Interaction Abstraction for Public Displays

Contents

6.1 Introduction

In this chapter we present our solution for an interaction abstraction toolkit for public display applications, presenting the main concepts, architecture, and implementation. Our solution is primarily inspired on the widget concept: a programming object that provides high-level interaction events to applications, hides the low-level actions needed to interact with the application, has a graphical representation, provides input feedback, and can be extended by programmers to create other widgets.

6.1.1 Main features

The PuReWidgets toolkit provides a number of important features for interactive public display applications:

Multiple, extensible, widgets The toolkit incorporates various types of interaction widgets, supporting the previously identified interaction tasks[1]: select, data entry, upload, download, and check-in. Existing widgets can be customised and composed into new widgets, and completely new widgets can be created by application programmers.

Dynamically generated graphical interfaces The toolkit automatically generates graphical user interfaces for desktop and mobile interaction with public displays. It also generates Quick Response codes (QR codes) for user interaction through camera equipped mobile devices.

Independence from specific input mechanisms and modalities The toolkit supports several interaction mechanisms such as Short Message Service (SMS), Bluetooth naming, OBject EXchange (OBEX), email, touch-displays, in addition to the already mentioned desktop, mobile, and QR code interfaces.

Asynchronous interaction The toolkit supports asynchronous interaction, allowing applications to receive input events that were generated when the application was not executing on the public display.

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

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

[1] We left out the dynamic manipulation task at the moment, because it imposes greater restrictions on the supported interaction mechanisms, and requires very application-specific solutions for handling multiple user interaction.

6.1.2 Design decisions and assumptions

While designing the PuReWidgets toolkit, we targeted it at web-based public display applications, even though the central concepts could be applied to "native" applications running over standard desktop Operating Systems (OSs). Targeting the web platform has a number of advantages:

- Applications can easily take advantage of the extensive web content and content access Application Programming Interfaces (APIs) that already exist.
- Applications are easier to distribute, update and maintain. Updates are immediate and display owners[2] will not have to reinstall the application.
- Applications are naturally multi-platform and independent of the OS running on the public display.
- Because there is no need to copy files to install the application, configuring the same application on various public display can also be made easier by using a single web interface.

Web applications also have some limitations, namely regarding the restrictions on the availability of various types of machine resources, such as graphical processing, file system access, and local input resources. However, given the recent advances in browser technologies and the current trend towards wider use of the web as an execution environment, we can expect that the limitations faced by web applications will continually decrease.

Application life-cycle

Although PuReWidgets is not tied to a particular application life-cycle, to facilitate the description of the architecture, we assume that the usual life-cycle of a public display application is the one depicted in Figure 6.1, and described next.

An application is initially developed and *deployed* by an application developer. Deploying an application means that it will be hosted on a third-party web server and publicly accessible from that point forward, via a Uniform Resource Locator (URL).

Once an application is deployed, display owners are able to use it in their displays. In the PuReWidgets context, it is not important how display owners find applications. In order to use an application in a display, the display owner will have to *associate* it with a particular display. This step may entail several things such as configuring the display scheduler software with the address of the new application, specifying the amount of display time and screen area the application should have, and possibly

[2]We use the term display owner to refer to the person that is responsible for maintaining the public displays in a specific location, setting up and configuring their content.

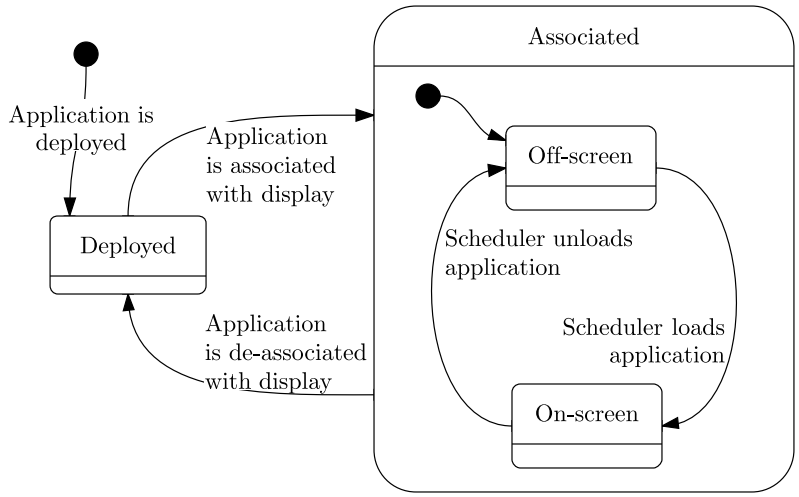


Figure 6.1: General life-cycle of a public display application

configuring the application so that it displays content that is appropriate for the particular place where the display is located.

Once an application is associated with a display, the display scheduler can decide to give the application display time by putting it *on-screen*.[3] Putting an application on-screen means loading the web page of the respective application in a browser component. Putting an application *off-screen* means removing it from the display, probably shutting it down by closing the web page, unloading it from the browser component. (Although PuReWidgets also works with applications that are simply taken to a background state, hidden from view.) We make no assumption as to the type of scheduler that a particular display will use. Scheduling could use a simple time-based algorithm, based on a pre-defined timetable of display time for each application, or a more complex event-based algorithm, which dynamically gives applications display time based on external events such as user interactions.

API design

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

[3] We borrow the terms on-screen and off-screen from the cinema terminology which refer to characters that are present on a scene, but are not seen by the audience. We apply the terms to both applications and widgets.

to produce separate interface descriptions with the only purpose of being used for generating the web Graphical User Interface (GUI).

Flexible. We wanted an easy to program system, where developers 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.

6.2 Architecture

The PuReWidgets system was designed to support displays in various independent administrative places, running various applications developed by third-party developers. Figure 6.2 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 6.3). When the application is on-screen, the PuReWidgets library receives input events from the PuReWidgets service, and triggers the appropriate high-level events on the application. The development process of a public display application that uses PuReWidgets is similar to the development of a regular web application: developers include the PuReWidgets (object-oriented) library in their projects and use the available functions of the library to code the application, instantiating widgets and registering callback functions for interaction events. These callbacks correspond to high-level interaction events and are triggered by the widget instances after an interaction event is detected and processed by the widget. Developers then deploy the set of HyperText Markup Language (HTML), Cascading Styles Sheet (CSS), and Javascript files that compose their application to a web server.

6.2.1 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, email, etc. (I/O module). The IM exposes an HyperText Transfer Protocol (HTTP) Representational State Transfer (REST) service for submitting and receiving widget information and input events that is used by the PuReWidgets library. The IM is

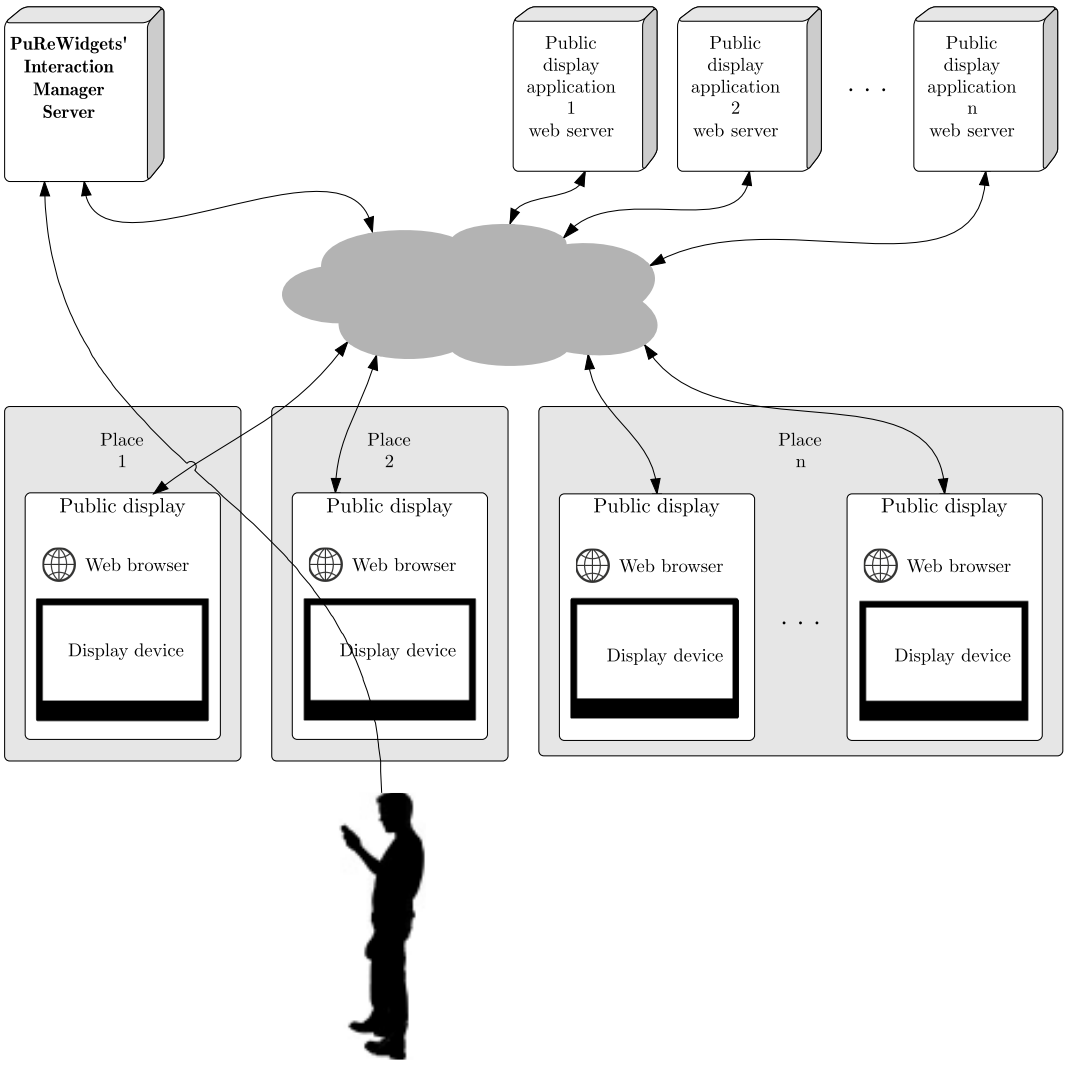


Figure 6.2: PuReWidgets' physical components diagram.

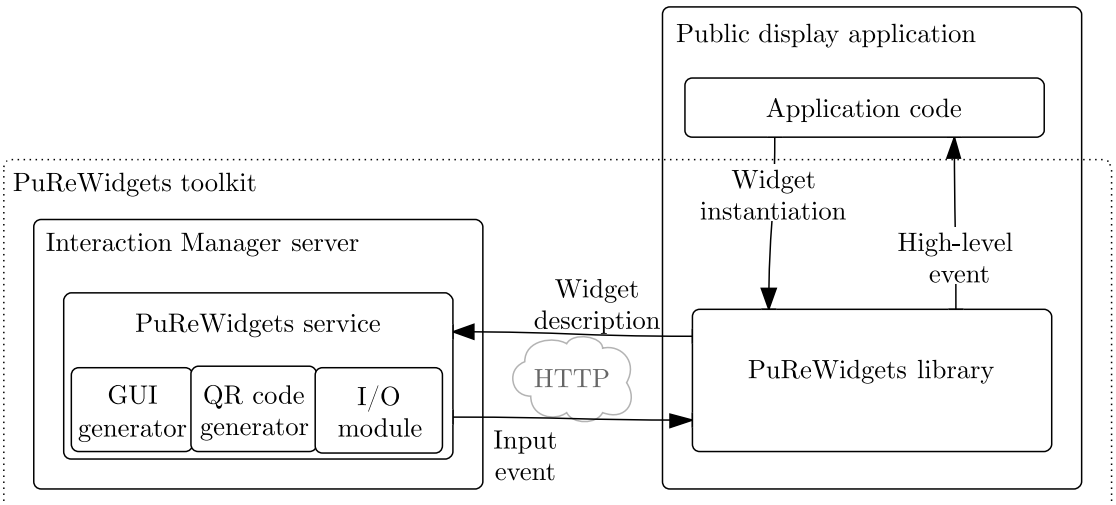


Figure 6.3: PuReWidgets' general architecture.

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 be something small like a specific cafeteria, with a single public display, or a wider place like a university campus, with various public displays. A single IM server can handle multiple independent places. Each place is identified by a unique *place id*, within the IM.

Application An *application* is a web application that uses the PuReWidgets library, which display owners will typically identify by its URL. Display owners may associate several applications with a single place. Each association is an application instance in the IM, identified by an *instance id*. The same application can be associated multiple times and given different instance ids in the IM (for example, the display owner may want to display the same application with different configurations in the same or in different displays).

Widget A widget represents an interaction feature of an application. Applications instantiate widgets at runtime, and give them unique *widget ids* (unique in the scope of the application). When widgets are instantiated by an application they must be registered, i.e., their description sent to the IM. The registration process itself is hidden from the application and is done by the PuReWidgets library.

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. Widgets have at least one widget option.

Reference code The IM assigns a unique (within the scope of a place) textual *reference code* to each widget option. These reference codes are human-readable identifiers to be used in text-based interactions (such as SMS or email 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 interaction 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.

6.2.2 PuReWidgets library

The PuReWidgets library is the toolkits' interface with programmers. It is a web client library[4] that offers programmers various widgets that abstract user interaction into high-level interaction events to applications. The PuReWidgets library transparently handles communication with the PuReWidgets service for various bookkeeping operations, including registering the widgets in the IM, receiving user input, forwarding input to the correct widget, and providing system-level input feedback on the public display. The library has various components (see Figure 6.4):

Widgets Applications mainly use the functions of the PuReWidgets library through its various widgets, instantiating, configuring, receiving interaction events, and deleting them.

Communications manager The communications manager component implements the various REST services provided by the PuReWidgets service of the IM, providing an interface for the rest of the library components.

Local storage The local storage component provides access to the local storage browser functions. This component providers higher-level functions than the ones directly available through Javascript and also abstracts the local storage into application- and widget-specific local storages so that applications and widgets do not have to deal with possibly conflicting names for their key-value objects.

Input feedback manager The input feedback manager component provides graphical popup panels for displaying input feedback and schedules and coordinates their visibility on the public display.

Widget manager The widget manager component is responsible for keeping a consistent state between the application's widgets and the IM database. It manages all the widgets instantiated by an application, guaranteeing that they are registered in the IM. It also receives and routes input events from the IM to the correct widget instance.

6.3 Widgets and Events

PuReWidgets provides support for the various interaction tasks defined in chapter 5, in the form of widgets.

[4]In reality, PuReWidgets offers also a library for server-side code, but for simplicity's sake, at this point we focus on the web client library.

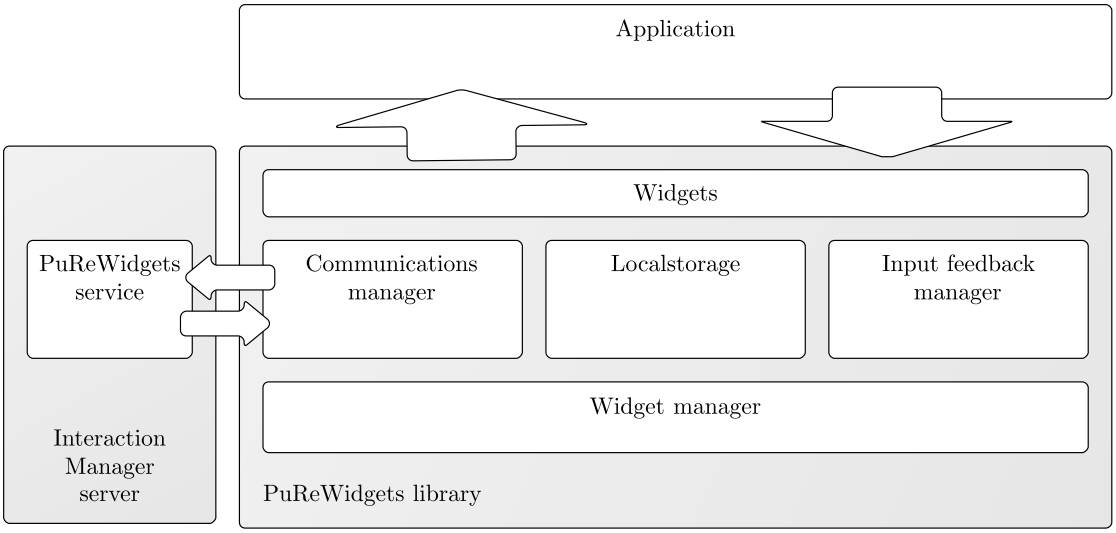


Figure 6.4: PuReWidgets' library components.

When users interact, widgets trigger an interaction event by invoking the callback function that the application registered when the widget was instantiated. The data associated with the event will be different for each widget, but most data is common to all widgets:

user id An user id assigned by the IM, that uniquely identifies the user.

user nickname A human-readable name for the user that can be publicly displayed.

widget object reference A reference to the internal program object that represents the widget.

widget option id The widget option id that was targeted by the user input.

widget specific data Generic object with widget specific event data.

Currently, PuReWidgets provides the following set of widgets.

Button A button widget allows users to trigger actions in the public display application. The interaction event generated by a button is a simple triggered event, with no widget specific data. The default graphical representation of a button (see Figure 6.5) displays a button label and the reference code.

Figure 6.5: Default graphical appearance of a button.

List box The list box widget allows users to select among a set of related items. The high-level event generated by a list box identifies the selected item through the widget option id. Graphically, a list box has the appearance of a box with a title and a set of vertically arranged item names and corresponding reference codes (see Figure 6.6).

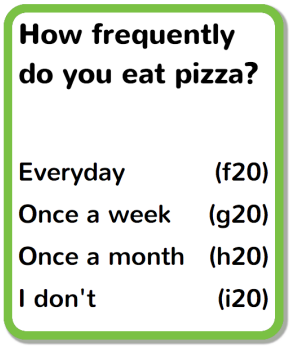


Figure 6.6: Default graphical appearance of a list box widget.

Text box A text box widget allow users to input free text. The interaction event generated by a text box contains a text string corresponding to the user's input, in the widget specific data. Graphically, a text box is depicted as a standard single line input field, with a label and reference code inside. It has also a flashing cursor at the leftmost side (see Figure 6.7).

Figure 6.7: Default graphical appearance of a text box widget.

Upload An upload widget 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[5] so that the application can then download and use the content.

Figure 6.8: Default graphical appearance of an upload widget.

[5]PuReWidgets handles different forms of file uploading. If the upload includes the file contents (for example as an email attachment), PuReWidgets will store the file contents and provide applications with a temporary URL for the file. If the upload consists already of an URL for a file, PuReWidgets simply passes it to the application.

Download The Download widget allow the application to provide files that users can download to their personal devices, 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. Applications are required only to, when instantiating the widget, specify the location (an URL) of the associated media file.

Figure 6.9: Default graphical appearance of a download widget.

Check-in 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. Check-in widgets are different from the other widgets in that their input is not directed specifically at a single application, but instead to a place: when users check-in in a place, every application that instantiated a check-in widget will be notified.

Figure 6.10: Default graphical appearance of a check-in widget.

Widget registration

In order for the IM to be able to route user input to the correct application and to be able to generate the web GUI, widgets must be registered in the IM. The registration process itself is hidden from the application and is done automatically by the PuReWidgets library, when a widget is instantiated or changed by the application. To register a widget, the following information is sent to the IM:

ids The place id, instance id, widget id, and option ids, are all sent to the IM.

widget type The type of widget corresponds to the interaction tasks defined in chapter 5: select, entry, download, upload, or check-in, or user-defined type. This is used by the IM to adapt its behaviour in response to a user input and to provide different graphical representations in the web GUI.

short description A short description (e.g., one to three words) of the widget's meaning in the context of the application.

long description A longer description (several words) of the widget. The short and long descriptions (and the control type) are used by PuReWidgets to render an application's widgets in the web GUI. These descriptions are meant to provide context to the widgets, since, in these cases, the user does not have the context of the other elements of the application.

list of option short descriptions A short description of the various options.

list of option long descriptions A longer description of the various options. As with the widget's short and long descriptions, the options' short and long descriptions are used mainly to provide context to alternative renderings of the application's widgets.

list of suggested reference codes (Optional) Each option will be assigned a unique reference code by the IM. This field allows applications to suggest which reference codes they would like to have. PuReWidgets will try to honour these requests but it may not be able to do so if the suggested reference code is already in use.

icons Widgets can have an icon for each widget option. Icons are specified as a URL where the icon can be found.

6.4 User Interaction with PuReWidgets

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

6.4.1 Text-based input

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. These mechanisms require users to explicitly say which widget they wish to address by entering the reference code of the widget in the text message. The IM server provides a set of I/O modules that can receive raw text input from different sources and interpret the interaction commands that are present. Figure 6.11 provides a more detailed view of the IM server and the I/O modules for text-based interaction.

Generically, text-based interactions require users to follow a simple command structure in their text messages, in order to identify the widget option they wish to address. The full command syntax is

```
<place_reference_code>.<widget_reference_code> <additional_parameters>
```

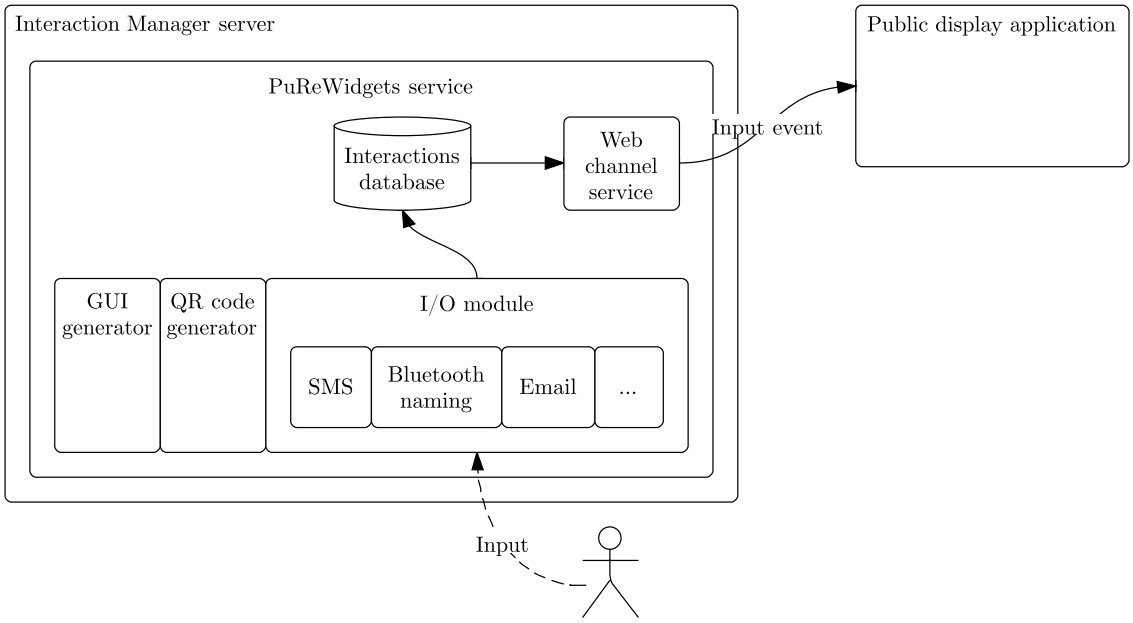


Figure 6.11: I/O modules in PureWidgets.

The additional parameters are widget-specific and not all widgets require them. For example, to activate the 'like' button in a public display located at the UCP university (see Figure 6.5), users would simply send a "ucp.p10" message. To send a poll suggestion to a text box (see Figure 6.7) however, they would enter "ucp.u10 How often do you read books?".

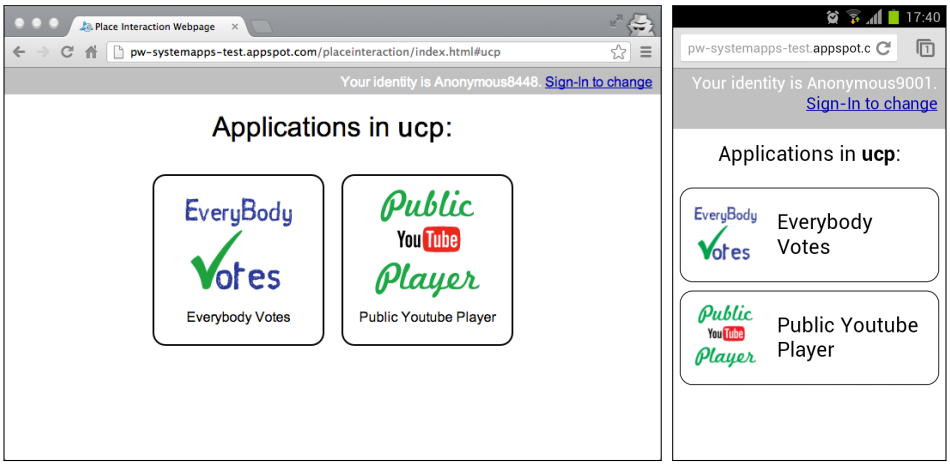
When using a text-based interaction mechanism, users are identified by the address of the respective mechanism. For example, for SMS messages the address is the user's phone number, for email messages the address is the user's email address, and for Bluetooth naming the address is the user device's Media Access Control (MAC) address. Although not currently implemented, these addresses could be associated with a user's profile in the IM allowing the public display to get richer information about its users (see section 8.2 – Future Work – for a discussion about this possibility).

6.4.2 Dynamic graphical interface generation

PuReWidgets provides a dynamically generated web-based GUI for desktop and mobile devices, which allow users to interact with the available applications in a particular place.

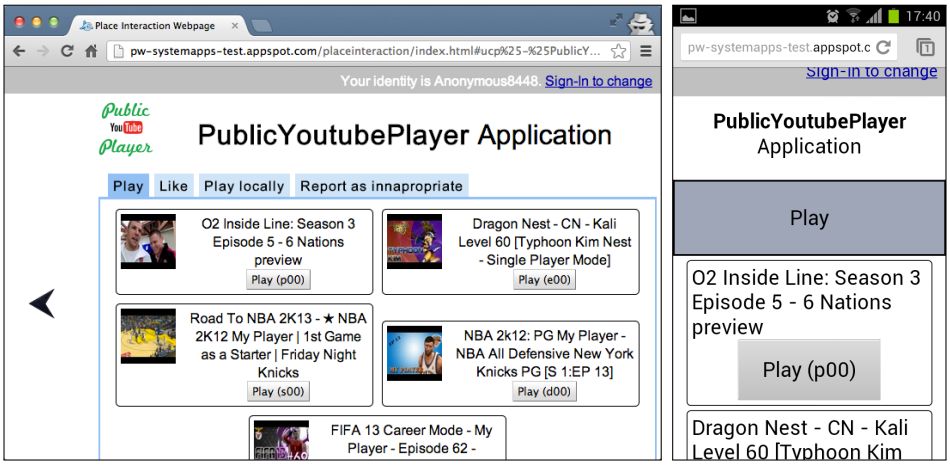
For each place, the IM server provides a web GUI which allows users to see the available applications in that place (see Figures 6.12a and 6.12b), and interact with any widget currently in use by any application (see Figures 6.12c and 6.12d).

When rendering the application widgets, PuReWidgets uses the information that was registered when the application created the widget. For example, the widget



(a) Deskop view of application list.

(b) Mobile view view of application list.



(c) Deskop view of widget list.

(d) Mobile view of widget list.

Figure 6.12: Automatically generated web GUI.

control type is used to determine what type of web controls are needed to provide the interaction (a textbox, a listbox, a button, an upload form, etc.), and the long descriptions are used to provide descriptions for the individual widgets (see for example Figure 6.12c).

The web GUI allows users to interact anonymously, while making sure that the system is able to distinguish different users. For this, the web GUI randomly generates a user id and nickname (in the form of "Anonymous####"), the first time a user accesses the webpage. If users choose to identify themselves, they can do this by logging in with one of several authentication providers (the web GUI currently uses the Janrain service [Janrain, 2013], which provides a unified authentication mechanism across several providers such as Google, Facebook, Twitter, LinkedIn, etc.). In this case, the user's id and nickname are extracted from the provider's account profile.

In order to provide some visual structure to the web GUI, particularly for applications that have several widgets, the short description is used to group widgets in the web layout: widgets with the same short description are grouped together under the same panel.

It should be noted, however, that this automatic interface generation for public display applications does not preclude application developers from creating a custom web or mobile interface to their applications. Both could even be integrated in the display system: the web GUI could load by a default the custom application interface, but fall back to the dynamically generated web GUI one if the former did not exist.

6.4.3 QR code interaction

PuReWidgets also creates QR codes for individual widget options, allowing immediate interaction with specific application features simply by scanning a visual code.

Applications can use the PuReWidgets library to create and show on the public display QR codes for any widget they have created. For example, applications may choose to display QR codes for some or all of their widgets in the public display graphical interface, allowing users to scan the code directly from the public display screen. Applications can request QR code images of various sizes, allowing them to adapt to the display's positioning and average user's distance to the screen.

Display owners also have access to the QR codes for every widget of every application they have configured in a given place. 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. Imagine a polls application in which one of the polls asks users to tell which sports team they think will win the next championship. At the day of an important match for the local team, the display owner may choose to print flyers and leave them at the coffee tables to engage its customers in that event. To generate QR codes, display

owner access a web interface (see Figure 6.13) where they can see all the available applications in a specific place, and select one to list the available QR codes. Display owners can then take the QR code images from the web page and use them to create custom flyers, posters, etc. (see Figure 6.14a). Alternatively, applications can use the QR code service available through the PuReWidgets library, and provide ready to print web pages with custom layouts that display owners can simply print.

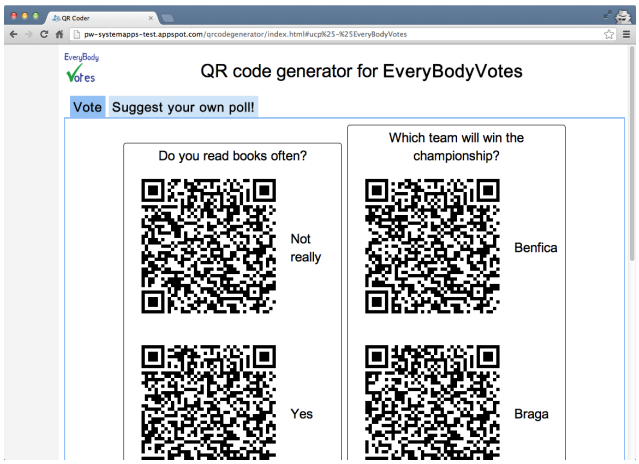
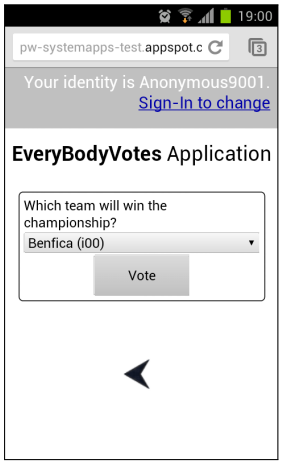


Figure 6.13: Web interface for obtaining QR codes for specific widgets.

(a) Sample flyer with QR codes for a specific poll.



(b) Sample interface for QR code interaction.

Figure 6.14: QR code interaction.

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 an interface for interaction with the specified widget. This interface is similar to the web GUI interfaces described in the previous section. For example, scanning the QR code associated with the "Benfica" answer for the "Which team will win the championship?" poll of Figure 6.14a, would result in the interface depicted in Figure 6.14b with the correct option already selected. Users can also login in the QR code webpage, allowing inputs to be identified instead of anonymous.

6.4.4 Touch-screens

Widgets in PuReWidgets are also touch-enabled. In this case, interaction is always anonymous and the widgets must be on-screen (visible in the display) in order to be interacted with (see Figure 6.15).

Currently, PuReWidgets supports touch interaction with buttons, list boxes, and text boxes. Download and upload widgets can also be easily supported but they would require additional hardware setup on the public display. For example, users could upload and download files from a personal Universal Serial Bus (USB) stick, but this would require access to a USB port connected to the public display's computer. Check-in interactions are harder to support through touch-based interaction as it would require users to authenticate, which not only poses privacy and security concerns, but also raises the interaction barrier too much to make it an attractive option (however, Near Field Communication (NFC) or other personal cards could be used for this).

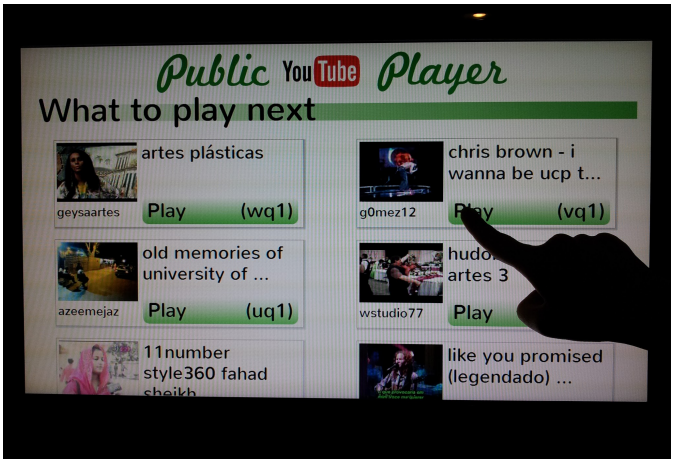


Figure 6.15: Touch interaction with a public display application.

6.4.5 Input feedback

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. Public displays applications may choose one of two design strategies regarding the exposure of user interactions [Dix and Sas, 2010]: some applications may wish to hide interactions, while other may wish to expose interactions. PuReWidgets supports both strategies, by providing a default graphical input feedback but letting programmers disable this behaviour.

For the first design strategy, it means simply that there should be no visible system-level input feedback on the public display: the PuReWidgets library should simply trigger the high-level event and let applications decide what to do regarding the input feedback, including nothing at all. Applications may choose this for several reasons such as to maintain user privacy, to maintain a cleaner graphical interface,

or simply because system feedback is not considered important for that particular application. Whatever the reason, the PuReWidgets library provides a mechanism that allows applications to disable the graphical input feedback that is shown by default.

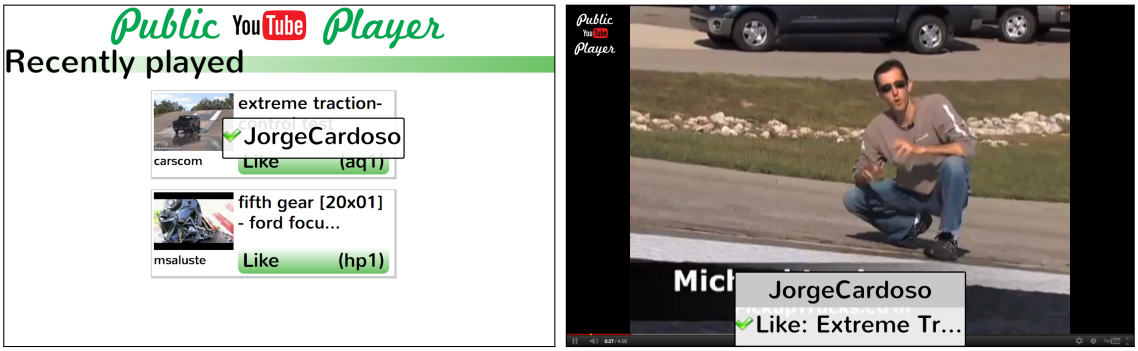
For the second design strategy, the toolkit provides some assistance by generating graphical input feedback on the public display, at the same time that a high-level event is generated. This is the default behaviour of the PuReWidgets library and it can be an important feature for several reasons: it helps to create awareness about the interactive features of an application; it helps creating a more dynamic application by showing user activity; it can help enticing more user interactions [Brignull and Rogers, 2003].

Our approach is to provide a base mechanism for presenting feedback on the public display, which consists of a panel that pops up on top of the corresponding widget. This feedback allows not only the active users to get a confirmation of their actions, but also bystander users to get a better understanding of what is going on the public display. The information displayed in the panel is defined by each widget and is configurable by the programmer but, generally, it identifies the user (a masked version of the user id if the identification is considered sensible information, such as the phone number for SMS interactions) and the action that the user performed.

The toolkit provides slightly different feedback information on the public display, depending on whether the widget that is receiving the input is currently on-screen, or off-screen. For on-screen widgets, the widget itself provides context to the feedback information, so the feedback panel contains, generally, less information. For example, feedback for a button can be simply the nickname of the user that acted on the button (see Figure 6.16a). For off-screen widgets, the feedback panel must provide information that helps users identify the widget itself. By default, PuReWidgets displays a panel at the bottom of the screen with the user's nickname and the short description (label) of the button that was acted on. However, applications can complement this with application-specific information. For example, in Figure 6.16b, the application displays the user's nickname, the button's short description and the title of the video.

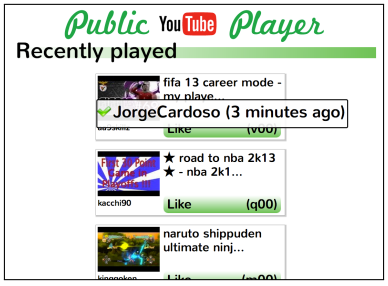
The feedback also helps users understand the asynchronous nature of the interaction by providing time information when the input that is causing the feedback on the display is older than a configurable amount of time (1 minute by default). This can happen when users send input to an off-screen application (or the application is taken off-screen just after the user sent the input). In these situations, the application will only receive the input event, and generate graphical feedback, when it is put on-screen. For input that is older than 1 minute, by default, the feedback includes the age of the input, e.g., "3 minutes ago..." (see Figure 6.16c).

The feedback information is mostly a confirmation of the reception of the user's input, but it can also serve to indicate errors. Particularly with input mechanisms such as SMS, Bluetooth naming, email, etc., where users have to abide by the command syntax, there is a chance that users will make mistakes when entering their input. In these cases, the input feedback panel can serve to indicate that the



(a) On-screen button feedback panel.

(b) Off-screen button feedback panel.



(c) On-screen button feedback panel with age information.

Figure 6.16: Input feedback panel for on- and off-screen buttons.

input was received but it was not correctly understood due to a syntax error, and possibly an indication of what the error was.

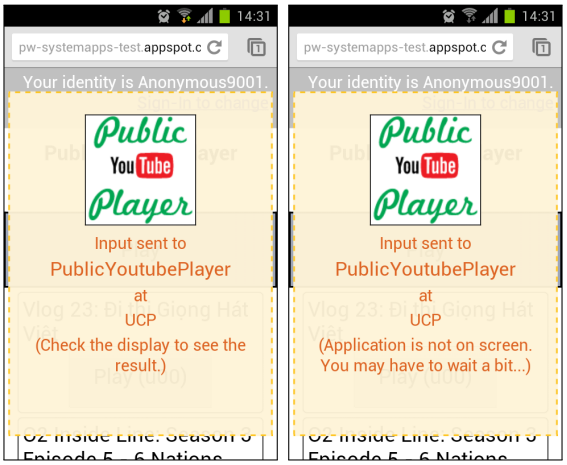
Input feedback on the web GUI

The input feedback that is displayed on the public display itself is applicable to all interactions, regardless of the input mechanism that was used. However, when users interact through the web GUI, or QR code interfaces there is a possibility of also providing some feedback directly on the personal device, confirming that the user input was actually received by the display system (see Figure 6.17).

6.5 Implementation Details

PuReWidgets is currently implemented using Google's Appengine[6] server platform, for the IM, and Google Web Toolkit (GWT),[7] for the library. This means that it currently supports web-based public display applications built with GWT – a development toolkit for web application development where applications are mostly written in the Java language (but in combination with HTML, and CSS) and subsequently translated to Javascript.

[6] `https://developers.google.com/appengine/`
[7] `https://developers.google.com/web-toolkit/`



(a) Application is currently on-screen.

(b) Application is currently off-screen.

Figure 6.17: Input feedback on the web GUI.

The following HelloWorld application will be used to explain the various parts of a public display application that uses the PuReWidgets library and to explain some of its functions in more detail.

```
1  /**
2   * The HelloWorld class is a simple PuReWidgets application that
3   * provides an action button to toggle the background colour of
4   * the screen.
5   */
6  public class HelloWorld implements PDApplicationLifeCycle, EntryPoint {
7
8      // toggles the colour
9      private boolean on;
10
11     /**
12      * This is the GWT entry point method.
13      */
14     public void onModuleLoad() {
15         this.on = false;
16
17         // Give a default id to the application instance and
18         // initialize PuReWidgets' background processes.
19         PDApplication.load(this, "MyHelloWorld");
20     }
21
22
23     /**
24      * This is the PuReWidgets application entry point
25      */
26     public void onPDApplicationLoaded(PDApplication pdApplication) {
27
28         // Create a PD button and set it to 200 pixels width.
29         PdButton button = new PdButton("myPdButtonId", "Click me");
30         button.setWidth("200px");
31
32         // Add the button to the HTML's DOM so that it is visible
33         // on the public display interface.
34         RootPanel.get("buttonDiv").add(button);
```

```
35
36          // Register the callback to react to button presses
37          button.addActionListener(new ActionListener() {
38
39              @Override
40              public void onAction(ActionEvent<?> e) {
41
42                  // toggle the state
43                  HelloWorld.this.on = !HelloWorld.this.on;
44
45                  Style style = RootPanel.getBodyElement().getStyle();
46
47                  //turn the background on or off (white or black)
48                  if ( HelloWorld.this.on ) {
49
50                      //turn off
51                      style.setBackgroundColor("#000000");
52
53                  } else {
54
55                      //turn on
56                      style.setBackgroundColor("#ffffff");
57
58                  }
59
60                  // add the user's nickname to the bottom of the screen
61                  Label newUser = new Label(e.getNickname());
62                  RootPanel.get("messagesDiv").add(newUser);
63
64              }
65          });
66      }
67  }
```

6.5.1 Application loading

Since a PuReWidgets application is also a GWT application, it's entry point is GWT's `onModuleLoad()` method. In GWT's entry point, we simply load our application, giving it a default instance id. Loading an application will make the PuReWidgets library instantiate its internal datastructures and make an initial call to the IM server for getting the application's data and initialising the persistent communication channel for input notifications. The loading process triggers an `onPDApplicationLoaded()` callback,[8] which is where programmers can start making use of the PuReWidgets library (see Figure 6.18).

6.5.2 Widget instantiation

In this HelloWorld example, we create a single action button widget:

```
29  PdButton button = new PdButton("myPdButtonId", "Click me");
```

[8] Just like in the GWT toolkit, many functions in PuReWidgets are asynchronous.

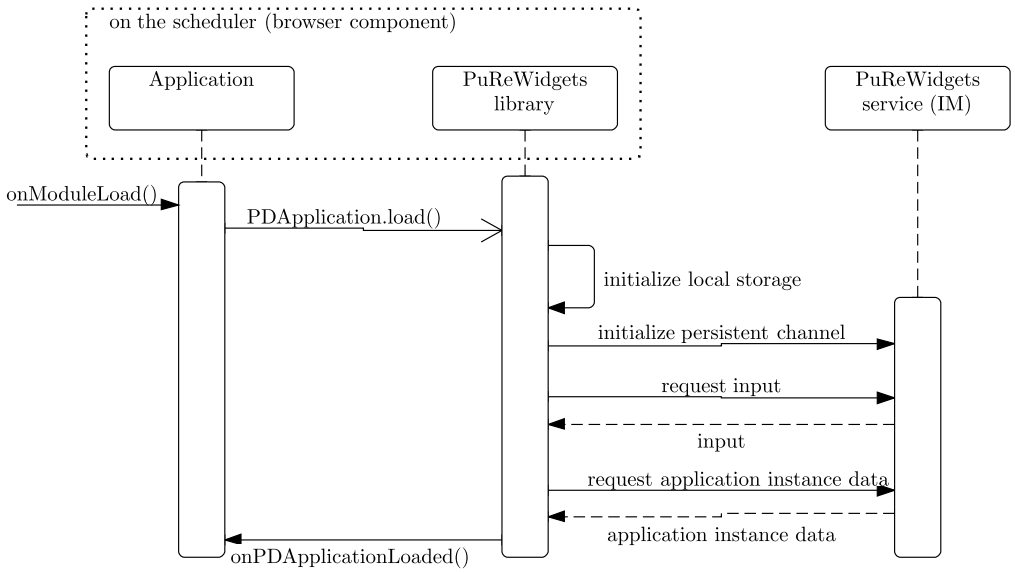


Figure 6.18: Sequence diagram for application loading.

When instantiating a widget, programmers need to specify the widget's id ("myPdButtonId") – a unique name for the widget within the application. This id allows the IM to correlate the newly created widget with previously existing widgets and determine if a given widget is already stored and if any of its properties have changed. In the PdButton widget case, it is also necessary to specify the button's label – the text string that will appear on top of the graphical representation in the public display graphical interface.

In order for the graphical representation of the button to appear on the public display, it is necessary to add the widget to the Document Object Model (DOM) of the webpage. This is done just like with a standard GWT widget (a PuReWidgets widget is also a GWT widget):

```
34 RootPanel.get("buttonDiv").add(button);
```

When a widget is instantiated, the PuReWidgets library needs to send the widget's description to the IM server. (In fact, the widget's description is sent to the IM every time the application makes changes to the widget.) In order to reduce communication with the IM, the PuReWidgets library uses a delayed synchronisation technique that consists of waiting for several widget instantiations (or changes) before propagating them to the IM server (see Figure 6.19). When the synchronisation timer expires, all new or modified widgets will be registered on the IM. Registration consists of sending the widget's information to the IM (see section 6.3 for details on the information that is sent). For widgets that are being registered for the first time, the IM will assign a unique reference code, which the PuReWidgets library will receive and possibly update on the graphical view of the widget.

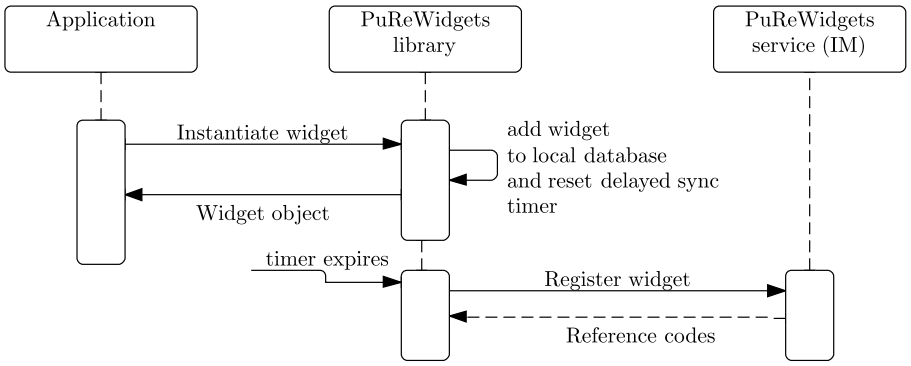


Figure 6.19: Sequence diagram for widget instantiation.

6.5.3 Input events

In PuReWidgets, all widgets trigger an `ActionEvent` that describes the high-level event generated by the widget in response to user input (see section 6.3 for a description of the main input event data). Applications must register a callback function that will handle these high-level events. In the case of the HelloWorld application, we simply toggle a variable to set the background colour, and append the nickname of the user that has interacted to the end of the screen:

```
37 button.addActionListener(new ActionListener() {
38
39     @Override
40     public void onAction(ActionEvent<?> e) {
```

Figure 6.20 shows the sequence diagram of the input processing through the main PuReWidgets components. When users interact using any of the supported interaction mechanisms, their input is received by the IM server, which will analyse the input to determine the target application. If the target application is on-screen, the input is forwarded immediately via the persistent connection that the communications manager component of the PuReWidgets library keeps with the IM. The input is then passed to the widget manager component, which forwards it to the correct widget. The target widget then processes the input and triggers a high-level event in the application. If the target application is off-screen, the input is simply stored, until the application is loaded again by the public display scheduler and asks for the input received while it was off-screen.

6.5.4 Extending widgets

In PuReWidgets, widgets can be extended in several ways. The most common is to extend a widget's graphical appearance on the public display in order to match the application's visual style, but it's also possible to create new widgets which extend another widget's behaviour, or that combine several existing widgets into a composite one.

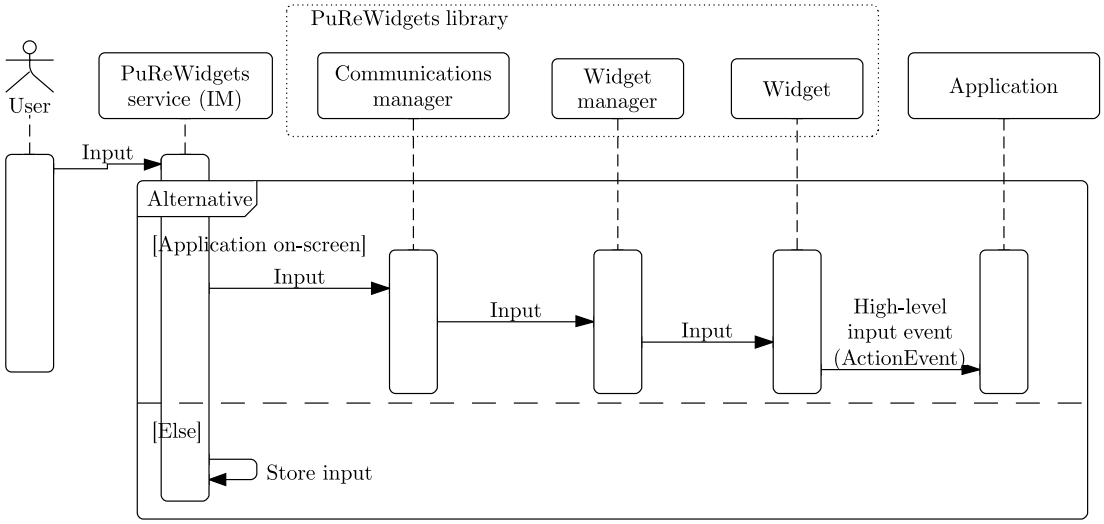


Figure 6.20: Sequence diagram of input event.

Extending behaviour

Although it is possible to create a completely new widget from scratch, the easiest way it to reuse the code from an existing widget, extending it's class definition and overriding some of its methods.

The following code example creates a new button widget that triggers an event only when a pre-determined number of different users have interacted with it. This example extends the existing `PdButton` implementation overriding the method that handles user input:

```
1  public class PdBucketButton extends PdButton  {
2
3      /**
4       * The number of different users that need to interact
5       * with this BucketButton to trigger an event.
6       */
7      private int threshold;
8
9
10     /**
11      * The list of ids of the users that have interacted so far.
12      */
13     private ArrayList<String> userIds;
14
15     /**
16      * Creates a new BucketButton with the given id, caption,
17      * and threshold.
18      */
19     public PdBucketButton(String widgetId,
20                           String caption, int threshold) {
21         super(widgetId, caption);
22
23         this.threshold = threshold;
24
25         // load the list of users that interacted from localstorage
```

```
26          userIds = this.getLocalStorage().loadList("users");
27      }
28
29      /**
30       * Handles input from the user, creating the ActionEvent that will
31       * be sent to the application, and the InputFeedback that will be
32       * displayed on the public display.
33       *
34       * The PdBucketButton triggers an application event only when
35       * "threshold" different users have activated the button.
36       */
37      @Override
38      public InputFeedback<PdButton> handleInput(WidgetInputEvent ie) {
39
40          // reuse the superclass's implementation
41          InputFeedback<PdButton> feedback = super.handleInput(ie);
42
43          // Add new users to a local list of users
44          if ( !this.userIds.contains(ie.getUserId()) ) {
45              this.userIds.add(ie.getUserId());
46
47              // save to localstorage, in case our app gets unloaded
48              this.getLocalStorage().saveList("users", this.userIds);
49          }
50
51          // By default the ActionEvent is null, i.e., no event will
52          // be triggered
53          ActionEvent<PdButton> ae = null;
54
55          // If we crossed the threshold, trigger an ActionEvent
56          if ( this.userIds.size() >= this.threshold ) {
57
58              ae = new ActionEvent<PdButton>(ie, this, null);
59
60              // clear for next round...
61              this.userIds.clear();
62
63              // save to localstorage...
64              this.getLocalStorage().saveList("users", this.userIds);
65          }
66
67          // Set the correct ActionEvent to trigger on the application
68          feedback.setActionEvent(ae);
69
70          return feedback;
71      }
72  }
```

The main logic of this `PdBucketButton` is implemented in the `handleInput()` method. This method is called by the widget manager component of the library (see Figure 6.4, on page 130) whenever a new input is detected for the widget. `PdBucketButton` keeps a list of users ids that have already interacted with the button, adding a new entry to the list whenever a new user interacts. If the size of the list reaches the pre-determined number (threshold), the widget creates a new `ActionEvent` object and returns it inside an `InputFeedback` object. If the threshold hasn't been reached, a null `ActionEvent` is passed. The widget manager will then

use the `InputFeedback` objecto to display graphical input feedback on the public display, and trigger the event in the application (if not null).

This example also illustrates another feature of the PuReWidgets library that allows widgets to store data locally in a persistent manner, by using the browser's local storage facilities. PuReWidgets provides all widgets with an individual local storage area, allowing them to store and retrieve data across applications' loading and unloading:

```
26 userIds = this.getLocalStorage().loadList("users");
```

In this particular case, the `PdBucketButton` keeps the current list of user ids that have already interacted in the local storage, guaranteeing that it will be in the correct state if the application is unloaded and then loaded again.

Extending graphical representation

It's also possible to change an existing widget's graphical representation on the public display by using standard CSS style rules. For example, to change the previous PdBucketButton style into something like Figure 6.21, we could add the following lines to it's constructor (the style could also be changed by the program using the widget, instead of by the widget itself):

```
Resources.INSTANCE.css().ensureInjected();
this.setStyleName(Resources.INSTANCE.css().bucketbutton());
```

and define the style using the following files (this is a standard GWT approach to styling the application):

```
/* File: bucketbutton.css */

.bucketbutton {
    padding: 0;
    font-family: Verdana, Geneva, sans-serif;
    font-size: 2em;
    width: 100%;
    height: 75px;
    display: inline-table;
    border: 0px solid white;
    border-bottom: 5px solid black;
    border-left: 5px solid black;
    border-right: 5px solid black;
}

/*File Resources.java */
public interface Resources extends ClientBundle {
    public static final Resources INSTANCE =
                                    GWT.create(Resources.class);

    @Source("css/bucketbutton.css")
    public BucketButtonCss css();
}
```

```
/* File: BucketButtonCss.java */
public interface BucketButtonCss extends CssResource {
    String bucketbutton();
}
```

The same can be done to every other widget in PuReWidgets.

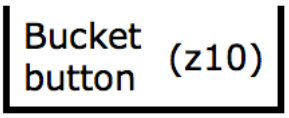


Figure 6.21: Possible bucket button graphical representation.

Extending by composition

Another possibility for creating new widgets is to use the existing ones as building blocks for composing more complex widgets. Composite widgets can listen to events from several widgets and integrate them into higher-level events. In the next example, a text box widget and an upload widget are combined in a tag receiver widget that allows users to send tags explicitly via the text box, or implicitly by sending an MP3 file (the widget extracts the tags contained in the file). The PdTagReceiver widget provides applications with an interaction event only when new tags are received from a user. The strategy here is for the composite widget to register and listen to its child widget's events, process them according to its own logic, and only trigger a high-level if specific conditions are met (in this case, the condition is that new tags have been received).

```
1  public class PdTagReceiver extends PdWidget implements ActionListener {
2
3      /**
4       * This widget's UI is described in an XML file using GWT's UiBinder
5       */
6      private static PdTagReceiverUiBinder uiBinder =
7              GWT.create(PdTagReceiverUiBinder.class);
8      interface PdTagReceiverUiBinder extends
9              UiBinder<Widget, PdTagReceiver> {}
10
11     /**
12      * A PdTagReceiver is composed of a text box and an upload widgets.
13      */
14     @UiField
15     PdTextBox pdTextBox;
16
17     @UiField
18     PdUpload pdUpload;
19
20
21     /**
22      * The current list of tags.
23      */
24     private ArrayList<String> tags;
25
26
```

```
27      public PdTagReceiver(String widgetId) {
28          // Bookkeeping operations for all PdWidgets
29          super(widgetId);
30
31          // Initialize the ui based on the xml description file
32          initWidget(uiBinder.createAndBindUi(this));
33
34          // This composite widget will listen
35          // to its child widgets' events
36          pdTextBox.addActionListener(this);
37          pdUpload.addActionListener(this);
38
39          // Load the set of tags from local storage.
40          tags = this.getLocalStorage().loadList("tags");
41      }
42
43
44      /**
45       * Listen to events from the widgets that compose this one and
46       * generate a higher-level event only when new tags arrive.
47       */
48      @Override
49      public void onAction(ActionEvent<?> e) {
50          Object source = e.getSource();
51
52          ArrayList<String> newTags = new ArrayList<String>();
53
54          // tag arrived from the textbox
55          if ( source == pdTextBox ) {
56
57              // extract the tag from the textbox ActionEvent
58              String tag = (String)(e.getParam());
59
60              // if its a new tag, add it to the list
61              if ( !this.tags.contains(tag) ) {
62                  newTags.add(tag);
63                  this.tags.add(tag);
64                  this.getLocalStorage().saveList("tags", this.tags);
65              }
66          // tag arrived from the file upload
67          } else if ( source == pdUpload ) {
68
69              // extract the tags from the mp3 file
70              ArrayList<String> tags =
71                  AudioService.getMp3Tags((String)(e.getParam()));
72
73              // add new tags to the list
74              for ( String tag : tags ) {
75                  if ( !this.tags.contains(tag) ) {
76                      newTags.add(tag);
77                      this.tags.add(tag);
78                      this.getLocalStorage().saveList("tags", this.tags);
79                  }
80              }
81
82          }
83
84          // trigger an event if we received new tags
85          if ( newTags.size() > 0 ) {
```

```
86
87              // the action event will contain information about the user
88              // that contributed the new tags, and the list of new tags
89              ActionEvent<PdTagReceiver> ae =
90                      new ActionEvent<PdTagReceiver>(e.getUserId(),
91                          e.getNickname(), this, null, newTags);
92
93              this.fireActionEvent(ae);
94          }
95      }
96
97
98
99      /**
100      * Used by UiBinder
101      */
102     @UiFactory
103     public PdTextBox createTextBox() {
104         PdTextBox pdTextBox = new PdTextBox(this.widgetId+"-textbox",
105                                             "Send some tags", null);
106         return pdTextBox;
107     }
108
109     /**
110      * Used by UiBinder
111      */
112     @UiFactory
113     public PdUpload createUpload() {
114         PdUpload pdUpload = new PdUpload(this.widgetId+"-upload",
115                                          "Upload an mp3");
116         return pdUpload;
117     }
118
119 }
```

6.5.5 Server-side PuReWidgets library

PuReWidgets also provides a server-side library for Appengine applications that provides functions for creating, updating, and deleting of widgets and for input notification.

By using the server-side library, applications can be immediately notified of any input, regardless of the current screen state of the application in the public display. For applications that use this server-side library, PuReWidgets immediately notifies them when an input arrives (see Figure 6.22). For this, PuReWidgets needs to know the notification URL of the application in order to send it the input event description (currently, the URL is manually configured in the IM). If there is no notification URL, PuReWidgets follows the standard process depicted in Figure 6.20, on page 145.

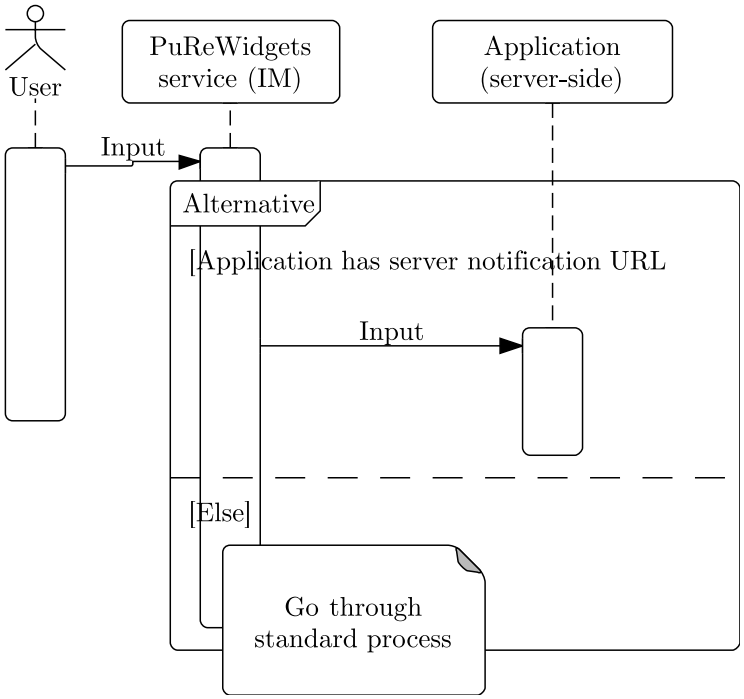


Figure 6.22: Sequence diagram of input event for server-side notification.

6.6 Conclusion

We have presented an interaction abstraction toolkit for web-based public display applications that is in line with the requirements set out in the beginning of this work.

The PuReWidgets toolkit provides multiple, extensible, widgets that support the various interaction tasks for public displays we identified in chapter 5; provides a dynamically generated graphical web GUI for rich desktop and mobile interaction with public displays; is independent of specific input mechanisms and modalities, supporting a wide range of input mechanisms without burdening developers with their details; provides asynchronous interaction for always available interaction; supports concurrent, multi-user interaction; and provides customisable graphical representations for widgets, and system feedback on the public display.

PuReWidgets integrates into a regular application development process by using the same programming constructs as regular web applications, providing flexible (client- and server-side) libraries that allow developers to have fine control over the details of the application's interface.

Chapter 7

Evaluating PuReWidgets

Contents

7.1 Introduction

Evaluating a programming toolkit is a difficult task because there are many possible dimensions that can be evaluated. The system's performance, scalability, API, various code metrics for the resulting applications, usability of resulting applications, are examples of possible aspects that can be evaluated. The API of the toolkit's programming library alone can be evaluated against various desirable characteristics, such as being easy to learn and memorize, leading to readable code, being hard to misuse, being easy to extend, and being complete [Blanchette, 2008]. Evaluating a toolkit for interactive public display applications is particularly difficult also because there are no direct alternatives that we can use as a baseline for comparing.

We considered that an evaluation of the toolkit would have to address various dimensions. Doing an in depth evaluation focused on a single dimension would mean taking the risk of missing a critical aspect of the system on one of the left out dimensions. Given the novelty of the application area, our rationale is that it is more important at this time to evaluate and validate a wider range of dimensions, making sure we have a generally viable system.

This choice was also inspired in the approach taken by others when evaluating programming toolkits and libraries [Amador and Gomes, 2011; Hardy and Alexander, 2012; Heer et al., 2005; Klemmer et al., 2004]. We evaluated PuReWidgets from four different perspectives. First we evaluate the system's performance, focusing on the Interaction Manager (IM) server. We then evaluate the API's flexibility and capabilities using our own experience in developing interactive applications with it. We also evaluate the API's usability from the perspective of independent programmers. Finally, we provide an evaluation of the resulting system's usability from the perspective of an end-user interacting with a real-world deployment of a public display.

7.2 System Performance

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 its limitations and understand how well it performs when handling various simultaneous places and applications.

Given that the IM is implemented over Google's Appengine cloud service, our tests were also performed over this platform. However, the results would be applicable to other similar platforms, as they reflect the general architecture of the toolkit and not its specific implementation in Appengine.

Google's Appengine provides a scalable infrastructure, with distributed data storage and dynamic application instances. Google charges Appengine applications based on the amount of resources used – CPU, API calls, bandwidth, and storage are

the general resource types charged by Google.[1] Some of these resources are tied to specific services, which may not be used by every application. Appengine provides a daily free quota for each resource; below the free quota usage, applications are not billed.[2]

For the IM, the resources that are most relevant are the frontend instance hours, datastore operations, and channels. Frontend instance hours refers to the sum of the running time of the various application instances. Instances are computing units, similar to virtual machines, used by Appengine to scale an application. By default, instances are automatically allocated by Appengine to serve incoming requests. If an instance's incoming requests queue grows too large, Appengine allocates a new instance to handle the load. When the load decreases, so do the number of instances. Because creating instances is a costly operation, they are not killed immediately when there are no more requests to serve, instead Appengine waits 15 minutes before killing the instance. Datastore write and read operations are the number of low-level operations over the application's datastore – an object database that replicates data across multiple data centres. The datastore holds data objects known as entities and a single high-level operation over an entity may require several read and write operations, depending on the entity's structure. Channels are persistent connections between a Javascript client and an Appengine application. Channels have a channel id that can have a lifespan of at most 24 hours.

In this evaluation we wanted to answer the following questions:

1. How does the IM scale with an increasing load (with the number of places, applications, and interactions)?
2. What are the execution times of the various types of requests that the IM handles?

To answer these questions, we measured the resource usage and the execution time of the various requests to the IM for an increasing number of applications configured to run in several places.

7.2.1 Procedure

The PuReWidgets implementation (both the server and client sides) uses various caching mechanisms to reduce the communication bandwidth and CPU processing needed. On the Appengine side, for example, caching takes advantage of the Memcache service provided by Google – a distributed memory cache service. PuReWidgets uses Memcache to temporarily store datastore results in order to reduce the number of datastore accesses. Memcache may evict stored values due to lack of

[1] The complete list of resources, at the time of writing, can be found at `https://developers.google.com/appengine/docs/quotas`.

[2] Billing must be explicitly activated, if not activated, applications that go over the free quota are simply denied further resources until the daily quotas are reset.

available memory (the exact cache size is not known to developers), or due to a server failure (Memcache is a distributed service, but it is not guaranteed to survive a server failure). Because of these mechanisms, it is not possible to have a deterministic model of how many resources the IM will need to serve an application.

In order to have an estimate of the resource needs of the IM, we ran a simulation of an increasing load, and measured the resource usage and server execution times. For this, we developed a test application that 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, during the 10-hour period, are loaded/unloaded 20 times each (200 in total), receiving a total of 200 inputs, creating a total of 145 different widgets, and deleting 100 of those widgets. Creating, and deleting widgets, and receiving input 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.

After each running session, we manually accessed and recorded the information from Appengine's administration console, which shows the currently percentage of free daily quota resources consumed. We also manually accessed and recorded the profiling information of Appstats console,[3] – the profiling tool of Appengine – which records the last (approximately) 1000 requests,[4] reporting the execution time and cost of each HyperText Transfer Protocol (HTTP) request made to the server application.

7.2.2 Results and discussion

The results for the quota usage for the various resources and the request execution time, during the 10-hour simulation period, are shown in Figures 7.1, 7.2, and 7.3.

[3] `https://developers.google.com/appengine/docs/java/tools/appstats`

[4] The actual number of requests that are recorded by Appstats depends on the available memory resources.

Quota usage results are indicated as a percentage of the daily free resource quota provided by Appengine.

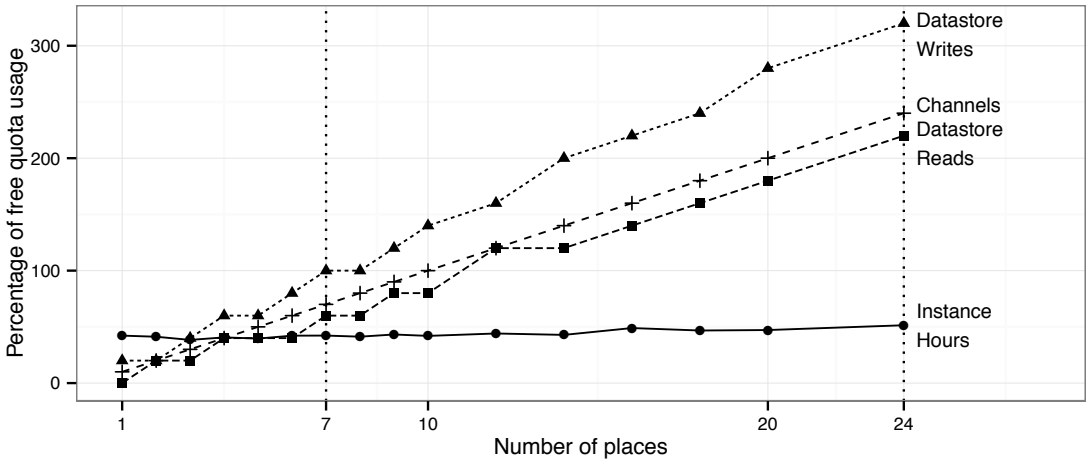


Figure 7.1: Quota usage for an increasing number of places.

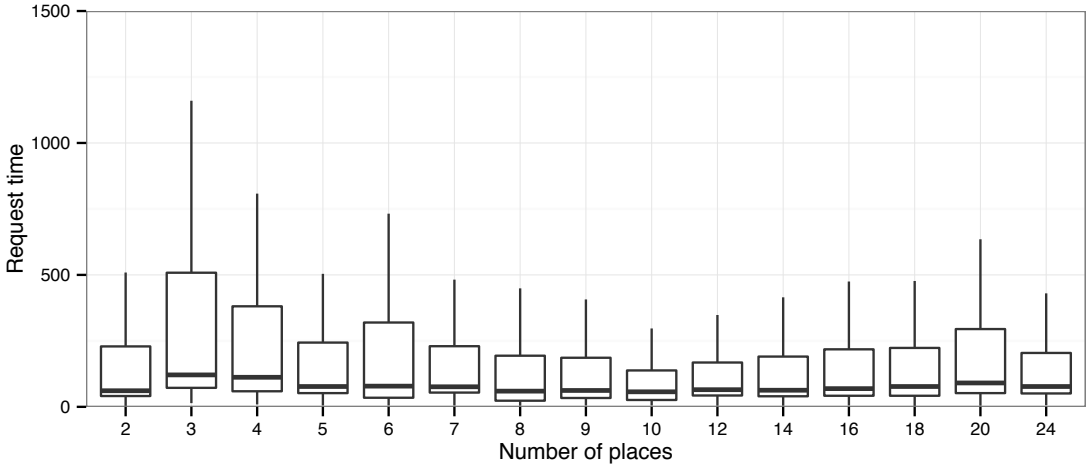


Figure 7.2: Execution time for an increasing number of places.

The plot of Figure 7.1 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 7.2 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 was not greatly affected by the number of places. These are expected results given that, in the current implementation of the IM, there is a linear increase of the number datastore entities that need to be stored as the number of applications increases. Our tests were unable to push the limits of the instance hours, which just barely crossed the 50% threshold. Despite the linear increase in most resources, the execution time of the various requests

handled by the IM remained fairly constant, as shown in Figure 7.2. This indicates that the IM is able to scale well with an increasing number of applications, and interactions, being able to maintain its response time constant. This is an expected result since Google's Appengine is itself a highly scalable infrastructure.

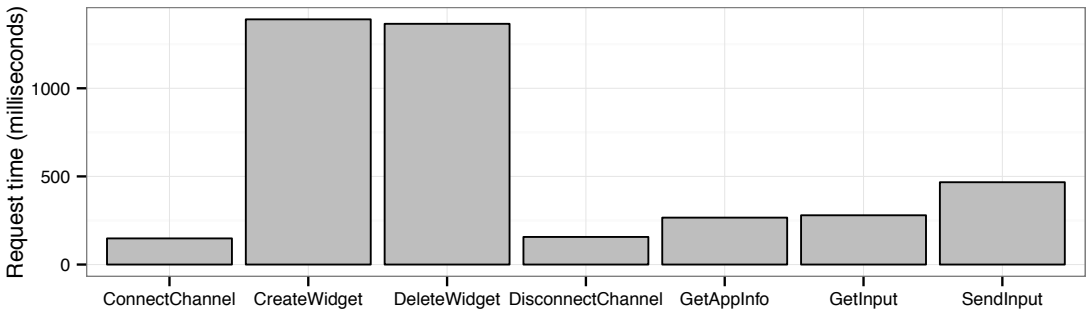


Figure 7.3: Average execution time for the various request types.

Figure 7.3 shows the average execution times and cost of the various requests types handled by the IM. The most lengthy and expensive requests are creating, deleting, and sending input to widgets. Each application creates and deletes widgets in batches, so the values in the chart show the average time and cost for creating and deleting an average number of widgets. These are the operations that need more datastore operations and CPU time. Creating a widget requires the IM to check if it exists already, assign reference codes, and save all widget's parameters and widget options to the datastore. Deleting a widget is also an expensive operation (in terms of time and cost) because it means not only deleting the widget and its widget options from the datastore, but also all related input. However, although these operations are comparatively lengthier to execute than the other operations on the IM, from an application's perspective this is not a problem. All communication between the PuReWidgets library and the IM is made with asynchronous calls meaning that applications don't have to wait for the completion of a call to continue their execution. When a widget is created, applications can immediately display it graphically even before the IM has received the widget's information (the only information that cannot be rendered at that time are the reference codes). These execution times can serve as a reference for programmers to understand the cost of each operation in the IM.

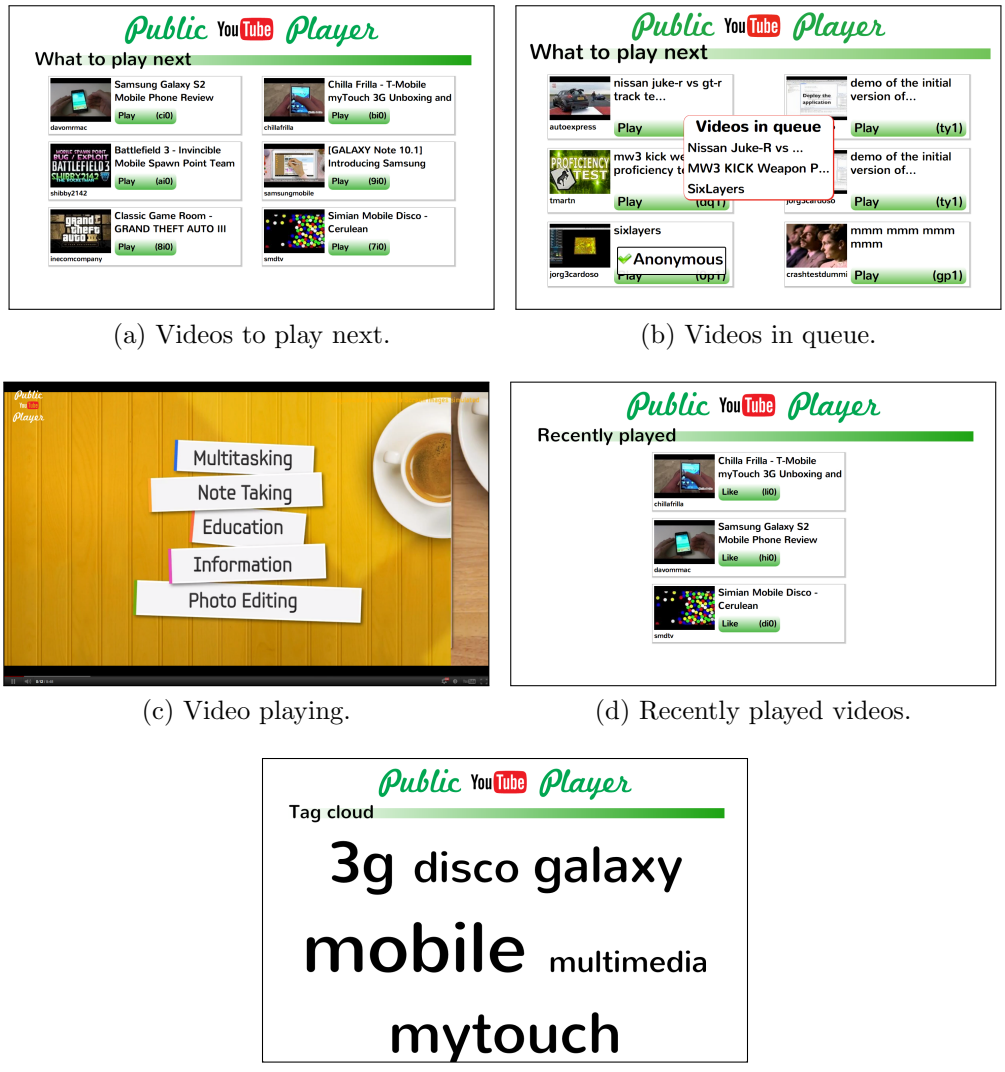
Although these were limited tests, unable push the IM to its limits, we can conclude that there are no evident problems with the implementation of the server and that it is capable of scaling well.

7.3 API Flexibility and Capabilities

The development of the PuReWidgets toolkit followed an iterative approach that alternated between implementation of the toolkit and implementation of three applications that used the toolkit. This allowed us to better fit the library's Application

Programming Interface (API) to real world needs. We developed three interactive public display applications during this phase: the Public YouTube Player, Everybody Votes, and Wrod Game. While developing these applications, we iterated through several versions of the toolkit in order to provide better support for the features we wanted to include in the application. These applications also allow us to evaluate the flexibility and the limits of the functionality of the current version of the toolkit.

7.3.1 Public YouTube Player



(a) Videos to play next.

(b) Videos in queue.

(c) Video playing.

(d) Recently played videos.

(e) Tag cloud.

Figure 7.4: Screens in the Public YouTube Player application.

The public video player is an application that searches for, and plays YouTube videos. This application allows display owners to specify a list of keywords to be used to search for videos. It also allows display owners to specify a list of featured YouTube users from which the application will display videos, in addition to the ones that result from searches. The general logic of the application is as follows:

1. The application selects one keyword from the current set of keywords (initially this set of keywords is equal to the set specified by the display owner) and searches for videos containing that keyword. The keyword set has weights associated with each keyword, so that some keywords are more likely to be chosen.

2. The application displays six videos on the public display (see Figure 7.4a), combining videos from the search result, and videos selected from the featured users (the percentage of videos from these two sources can be configured by the display owner). For each video, the application shows a thumbnail of the video, its title, and the YouTube username that uploaded it. It also shows a button that allows users to select the associated video for playback. If users select a video for playback, the video is put on a queue (see Figure 7.4b). Videos that are selected for playback, are removed from the set of possible videos to play, causing the application to select another video to replace the removed one.

3. After a pre-configured (by the display owner) amount of time, the application retrieves the first video from the queue and plays it (see Figure 7.4c). If the queue is empty, it chooses one at random from the list of six videos.

4. When the video finishes playing, the application displays a list of recently played videos (see Figure 7.4d). This list shows a button next to each video that allows users to "like" the corresponding video. Liking a video increases the weight of its keywords in the keyword set of the application.

5. After another pre-configured amount of time, the application shows the current keyword set using a tag cloud style (see Figure 7.4e).

6. After another pre-configured amount of time, the screen with the search results is displayed again.

Although the application's interface on the public display only shows two different kinds of buttons, users have other interaction possibilities from the web GUI that is generated by PuReWidgets. The complete set of interactive features in this application is the following:

- Users can "like" one of the last six videos that have already been played; although the public display interface displays only the last three played videos, the web interface allows users to select up to the sixth video. This feature uses a button widget.

- Users can get the URL of a recently played video (the last six videos are available); In this application, when users select this feature they are redirected to the corresponding YouTube video web page. This feature uses a download widget.

- Users can select a video to be played from the list of search results; in this case, the public display interface and the mobile interface show the exact same number of videos that can be selected. This feature uses a button widget.

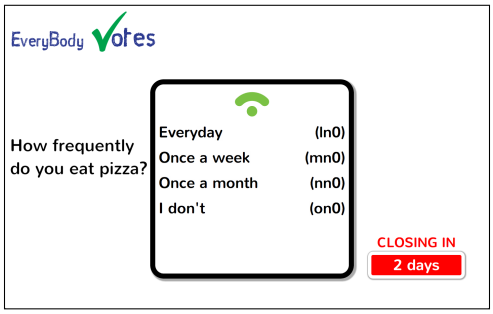
- Users can report an inappropriate video. If users found one of the last six played videos offensive, they can report it. This feature uses a button widget.

In the Public YouTube Player application, the interactive widgets are associated with specific YouTube videos. For example, when a video is removed from the list of videos that can be played next, the "play" button associated with that video is removed from the interface and deleted. This means that the application's interface is highly dynamic, with widgets being continually added and removed from the graphical public display interface, and from the IM.

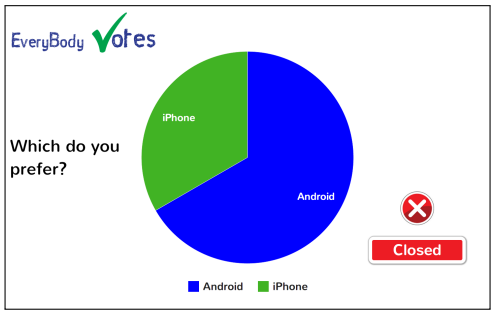
The display owner can customise this application using a web interface that allows various parameters to be edited and saved. These values are associated with a application instance running in a specific place.

7.3.2 Everybody Votes

The Everybody Votes application allows users to vote on polls that display owners create.



(a) Open poll.



(b) Closed poll.

(c) Suggestion box.

Figure 7.5: Screens in the Everybody Votes application.

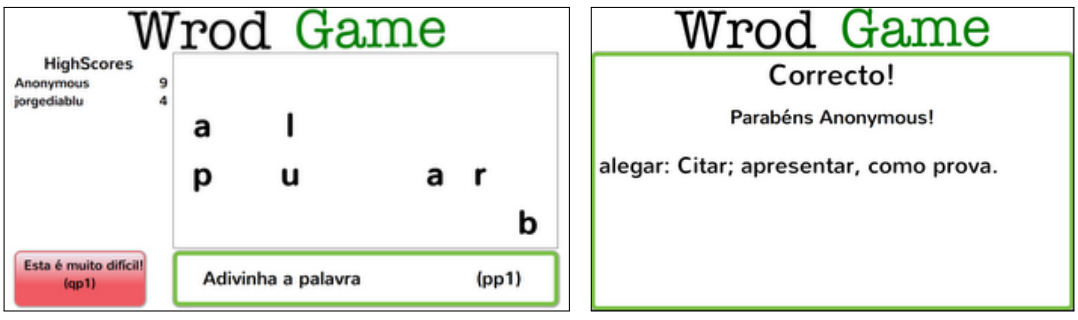
The Everybody Votes application is a polls application composed of three screens, depicted in Figure 7.5: an open polls screen (7.5a) which iterates through the open polls, showing the poll question, possible answers, and time left before the poll closes; a closed polls screen (7.5b) which iterates through the closed polls and shows their voting results; a suggest box screen (7.5c) enticing users to suggest their own polls (which will go through a moderation process by the display owner).

Display owners can use an application administrator interface to create the various polls for the application. When creating a poll, they can specify a title, the various options, the open date, the close date, and the end date. The open date specifies the day on which the application will open the poll, starting to display it and accepting votes; the close date specifies the day on which the application will not accept any more votes, and will start showing the results of the voting for that poll; the end date specifies the day on which the application will no longer display the poll. Users can interact with this application at any time to vote on the various open polls. Each poll has a listbox widget associated, which lists the various alternative votes. The listbox widget is deleted when the poll is closed.

Users can also suggest a poll to the display owner by sending text to a textbox widget. This widget is permanent on the application. Display owners receive the poll suggestions in an email address configured in the application's administration interface. This application uses the server-side library of the PuReWidgets toolkit to receive and process poll suggestions. When a user sends a suggestion, the application's server side code is notified and processes the input immediately, sending the display owner an email with the suggestion.

7.3.3 Wrod Game

The Wrod Game application displays anagrams of Portuguese words (see Figure 7.6a) and allows users to guess what the word is. For each correct word, players get as many points as the number of letters in the word. When players guess the correct word, the application displays the definition of the word (see Figure 7.6b).



(a) Main screen. (b) Results screen.

Figure 7.6: Screens in the Wrod Game application.

This application uses only two widgets: one textbox that is used for receiving the guessed words, and a button widget that allows users to skip the current word and ask for a new one. These widgets are permanent in the application, i.e., the application only creates these two widgets, and they are never deleted. However, the textbox widget is updated whenever the current word on the screen changes: the long description of the widget is set to "What word is this:" followed by the anagram of the current word.

7.3.4 Discussion

Developing these applications highlighted the benefits of our API design goals, particularly the goal of trying to achieve a low learning threshold for programmers. The programming model of PuReWidgets is well integrated with Google Web Toolkit (GWT), allowing programmers to directly apply the standard GWT structures and techniques for creating, styling, and managing the graphical user interface of the application. This integration evolved significantly with the development of the three test applications. In this section, we discuss some of the aspects that were improved in the toolkit as a consequence of the development of these applications, and also some shortcomings we have detected.

Delayed synchronisation technique

In an initial version of the PuReWidgets library, programmers were required to explicitly send widgets to the IM, after they created or changed a widget. While programming the first versions of the Public YouTube Player, this soon proved to be a source of errors as it was easy to forget to send the widget to the IM. A better solution would simply be to automatically send a widget to the IM whenever any of its properties were changed (the base class for all widgets – `PDWidget` – could easily be altered to detect these changes). However, in many situations, configuring a widget means changing various properties, which would mean sending several sequential updates to the IM, using unnecessary network and server resources. To get the best of both worlds, we changed the Widget Manager component of the PuReWidgets library, introducing 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" in the Widget Manager and resets a delay timer. When the timer expires, all dirty widgets are sent to the IM. This allowed us to change the API of the library, removing the need to explicitly send a widget to the IM and, at the same time, minimise the network and server overhead.

Application parameters

While developing the Public YouTube Player application, it became obvious that there were application parameters that display owners should be able to configure. 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 these parameters are related to a specific instance of an application. PuReWidgets provides functions to read and write parameters in the form of name-value pairs that are stored on the application's server. Applications can provide an administration web interface that display owners can use to configure the various parameters for a specific application instance (PuReWidgets provides a template for this administration web page). Using these functions alleviates programmers from explicitly having to deal with different places and application instances. Addition-

ally, application parameters can be overridden by URL parameters, providing more flexibility in the way that applications can be configured.

Order of widgets in the dynamically generated web interface

The development of the Public YouTube Player application also pointed out the need to be able to have control over the order in which widgets are presented in the dynamically generated web interface. An initial version of the web GUI rendered widgets in an alphabetical order. However, when comparing the public display interface of the Public YouTube Player application and its web GUI, it was apparent that the different ordering of the widgets in the two interfaces could generate confusion. As a consequence, we introduced a widget parameter that programmers can use to explicitly define the ordering of the widgets in the web interface. Although this still does not provide full control over the layout, it allows for a more natural mapping between the public display interface and the web interface, which is of particular importance for applications with many widgets, as in the case of the Public YouTube Player application.

Input state

The Everybody Votes application evidenced a shortcoming in our current implementation of the web GUI that is not yet resolved: the inexistence of input state. When users vote on a specific poll using the web interface, their answer is recorded by the public display application but not reflected on the web GUI itself. If, a few days later, users don't remember their votes, they currently have no way of finding out (although in this particular application, users can always vote again in the same or in a different option: the application records only the last vote by a given user). This problem also occurs with other interaction mechanisms, such as SMS, email, Bluetooth naming etc., and in these cases there is no obvious solution (users can always check their SMS or email message history, but this is not a practical solution). The web interface, however, has the possibility of offering a solution to this particular problem by recording the input data of some of the widgets, much like what happens in some web forms that are partially filled with our previously entered data. To be a flexible solution, applications should be able to define which widgets should keep their input state in the web interface. This feature is not yet implemented in PuReWidgets, but it is planned for a future version. Providing users with their interaction history in the web GUI is also a feature that may mitigate some of the problems associated with the lack of input state (this feature is described in section 8.2.3 – Interaction history in the web GUI).

Different widgets for authenticated users

The Everybody Votes application evidenced another possible enhancement: the possibility of accepting input from authenticated users only. Currently, applications

have no way of specifying that they wish to receive input only from authenticated users. In the EveryBody Votes application this means that interactions from a touch-screen for example, will all be considered to be from the same "Anonymous" user, making it impossible to correctly count the votes. Input from the web GUI, if not authenticated, will still be distinguished (the web GUI generates anonymous ids for each device) but they still represent anonymous users. For some applications, it may be useful to prevent anonymous input. This would disable un-authenticated input mechanisms such as touch-screens and prevent anonymous interactions through the web GUI for that particular application, or application widget.

Rapidly changing widgets

While developing the Wrod Game application we realised another limitation of the current implementation of the web GUI: its inability to cope with rapidly changing widgets. In the current version of PuReWidgets, the web interface uses a polling approach to periodically ask the IM server for updates about the application's widgets. This polling approach was used mainly to limit the number of used channels (persistent connections with the server) due to Appengine free quota limitations. However, for applications that change their interface frequently, by adding or removing widgets, or by changing the description of existing ones, this polling approach results in temporarily out-of-sync interfaces. We noticed this particularly in the Wrod Game application, which changes the long description of the text box widget to reflect the current anagram. Frequently, the anagram displayed in the web interface was not the same as the one displayed in the public display, leading to users submitting wrong guesses. This is not a major problem, as the text box on the web GUI does not necessarily need to display the anagram: users would most likely look at the public display to see the letters. Still, this problem may be addressed with persistent connections, making sure the web interface is updated at a fast enough pace. This solution, however, may drastically increase the number of channels used, as they will depend on the number of different users that interact with the display system.

7.4 API Usability

We conducted a usability evaluation of the PuReWidgets toolkit by asking a group of programmers to use it during a programming session. We were interested in receiving feedback about the toolkit's concepts (places, applications, widgets, input feedback, IM server, on-screen and off-screen widgets), the API of the library, and its documentation, in order to further improve it.

Regarding the API usage, 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[5], and API javadocs[6]) and how to improve it.

We asked a group of programmers to use our toolkit through a series of pre-defined programming tasks in a lab environment. To select participants, we sent out an email inviting students from a computer engineering course to participate in our study (participation had a monetary reward of 20 euros). We specifically asked for participants with programming experience with the Java programming language and with the Eclipse Integrated Development Environment (IDE). We further asked participants to fill in a questionnaire for assessing their programming experience and demographic information (see appendix B.1).

7.4.1 Participants

Six programmers participated in our study, all male, aged between 21 and 24 years. All participants were experienced programmers with at least four years of programming experience and an average of six years of experience. They all stated to be experienced with the Java programming language and additionally, all had experience with C++. Four participants also said they had web development experience with Javascript, and all had experience with HTML, CSS, and JSON formats. They all had already used external APIs to get data into web applications (Facebook, Twitter, Google Maps, Reddit, were some of the APIs participants had experience with). They all had experience with the Eclipse development tool. None of the participants was familiar with the Google Web Toolkit framework for web development, however they all had experience with some other web development framework (jQuery, Ruby on Rails, Asp.Net, PHP, and Bootstrap, were the frameworks participants listed).

7.4.2 Procedure

The study had three main phases: an initial presentation by the researcher, a set of programming tasks, and a final questionnaire. In total, the study lasted approximately 4 hours.

In the initial presentation, we presented the study and its purpose, and we introduced participants to the PuReWidgets 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, which explains the main concepts around PuReWidgets, explains how to setup the development environment, presents a HelloWorld application, and explains how to test and 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 total, this phase of the study took less than 40 minutes.

[5] `https://code.google.com/p/purewidgets/wiki/TableOfContents`
[6] `http://purewidgets.googlecode.com/git/doc-public/index.html`

In the next phase, we gave participants written instructions about a series of programming tasks. We asked participants to complete those tasks, and use the wiki and javadoc documentation whenever needed, but also to freely ask the researcher for help when they had any doubts (the researcher would direct the participants to the wiki or javadoc documentation, if their question was answered there; otherwise he would give specific instructions). 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. This questionnaire (see appendix B.2) was meant to assess if participants felt they understood the various concepts, their opinion regarding the documentation, and general ideas on how to improve the toolkit.

7.4.3 Programming tasks

We asked participants to perform four programming tasks with PuReWidgets. 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 customise the input feedback messages of the toolkit. Additionally, to implement these tasks participants had to deal with issues such as assigning ids to widgets based on external objects ids (pictures, in this case).

Task 1 – HelloWorld

The first task was a warm up task that consisted in changing the existing HelloWorld application that was described in the initial presentation. The HelloWorld application consists of a single button placed in the centre of the screen. Activating the button toggles the background colour between white and black. 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

The second and subsequent tasks consisted of creating a picture slide show application. In order to allow participants to focus on the interaction aspects, and to allow enough time for them to complete the various tasks, a skeleton source code project – Slideshow – was provided. This project 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 this second task, participants were asked to open the SlideShow project and make the following changes (Figure 7.7 shows an example of the intended application):

- Add a button to each thumbnail that, when activated, displays the corresponding photo in a large view. Only three thumbnails should be visible but there should be additional three hidden thumbnails in the list. The hidden ones should also be available to be selected by users.
- When an image is displayed in the large view, the corresponding thumbnail should be removed from the thumbnail list, and another thumbnail, corresponding to a new photo from the photo service, added at the end. (When a thumbnail is removed, it should no longer be available to be selected).

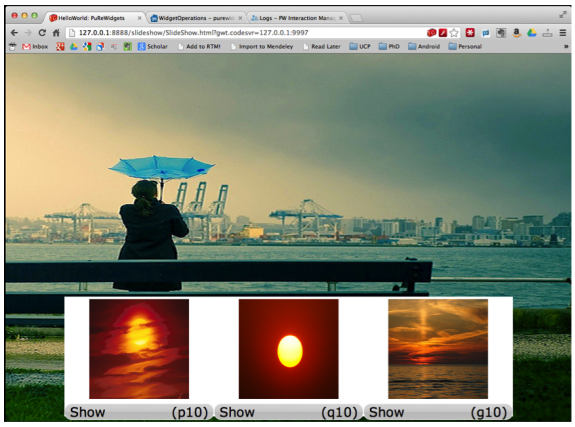


Figure 7.7: Sample screenshot of the resulting application from task 2.

Task 3 – Multi-user

In task three participants should change the previous application in order to only display the large view image after two different users had selected the same thumbnail (Figure 7.8 shows an example of the intended application).

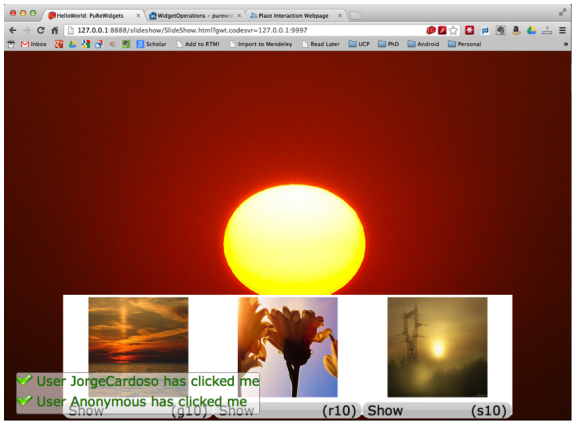


Figure 7.8: Sample screenshot of the resulting application from task 3.

Task 4 – Enhanced feedback

In the fourth and last task, participants were asked to change their previous implementations so that the slide show application would show the following feedback

messages: "User <username> selected this photo", for on-screen widgets; and "User <username> selected photo <phototitle>", for off-screen widgets (Figure 7.9 shows an example of the intended application).

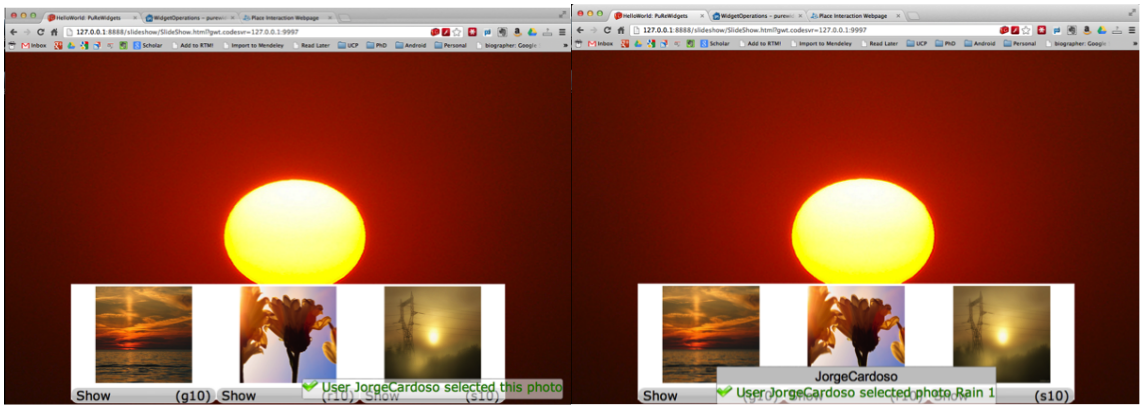


Figure 7.9: Sample screenshot of the resulting application from task 4.

7.4.4 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.

Source code

Inspection of the source code produced by the various participants revealed that in general all participants were able to accomplish the tasks, except for task 4, which was successfully completed by only three participants.

Task 1 was completed by all participants, even though participants interpreted the task differently. The objective of the task was to make a two-step interaction for setting the background colour of the application: first select a colour from the list box, and then activate the button to effectively change the colour of the background. The task was designed this way simply to make it more complicated to implement, as the more direct way would be to use a single list box widget that immediately changed the colour when selected. Most participants gave this last interpretation to the task, keeping the button in its original function of toggling between white and black colours, and adding a list box for selecting other colours (red, green, and blue). One participant removed the button altogether, and only two followed the intended meaning of the task.

Task 2 was also generally successfully completed. All participants were able to add a button to each thumbnail and make it display the corresponding image in the large slide show view. All were able to correctly assign dynamic ids to the button, using the id of the corresponding photo as part of the button's id. All participants except

one were also able to correctly delete the widget from the graphical interface and from the IM server when the button was pressed.

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 (`setOnScreenFeedbackInfo()` and `setOffScreenFeedbackInfo()`) 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.

Questionnaire

The final questionnaire was divided into 4 parts: concepts, tasks, documentation, and a final question about whether participants felt they could develop a public display application using PuReWidgets on their own. All parts except the first had a comments/suggestions box at the end, but those results are presented together with the comments recorded in the video of the session in the next section.

The first part of the questionnaire asked participants if they understood the main concepts around the PuReWidgets toolkit. Participants were asked to score on a 1 to 5 scale how much they agreed with a set of questions regarding the various concepts (see Figure 7.10). In general, participants scored each statement highly, indicating that they felt they understood the concepts. The widget concept was the one that scored worst (three participants gave a score of 5, one participant gave a score of 4, another gave a score of 3, and another gave a score of 2).

The second part of the questionnaire addressed the programming tasks, asking if participants considered that they had successfully completed the tasks and if they understood the relevant programming functions. Participants were asked to state "true" or "false" regarding the questions about whether they successfully completed the tasks, and to score on a 1 to 5 scale how much they agreed with a set of questions about the various programming functions (see Figure 7.11 and Figure 7.12).

The third part of the questionnaire addressed the toolkit documentation, asking if participants used the documentation and how clear it was. Participants were asked to answer yes/no to whether they used the wiki and the javadoc documentation,

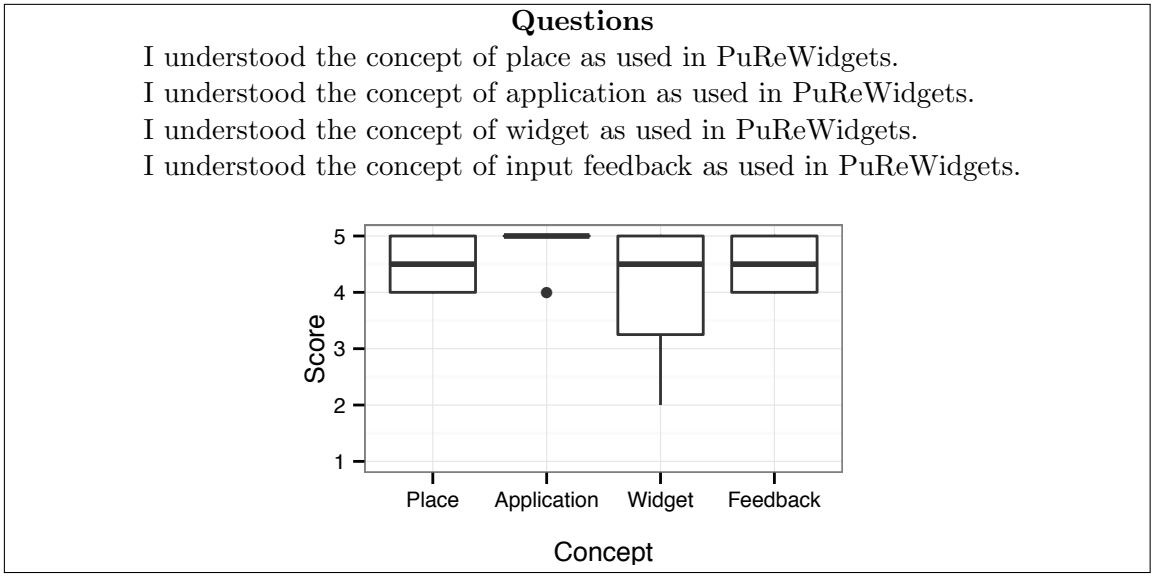


Figure 7.10: Results from the questions regarding the understanding of the various concepts associated with PuReWidgets.

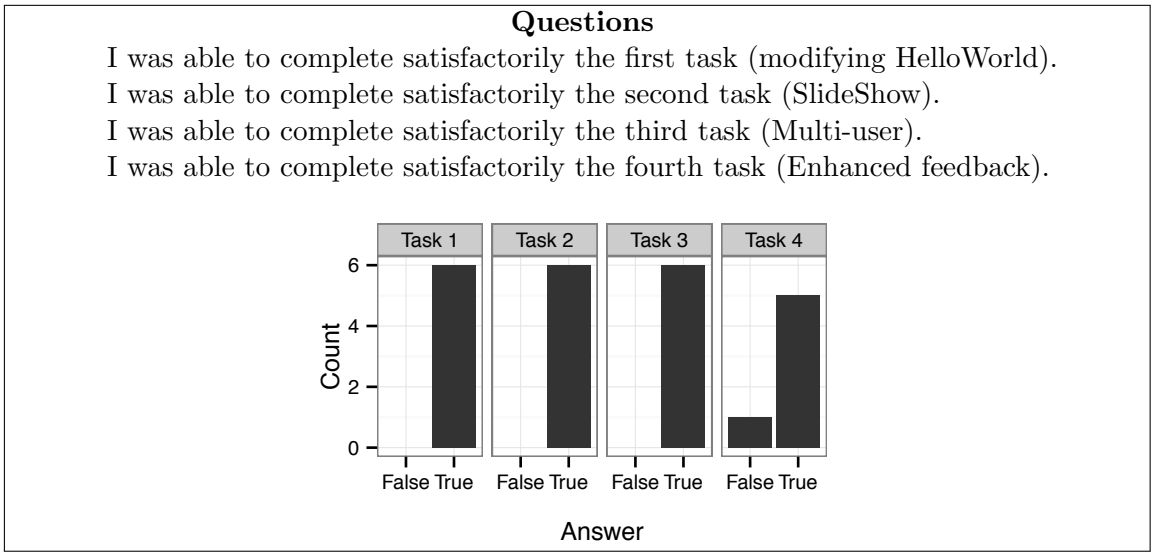


Figure 7.11: Results from the questions regarding the completion of tasks.

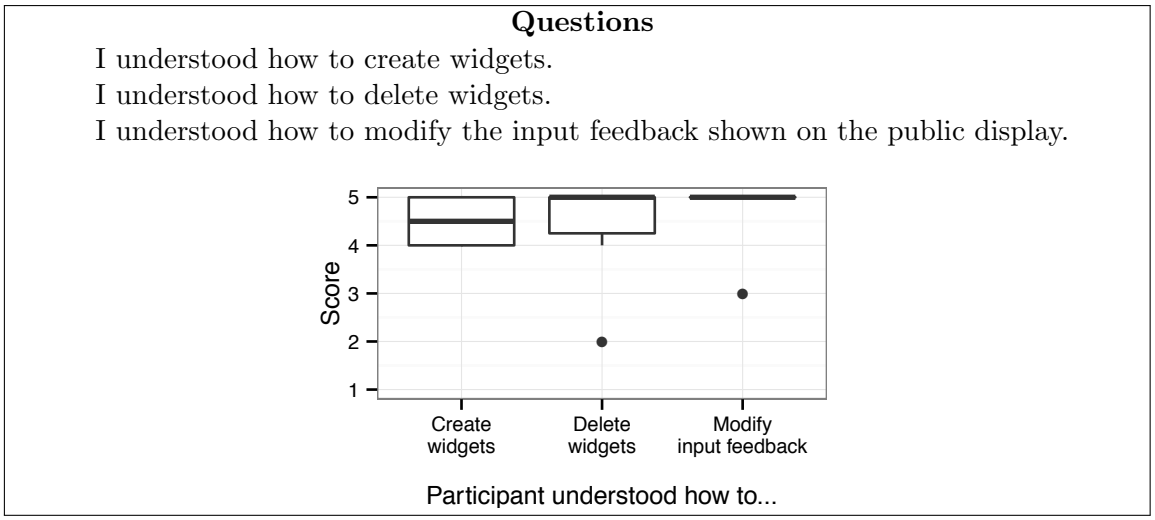


Figure 7.12: Results from the questions regarding the programming functions.

and to score on a 1 to 5 scale how much they agreed with a set of questions about the clearness of the documentation (see Figure 7.13 and Figure 7.14).

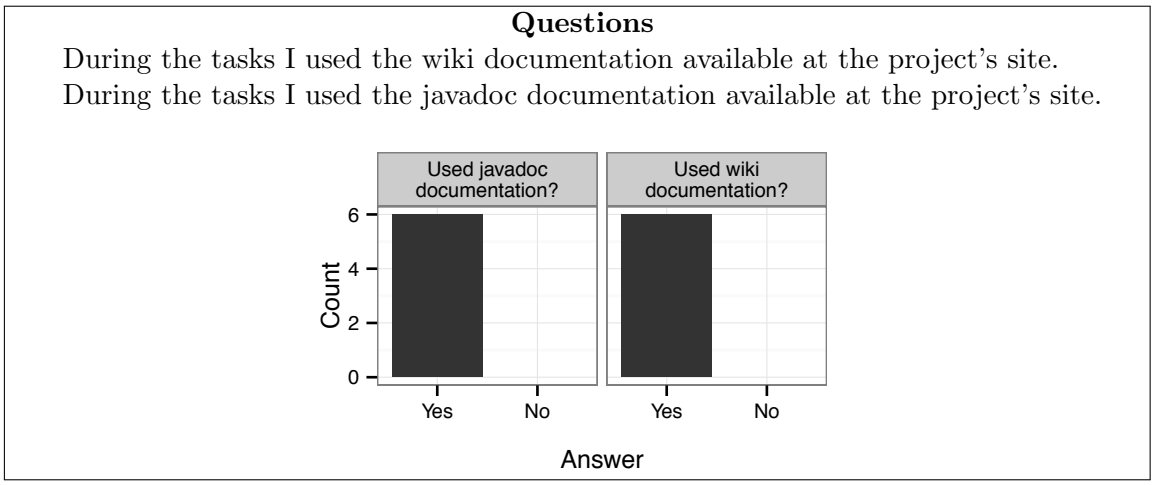


Figure 7.13: Results from the questions regarding the usage of documentation.

The last part of the questionnaire consisted of a single question to assess how confident participants were about using the PuReWidgets toolkit on their own (see Figure 7.15).

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, and the comments for the various parts of the questionnaire. We transcribed the comments that participants made from the video recording of the session and then analysed the complete set of comments from the video recording and questionnaires together. We classified the comments into 3 categories: documentation improvement, new toolkit features, and confusing aspects of the toolkit.

Questions

I think the wiki documentation was clear.
I think the javadoc documentation was clear.
I think the wiki and javadoc documentation was sufficient to complete the tasks.

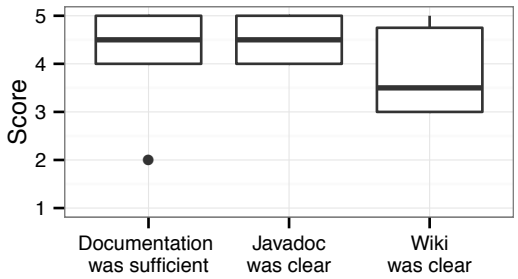


Figure 7.14: Results from the questions regarding the quality of documentation.

Question

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

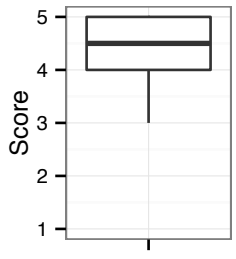


Figure 7.15: Results from question regarding participants confidence in using PuReWidgets to create their own applications.

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*"

> "*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) . . .*"

> "*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 or manual widget deletion from the Widget Manager, automatic widget id sanitization, and a loading indicator on the dynamically generated web GUI:

> "*In the Widget Manager there should be something like a garbage collector or a manual widget removal.*"

> "*It would be also interesting that, when an invalid id was entered, that id was 'cleaned' internally (for example, 'http://www' would be converted to 'httpwww') to avoid problems during development.*"

> "*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. Essentially participants expressed confusion 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.*"

7.4.5 Discussion

Even after only a very short contact with the toolkit, participants answered in a generally positive and confident way about PuReWidgets. The study showed that participants were able to understand the toolkit and use its documentation to complete the various tasks. Participants were generally confident that they could create an interactive public display application by themselves using PuReWidgets. The study did not uncover any major flaws in the toolkit's API or documentation, so we are reasonably confident that the PuReWidgets toolkit is already usable as is. However, the purpose of this study was to uncover aspects that could be improved so in this discussion we focus on these aspects.

Perhaps the most salient problem that the 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. There are several issues related to this that need to be addressed in a future version of the toolkit, particularly its documentation. The first issue is clearly communicating the concept of on-screen and off-screen widget. We did not include in the questionnaire an explicit question about the concepts of on-screen vs off-screen widget, but we believe participants factored these concepts together with the concept of widget, which accounts for the lower score of the statement about the concept of widget. Traditionally, widgets are associated with graphical objects on a display, so participants may have had 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. This can be achieved not only by expanding the textual explanation of what a widget is, but also by including diagrams or videos representing the different graphical states of a widget. Another issue is conveying more precisely why the two graphical states of a widget (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, a third issue that is currently missing in the documentation is 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 accounted for by the lack of documentation regarding at which point the input feedback is triggered. Currently, input feedback is displayed immediately *before* the application received the event so configuring the feedback message inside the callback method has no effect on the first feedback for that widget. A flowchart diagram in the documentation may help developers better understand the sequence of events that happens when an input arrives at the application, minimising the confusion about the configuration of the feedback messages.

Another problem that this study highlighted, particularly from the participants'

comments, is the need for a more flexible control of the IM during development. During the study, participants often changed how widget ids were generated, resulting in various unused widgets registered in the IM server, which makes testing the application more difficult, particularly if developers wish to use the dynamically generated web GUI for interaction. Participants suggested providing a development console on the IM that would allow for manual removal of the unused widgets. Another suggestion was to provide automatic removal feature similar to a garbage collector. This last suggestion is not directly applicable: it is generally not possible to determine if a widget that is currently not in use will never be used again by the application. In order to implement such a feature, the toolkit would need to impose harder restrictions on the way that widgets are managed, for example requiring that all widgets were created at the start of an application. However, this would limit the flexibility of the applications and go against our design principles. A possible solution would be to include this feature as a development mode feature, easily turned on or off by developers.

The study also pointed out a few aspects that can be improved regarding how widget ids are handled. The first aspect is to complete the documentation about the restrictions on the length and on the characters that can be used on a widget id. The second is to consider the suggestion of one of the participants and implement validation and sanitization of the widget id in order to make sure that invalid ids are detected as early as possible and not passed on to the IM server.

7.5 End-user Study

In order to evaluate the interaction with applications developed with PuReWidgets we deployed an interactive public display and analysed users' interactions with it.

The goal of this study was to find out what problems would arise during a real world deployment of interactive applications developed with PuReWidgets and if/how the toolkit could be improved in a future version to mitigate those problems. We were particularly interested in finding out any issues related to the user interaction process: finding out that an application was interactive, determining how to interact, and determining the result of the interaction.

7.5.1 Display configuration

For this study, we used a public display that was already in use at the School of Arts of the Portuguese Catholic University. The display was installed at the school's bar (see Figure 7.16) and had been used to show non-interactive content such as institutional videos.

For this study, we created a content schedule that included non-interactive applications and the three interactive applications developed with PuReWidgets described

in section 7.3. The complete set of applications that were configured in the public display was:

- Interaction applications
  - Public Youtube Player
  - Everybody Votes
  - Wrod Game
- Non-interactive applications
  - RSS news feed - displays news feeds from a local newspaper website.
  - Weather application - displays the weather for the local city.
  - Local videos - displays institutional videos.
  - Interaction instructions - displays instructions for interacting with the applications on the display. The application displays the system addresses and examples of how to interact via SMS, email, Web, and QR codes.

Figure 7.16: The bar of the School of Arts with the display at the top of the front wall.

For driving the content of the public display, we created a simple time-based scheduler that looped through a list of content items, where a content item consists of the web address of the item and the duration on the public display. The scheduler displayed one item at a time, in fullscreen mode. The scheduler software consists of a Google Chrome extension and it is available at `https://code.google.com/p/public-display-scheduler/`. Although the schedule varied for different day time periods (morning, lunch, afternoon), the general content schedule for the display was:

- Interaction instructions - 40 seconds.
- Public Youtube Player - 6 minutes.
- Interaction instructions - 40 seconds.
- Everybody Votes - 3 minutes.
- Interaction instructions - 40 seconds.
- Wrod Game - 3 minutes.
- RSS news feed - 2 minutes.
- Weather application - 30 seconds.
- Local videos - variable time (depends on current video).
- Interaction instructions - 40 seconds.

The display was configured to run from 9am to 6.30pm.

7.5.2 Participants

In order to have a group of users interacting periodically with the display for long enough, 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.

To select participants, we sent out an email to the staff's and PhD students' mailing lists of the school, asking for volunteers for the study. Four people answered the request, and participated in the study (two teachers, and two PhD students). None of the participants had used the interactive public display system before.

7.5.3 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 printed flyers with interaction instructions at the bar. 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 were distributed regularly – usually every other day – at the bar. These flyers, in addition to the public display application that displays interaction instructions directly on public display, were the only source of 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 prepared 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.

7.5.4 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 textboxes, and choose options from list boxes. However, they faced a few initial difficulties. We present the main issues pointed out by participants and also some of their suggestions for improving the system.

There were three main issues identified by participants during the interaction. The first 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 memorise the display system address, 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:

> "*some times 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 immediate 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*".

Participants also had some suggestions to make the system easier to use. A common suggestion to all participants was 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. One participant suggested that the applications should have a longer display time in order to allow users more time to read the information. A final suggestion by one participant was to have simpler reference codes for SMS interaction, in some situations:

> "*for example, on the football championship poll, instead of having arbitrary references codes, why not simply use the football team names?*"

7.5.5 Discussion

The fact that participants expressed no particular difficulty in using the various interaction mechanisms, and different mechanisms to interact with the same application feature, is a positive result that supports one of the main functions of the PuReWidgets toolkit: abstracting input.

The issues encountered and described by participants when they interacted with the display system point to possible enhancements to PuReWidgets. The first issue was about knowing the system's interaction mechanism addresses. Participants suggested having a permanent poster with the instructions near the display. This may be a solution in some situations, but display owners may not have this possibility for various reasons. PuReWidgets can provide some support for this issue. We already had at the time of the study an Interaction Instructions application that showed the same information that was printed on the flyers. Participants did not find this application very useful because of its limited display time. However, this limitation was imposed by the display scheduler software that was used for that study, which only supported displaying one application at a time. For displays without this restriction, the Interaction Instructions application could be configured to be always visible, for example as a ticker tape at the bottom of the screen, showing the various interaction possibilities in sequence. To provide more flexibility, as a result of this study, we have also created an Interaction Instructions widget for in-app instructions, that applications can choose to display at any time. This widget follows the ticker tape model, and allows applications to display, resize, and place it anywhere

on the screen. Using the application's screen space to display instructions can be an alternative for schedulers that do not support multiple applications at a time. If applications provide display owners with configuration options for enabling and disabling the display of instructions, the solution can be used for any kind of scheduler: display owners can choose to display the Interaction Instructions application and disable the in-app instructions, or the other way around. (We are currently considering providing this in-app instructions as a feature on any PuReWidgets application, so that application developers don't even need to worry about it.)

The second issue was about the difficulty of understanding the interaction with off-screen applications. Some participants expected applications to immediately appear on the display if they interacted with it. This is a problem created by the asynchronous interaction, which introduces an application model with which most users are not familiar. One way to mitigate this problem is to provide better feedback to the user about what is going to happen with his input. In the version used during the study, the web GUI showed only a standard "input sent" message as feedback to the user's interaction. As a result of this study, we have enhanced the feedback to provide more information. The current version detects if the target application is on-screen or off-screen and provides different messages to the user. If the application is on-screen, it instructs the user to look at the display to see the result of his interaction. If the application is off-screen it informs the user that it may take a while for him to be able to see the reaction of the application (currently, it's not possible to predict how long it will take). We have also planned a new version of the dynamically generated web GUI that allows applications to customise the feedback message (see section 8.2.3 – Application-specific feedback on the web GUI). However this applies only to this particular interaction mechanism. A more general solution would be to display feedback for interaction with off-screen applications directly on the public display itself (this topic is elaborated in section 8.2.3 – Feedback for off-screen application on the public display – in the next chapter).

The third issue was about interaction when away from the public display. We believe this to be a minor issue, as most users will probably not even try to interact with the display system if they are not near any display. Our intention is that interaction occurs near the public displays with which users are interacting, even if we did not take any active step to prevent otherwise. Some display systems actively prevent users from interacting if they are not near the display – automatically by detecting the user's position, or manually by requiring users to enter a code that is shown on the display. By not taking this approach, we have allowed the confusion expressed by one of the participants in this study. A possible way to mitigate this confusion is to better communicate in the dynamically generated web interface, that the interface is for interacting with a public display system and encourage users to be near the display.

One participant suggested that reference codes could be made simpler to memorise. This feature is already implemented in PuReWidgets: applications can suggest their own reference codes to be used in each widget. If there is no conflict with other already in use reference codes, the IM will honour the application's request. This was already implemented during the study, however the applications we have developed do not take advantage of this feature.

7.6 Conclusion

We have presented an evaluation of the PuReWidgets toolkit on a broad set of dimensions. By doing a broad evaluation, we aimed at obtaining a general view of the toolkit, making sure we covered all relevant aspects of its use and didn't miss any critical issue.

The system's performance showed that the toolkit's implementation can handle a large number of applications. Testing found no bottlenecks in the implementation, supporting the claim that it scales well with the number of applications.

The toolkit has already suffered various iterations in response to the simultaneous development of three real-world interactive applications. In addition, testing the PuReWidgets library API with real programmers revealed that programmers will be able to easily grasp and explore its concepts. Even only after a short exposure, first time programmers were able to use it to create a simple application without major problems. Based on our experience developing public display applications with PuReWidgets and on the testing with programmers, we are confident that the toolkit's features are appropriate for the kinds of public display applications we were aiming at, and that it will give appropriate support for novel interactive applications for public displays. Improvements suggestions (mostly documentation) found during this study will be integrated into a future version.

Tests with end users and a real world deployment found additional interaction issues that a toolkit may help to solve, even though many of those issues are not directly related to the toolkit itself. One example of this is providing in-app interaction instructions so that applications and display owners have more flexibility in choosing how instructions are presented to users. This is a feature already present in the latest version of the toolkit.

Chapter 8

Conclusions

Contents

Public digital displays have become increasingly ubiquitous in our technological landscape, but they are still under-explored, considering their flexibility and communication potential. Public displays can become an important communication channel in the future and even reach the attention and usage that smartphones have today. For that to happen, however, there are several aspects that need to be improved and interactivity is one of the most important. Only by thinking about and designing new, engaging, and relevant interactions with public displays can we raise their value for society. Up until now, interactive public displays have been mainly the business of the research community, exploring the potential of this new medium with various ad-hoc interactive systems.

The general goal of this work was to develop high-level interaction abstractions and tools for public display applications that support public, shared, remote interaction, abstracting the low-level details of various input mechanisms. By providing high-level abstractions and tools, designers and developers can focus on the user experience of the interaction with a public display application without being caught up by the low-level details and specificities of a particular interaction mechanism. This general goal was divided in three objectives.

The first objective was to analyse public display interaction from the point of view of the information that is generated as a side effect of the interaction – digital footprints – and that may be used as the basis for content adaptation behaviour on public displays. This analysis resulted in a framework of digital footprints for socially-aware public display systems that helps designers to create public display systems that continually adapt to their social environment.

The second objective was to identify and characterise a set of interaction tasks and controls that are appropriate for public display interaction. This should provide developers with a conceptual tool that enable them to think about the interaction features of an application without worrying about low-level input mechanism details. It should also provide the basis for the development of software toolkits for the development of interactive public display applications.

The third objective was to design, implement, and evaluate a software toolkit that provides interaction abstractions that programmers can use for incorporating interactive features into their public display applications. Just like we have various toolkits for creating graphical user interfaces for desktop applications, this toolkit should be an initial effort at an equivalent toolkit for public display applications.

8.1 Contributions

While pursuing our objectives, we have developed a toolkit of abstractions, which includes several contributions to the research community that we now summarise.

8.1.1 Digital footprints for socially-aware public displays

The first contribution was the analysis of public display interaction from the point of view of the digital footprints they leave behind and that may feed several types of content adaptation behaviour on public displays. The digital footprints focus on the high-level information about the user or audience that can be extracted from the interaction, abstracting the concrete interaction features that generate that information.

We have developed a framework of digital footprints for socially-aware public displays that consists of four digital footprints: presence sensing, self-exposure, user-generated content, and actionables. We have analysed how these digital footprints can be generated by various types of interaction features in public displays, using various interaction mechanisms.

Several strategies can be used to take advantage of these footprints with the ultimate goal of creating more relevant displays that are able to automatically adapt to their environment. We have mapped the digital footprints to various types of content adaptation and audience characterisation strategies such as attraction loops, audience measurement, demographic targeting, contextual targeting, behavioural targeting, optimisation of individual content exposure, and impact assessment.

8.1.2 Interaction tasks and controls for public displays

The second contribution was the characterisation of the interaction tasks and controls that are appropriate for public display interaction. We made a comprehensive survey of public display systems and analysed the interaction features they provide from the perspective of the information that is exchanged between the user and the system. Using the concept of interaction task as defined by [Foley et al., 1980], we characterised six interaction tasks for public displays: select, data entry, upload, download, signal presence, and dynamic manipulation. We characterised each interaction task with properties and possible values for those properties. We developed an initial set of interaction controls for public displays that correspond to specialised forms of the interaction tasks. These controls abstract the concrete interaction mechanisms used to interact and provide high-level interaction events to public display applications. We also developed a design space that maps several interaction mechanisms, the interaction tasks they support, and examples of concrete implementations from existing systems. Together, the list of interaction tasks, controls, and the mapping between interaction tasks and interaction mechanisms can be used by system designers to decide what controls to provide, what interaction mechanisms to support, and how to implement those controls using a specific interaction mechanism. This design space formed the basis for our toolkit, and can form the basis for other interaction toolkits in the future.

8.1.3 A programming toolkit for public displays

A third contribution was the identification of the fundamental requirements for an interaction abstraction toolkit for public displays. We have analysed the interaction environment surrounding public displays and used it to frame a number of requirements that should be met by interaction toolkits. We then used these requirements as the basis for the development of our solution.

A fourth contribution was the programming toolkit itself. We have designed and implemented a toolkit that provides to any programmer the ability to easily create and deploy interactive applications for public displays. We have built PuReWidgets around the interaction controls and interaction requirements identified previously in this work. PuReWidgets has the following main features:

Multiple, extensible, widgets The toolkit incorporates various types of interaction widgets, supporting the previously identified interaction tasks: select, data entry, upload, download, and check-in. Existing widgets can be customized and composed into new widgets, and completely new widgets can be created by application programmers.

Dynamically generated graphical interfaces The toolkit automatically generates graphical user interfaces for desktop and mobile interaction with public displays. It also generates Quick Response codes (QR codes) for user interaction through camera equipped mobile devices.

Independence from specific input mechanisms and modalities The toolkit supports several interaction mechanisms such as Short Message Service (SMS), Bluetooth naming, OBject EXchange (OBEX), email, touch-displays, in addition to the already mentioned desktop, mobile, and QR code interfaces.

Asynchronous interaction The toolkit supports asynchronous interaction, allowing applications to receive input events that were generated when the application was not executing.

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

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

A fifth contribution was the in-breadth evaluation of the PuReWidgets toolkit along several dimensions, including the system's performance and scalability, the API usability, and a real-world deployment.

8.1.4 Open-source software projects

This work has also contributed to the research and programming community with several open-source software projects, allowing anyone to use, modify, and adapt for further research and development.

The following open source projects were initiated during the course of this work:

- `https://code.google.com/p/purewidgets/`
  - The PuReWidgets toolkit itself.
  - Three interactive public display applications
    - Public YouTube Player
    - Everybody Votes
    - Wrod Game
- `https://code.google.com/p/public-display-scheduler/`
  A Google Chrome extension that serves as an application scheduler.

8.2 Future Work

During the course of this work we identified some challenges related to interactive public display applications, which fell outside the core of this thesis. These challenges may provide opportunities for further research around the topic of interactive public display applications.

8.2.1 More flexible content scheduling

The most common approach for content scheduling in public displays is to follow a timetable were each content item is given a pre-determined amount of display time. This approach works well with time-based content where the content's duration is known, such as in videos, or with non-time-based content where the display owner can easily decide how much display time the content should have, as in still images or text. However, once we have rich interactive applications, the traditional scheduling approach may compromise the user's experience.

For example, our Public YouTube Player application would frequently be terminated by the display's scheduler in the middle of a video that a user had just asked to see, disturbing that user's experience using the application. Some applications may require display time in response to asynchronous events such as user interactions or other external events. For example, an application may wish to display a

calendar notification only when a specific user or group of users, who subscribed to those calendar notifications, are present. In these cases, the traditional scheduling approach does not work. For these situations, the display's content scheduler should provide a mechanism for applications to trigger short time notifications.

Interactive applications need a more flexible scheduling mechanism that allows them to negotiate extra display time, if needed, and to request display time if they have important or time-critical content to display. The challenge is making sure that display owners still have enough control over of what content is played and for how long and that applications are not allowed to misbehave and take over the display time of other applications. Some specific questions that arise from this challenge are:

- What tools do display owners need to specify their needs regarding application scheduling?
- Should application developers have some degree of control over the application's scheduling? How much, and how can it be specified?
- How can the display system combine the developer's and the display owner's needs regarding application scheduling?
- What mechanisms do we need to enforce a fair usage of the display time among the various applications?

8.2.2 Public display application framework

Another issue we faced when developing interactive applications for public displays was the lack of a proper application management. Some of our applications have to complete a considerable amount of initialisation operations before being able to display their content and react to user input. However, because our execution environment loads and immediately displays the application, users are faced with loading splash screens. This loading time could have been used to display the previous content for a bit longer. Our applications would also have benefited from the possibility of executing termination operations before they are unloaded from the public display. This would allow applications to better manage their resources by, for example, giving them time to write data to persistent local or remote storage. Currently, applications have to take aggressive strategies to make sure they don't lose data by persisting data more often than what would be necessary. Additionally, we noticed that in many situations there would be no need to unload an application to display another one. In some situations the display needs to show a given application only for a very short time, for example, a notification or alert. In our current execution environment, this would mean unloading the current application to display the alert for a short time and then re-loading the same application again. Loading and unloading may constitute a significant overhead for public display systems that switch between applications frequently.

Ultimately, what public display systems need is a complete application framework, just like we have in other platforms. For example, the Android mobile platform defines a mobile application framework that specifies, among other things, what kind of applications it supports, and what is the runtime life-cycle of an application.

For example, on Android, developers can create graphical user interface applications that are explicitly launched by users when they need to use the application; service applications that run on the background, occasionally creating notifications to alert the user; but also widget applications that have part of their graphical user interface permanently visible on the device's home screen. On public displays, however, independent applications are still a relatively new concept so various questions arise:

- Do these types of applications of the Android, or other computing platforms, make sense for public displays?
- Can we adapt them to public display systems (e.g., applications that only show up on the display when a user requests them; applications that run on the background, showing content only when there is something noteworthy to display; and application are by default visible on the display)?
- What other types of applications make sense for public display systems?

The Android platform defines a rich runtime application life-cycle that breaks down all the possible states and transitions between states of an application from the time it is loaded into memory and started, to the time it is shut down and removed from memory. This break down of possible states allows application programmers and system to negotiate the resources that an application needs in each state, guaranteeing an efficient usage of those resources on the one hand, and rapid application switching and loading, on the other hand. The transitions between applications states in Android are represented by callback functions that applications can listen to make sure they react appropriately to the transition. For example, an application may be paused if another application comes to the foreground (e.g., because the user requested another application). Applications can detect the paused state and stop animations and other CPU consuming operations and save its state to persistent storage (because paused applications may be destroyed by the system if it needs memory). When the application is resumed, it can start the animations again. It is easy to imagine that display systems will need this kind of resource management when the number of applications that each display handles grows. The basic questions to answer are:

- What applications states should exist for public display applications?
- What events can cause transitions and how should transitions be handled by applications?

8.2.3 Improvements to PuReWidgets

We have also identified various improvements that could be incorporated into PuReWidgets.

Javascript library

Although Google Web Toolkit (GWT) is a web development platform used by a large programming community, Javascript and Javascript-based libraries are also largely used today. Providing a Javascript library for PuReWidgets would open up the development of interactive public display applications to an even wider community. Re-implementing the PuReWidgets library (GWT) in Javascript means re-implementing all communication with the Interaction Manager (IM), the widget management logic for the delayed synchronisation, local storage functions, input feedback logic, and all the current widgets.

User profiles

Allowing users to create profiles available to the IM would allow the IM to better identify its users and provide applications with information that could be used to provide more interesting interactions. For example, a user profile that listed a user's phone number, email address, Bluetooth address, and the profile id for an authentication source such as Google, or Facebook, would allow the IM to associate with a single user, input from different sources (SMS, email, web GUI, Bluetooth naming) and provide applications with more accurate information about the id of the user of a given input.

Richer profiles with preferences information, biographical information, or profiles with connections to other Social Networking Site (SNS) profiles, would allow applications with access to these to better adapt to their user's preferences automatically. The challenge is achieving this integration while respecting the users' privacy and allowing them to have complete control over which application can access what information.

Interaction history in the web GUI

Providing an interaction or activity panel in the web GUI may help users in some situations. For example, in the Everybody Votes application it is easy to forget if and what option we voted for a given poll. Having an activity panel for each application would help users remember their interactions in these situations.

Given that the web GUI can be accessed via different devices, in order to provide an appropriate user experience, the activity information should be stored in a server

and synchronised across devices (although for un-authenticated users this would not be necessary, and a simple local storage solution would suffice). This feature could be combined with the previous one, so that the activity information would simply be another field of the user's profile in the IM.

Application-specific feedback on the web GUI

Currently, feedback messages on the web GUI when users interact with a given widget are generic, contrary to the feedback on the public display itself, which applications can customise. There are only two different messages - one for when users interact with on-screen applications and another when users interact with off-screen applications. These messages are always the same regardless of the application and regardless of the widget.

A future enhancement for PuReWidgets would be to allow applications to define their own feedback to appear on the web GUI. As for regular widget feedback on the public display, applications should be able to define different messages for each widget and the two situations: a message for the situation in which the application is currently on-screen and for the situation in which the application is off-screen. With well-structured messages, this would potentially mitigate some of the confusion users faced when interacting with off-screen applications.

Feedback for off-screen application on the public display

Currently, PuReWidgets is unable to display feedback on the public display for off-screen applications. There are several approaches that could be taken to provide this functionality. One approach would be to change the Widget Manager component of the PuReWidgets library in order for it to ask for input directed at all applications in the same place as the currently on-screen application, which could provide generic feedback for the input directed at other applications. However, this would only work in cases where the public display only shows one application at a time. If the display shows, for example, two applications spatially arranged on the display, that approach would result in duplicate feedback, as on-screen applications would not be aware of each other.

A better solution would be to develop a notification application and change the IM so that it detects input for off-screen applications and directs it to the notification application. This application would show generic feedback messages for the received input and, ideally, it would only be displayed for the needed duration for the feedback and as a popup window on the display, similar to what we now have with the notification centre for Mac OS X, or notification area in the Windows OS. This, however, would require a special scheduler software for the public display that would be able to display applications on-demand rather than based on a pre-defined schedule. Alternatively, the notification application could be displayed in a permanent region of the display, as a regular application.

8.3 Final Remarks

We hope that the PuReWidgets toolkit will contribute significantly to the emergence of more real-world deployments of interactive public display applications. With our toolkit of interaction abstractions, developers can concentrate on the user experience and on the creative aspects of application development. By making the toolkit open-source, we guarantee that anyone can freely use it and we hope that others will contribute to its development, or use it as inspiration for other public display toolkits. The ultimate goal is that public displays become a richer, more valuable communication medium for society.

Chapter A

List of coded papers

1. AgentSalon: Sumi, Y., & Mase, K. (2001). AgentSalon: facilitating face-to-face knowledge exchange through conversations among personal agents. Proceedings of the fifth international conference on Autonomous agents - AGENTS '01 (pp. 393–400). New York, New York, USA: ACM Press. doi:10.1145/375735.376344

2. Aware Community Portals: Sawhney, N., Wheeler, S., & Schmandt, C. (2001). Aware Community Portals: Shared Information Appliances for Transitional Spaces. Personal and Ubiquitous Computing, 5(1), 66–70. doi:10.1007/s007790170034

3. Beye & Meier: Beyer, G., & Meier, M. (2011). Music Interfaces for Novice Users : Composing Music on a Public Display with Hand Gestures. Proceedings of the International Conference on New Interfaces for Musical Expression, (pp. 507–510).

4. Blueboard: Russell, D. M., & Gossweiler, R. (2001). On the Design of Personal & Communal Large Information Scale Appliances. UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing (pp. 354–361). London, UK: Springer-Verlag.

5. Bluemusic: Mahato, H., Kern, D., Holleis, P., & Schmidt, A. (2008). Implicit personalization of public environments using bluetooth. Proceeding of the twenty-sixth annual CHI conference extended abstracts on Human factors in computing systems - CHI '08 (p. 3093). New York, New York, USA: ACM Press. doi:10.1145/1358628.1358813

6. Bluetone: Dearman, D., & Truong, K. N. (2009). BlueTone. Proceedings of the 11th international conference on Ubiquitous computing - Ubicomp '09 (p. 97). New York, New York, USA: ACM Press. doi:10.1145/1620545.1620561

7. BluScreen: Sharifi, M., Payne, T., & David, E. (2006). Public Display Advertising Based on Bluetooth Device Presence. Mobile Interaction with the Real World (MIRW 2006) in conjunction with the 8th International Conference on Human Computer Interaction with Mobile Devices and Services. Retrieved from http://www.hcilab.org/events/mirw2006/pdf/mirw2006_sharifi.pdf

8. C-Blink: Miyaoku, K., Higashino, S., & Tonomura, Y. (2004). C-blink: a hue-difference-based light signal marker for large screen interaction via any mobile terminal. UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology (pp. 147–156). New York, NY, USA: ACM. doi:http://doi.acm.org/10.1145/1029632.1029657

9. CityWall: Peltonen, P., Kurvinen, E., Salovaara, A., Jacucci, G., Ilmonen, T., Evans, J., Oulasvirta, A., et al. (2008). It's Mine, Don't Touch!: interactions at a large multi-touch display in a city centre. CHI 08 Proceeding of the twentysixth annual SIGCHI conference on Human factors in computing systems (Vol. 16, pp. 1285–1294). ACM. doi:10.1145/1357054.1357255

10. CoCollage: McCarthy, J. F., Farnham, S. D., Patel, Y., Ahuja, S., Norman, D., Hazlewood, W. R., & Lind, J. (2009). Supporting community in third places with situated social software. Proceedings of the fourth international conference on Communities and technologies - C&T '09 (p. 225). New York, New York, USA: ACM Press. doi:10.1145/1556460.1556493

11. Code Space: Bragdon, A., DeLine, R., Hinckley, K., & Morris, M. R. (2011). Code space: touch + air gesture hybrid interactions for supporting developer meetings. Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS '11 (p. 212). New York, New York, USA: ACM Press. doi:10.1145/2076354.2076393

12. ContentCascade: Raj, H., Gossweiler, R., & Milojicic, D. (2004). Contentcascade incremental content exchange between public displays and personal devices. Mobile and Ubiquitous Systems, Annual International Conference on (Vol. 0, pp. 374–381). IEEE Computer Society. doi:10.1109/MOBIQ.2004.1331744

13. Cwall: 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.

14. Digifieds: Alt, F., Kubitza, T., Bial, D., Zaidan, F., Ortel, M., Zurmaar, B., Lewen, T., et al. (2011). Digifieds: Insights into Deploying Digital Public Notice Areas in the Wild. Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia - MUM '11 (pp. 165–174). New York, New York, USA: ACM Press. doi:10.1145/2107596.2107618

15. Digital graffitti: Carter, S., Churchill, E. F., Denoue, L., Helfman, J., & Nelson, L. (2004). Digital graffiti: public annotation of multimedia content. CHI '04 extended abstracts on Human factors in computing systems (pp. 1207–1210). New York, NY, USA: ACM. doi:http://doi.acm.org/10.1145/985921.986025

16. Dynamo: Brignull, H., Izadi, S., Fitzpatrick, G., Rogers, Y., & Rodden, T. (2004). The introduction of a shared interactive surface into a communal space. Proceedings of the 2004 ACM conference on Computer supported cooperative work - CSCW '04 (p. 49). New York, New York, USA: ACM Press. doi:10.1145/1031607.1031616

17. e-Campus: Davies, N., Friday, A., Newman, P., Rutlidge, S., & Storz, O. (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 (p. 151). New York, New York, USA: ACM Press. doi:10.1145/1555816.1555832

18. FizzyVis: Coutrix, C., Kuikkaniemi, K., Kurvinen, E., Jacucci, G., Avdouevski, I., & Mäkelä, R. (2011). FizzyVis : Designing for Playful Information Browsing on a Multitouch Public Display. Proceedings of DPPI'11, Designing Pleasurable Products and Interfaces. Milan.

19. Gesture Frame: Li, Y., Groenegress, C., Strauss, W., & Fleischmann, M. (2004). Gesture Frame – A Screen Navigation System for Interactive Multimedia Kiosks. In A. Camurri & G. Volpe (Eds.), Gesture-Based Communication in Human-Computer Interaction (Vol. 2915, pp. 93–94). Springer Berlin / Heidelberg. doi:10.1007/978-3-540-24598-8_35

20. GroupCast: McCarthy, J. F., Costa, T. J., & Liongosari, E. S. (2001). UniCast, OutCast & GroupCast: Three Steps Toward Ubiquitous, Peripheral Displays. UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing (pp. 332–345). London, UK: Springer-Verlag.

21. Hello.Wall: Prante, T., Röcker, C., Streitz, N., Stenzel, R., Magerkurth, C., van Alphen, D., & Plewe, D. (2003). Hello.Wall - Beyond Ambient Displays. Video Track and Adjunct Proceedings of the 5th Intern. Conference on Ubiquitous Computing (UBICOMP'03). Seattle, Wash., USA.

22. Hermes Photo Display: Cheverst, K., Dix, A. J., Fitton, D., Kray, C., Rouncefield, M., Sas, C., Saslis-Lagoudakis, G., 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 (p. 47). New York, New York, USA: ACM Press. doi:10.1145/1085777.1085786

23. Instant Places: 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. doi:10.1109/MPRV.2008.74

24. Interactive Public Ambient Displays: Vogel, D., & 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 (p. 137). New York, New York, USA: ACM Press. doi:10.1145/1029632.1029656

25. iSchool: Zhang, S., & Jeng, W. (2011). Designing a public touchscreen display system for iSchool community. Proceedings of the 2011 iConference on - iConference '11 (pp. 808–810). New York, New York, USA: ACM Press. doi:10.1145/1940761.1940915

26. Jeon et al.: Jeon, S., Hwang, J., Kim, G. J., & Billinghurst, M. (2006). Interaction techniques in large display environments using hand-held devices. Proceedings of the ACM symposium on Virtual reality software and technology - VRST '06 (p. 100). New York, New York, USA: ACM Press. doi:10.1145/1180495.1180516

27. JoeBlogg: Martin, K., Penn, A., & Gavin, L. (2006). Engaging with a situated display via picture messaging. CHI '06 extended abstracts on Human factors in computing systems - CHI '06 (p. 1079). New York, New York, USA: ACM Press. doi:10.1145/1125451.1125656

28. Jukola: O'Hara, K., Lipson, M., Jansen, M., Unger, A., Jeffries, H., & Macer, P. (2004). Jukola: Democratic Music Choice in a Public Space. Proceedings of the 2004 conference on Designing interactive systems processes, practices, methods, and techniques - DIS '04 (p. 145). New York, New York, USA: ACM Press. doi:10.1145/1013115.1013136

29. Locamoda: LocaModa. (2010). LocaModa App Store. Retrieved from http://locamoda.com/apps/

30. Looking Glass: Müller, J., Walter, R., Bailly, G., Nischt, M., & Alt, F. (2012). Looking glass: a field study on noticing interactivity of a shop window. Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12 (p. 297). New York, New York, USA: ACM Press. doi:10.1145/2207676.2207718

31. MAID: Salvador, R., & Romão, T. (2011). Let's move and save some energy. Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology - ACE '11 (p. 1). New York, New York, USA: ACM Press. doi:10.1145/2071423.2071527

32. Mobile Service Toolkit: Toye, E., Sharp, R., Madhavapeddy, A., & Scott, D. (2005). Using smart phones to access site-specific services. IEEE Pervasive Computing, 4(2), 60–66. doi:10.1109/MPRV.2005.44

33. Mobilenin: 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 - MULTIMEDIA '05 (p. 199). New York, New York, USA: ACM Press. doi:10.1145/1101149.1101178

34. Notification Collage: Greenberg, S., & Rounding, M. (2001). The Notification Collage: Posting Information to Public and Personal Displays. Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 515–521). Seattle, Washington, United States: ACM. doi:10.1145/365024.365339

35. Opinionizer: Rogers, Y., & Brignull, H. (2002). Subtle ice-breaking: encouraging socializing and interaction around a large public display. CSCW'02 Workshop Proceedings.

36. OutCast: McCarthy, J. F., Costa, T. J., & Liongosari, E. S. (2001). UniCast, OutCast & GroupCast: Three Steps Toward Ubiquitous, Peripheral Displays. UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing (pp. 332–345). London, UK: Springer-Verlag.

37. Pendle: Villar, N., Kortuem, G., Van Laerhoven, K., & Schmidt, A. (2005). The Pendle: A Personal Mediator for Mixed Initiative Environments. Retrieved from http://eprints.lancs.ac.uk/12662/1/pendle-ie05.pdf

38. Plasma Posters: Churchill, E. F., Nelson, L., Denoue, L., Helfman, J., & Murphy, P. (2004). Sharing multimedia content with interactive public displays. Proceedings of the 2004 conference on Designing interactive systems processes, practices, methods, and techniques - DIS '04 (pp. 7–16). New York, New York, USA: ACM Press. doi:10.1145/1013115.1013119

39. Point & Shoot: Ballagas, R., Rohs, M., & Sheridan, J. G. (2005). Sweep and Point & Shoot: Phonecam-Based Interactions for Large Public Displays. CHI '05: CHI '05 extended abstracts on Human factors in computing systems (pp. 1200–1203). New York, NY, USA: ACM. doi:10.1145/1056808.1056876

40. Proactive displays: 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. doi:10.1145/1314683.1314684

41. Publix: Ventura, P., Sousa, H., & Jorge, J. (2008). Mobile Phone Interaction with Outdoor Advertisements. Workshop on Designing and evaluating mobile phone-based interaction with public displays. CHI2008. Florence.

42. ReflectiveSigns: Müller, J., Exeler, J., Buzeck, M., & Krüger, A. (2009). ReflectiveSigns: Digital Signs That Adapt to Audience Attention. In H. Tokuda, M. Beigl, A. Friday, A. J. B. Brush, & Y. Tobe (Eds.), Proceedings of the 7th International Conference on Pervasive Computing (Vol. 5538, pp. 17–24). Nara, Japan: Springer-Verlag. doi:10.1007/978-3-642-01516-8

43. Remote Commander: Myers, B. A., Stiel, H., & Gargiulo, R. (1998). Collaboration using multiple PDAs connected to a PC. CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work (pp. 285–294). New York, NY, USA: ACM. doi:http://doi.acm.org/10.1145/289444.289503

44. Semi-public displays: Huang, E. M., & Mynatt, E. D. (2003). Semi-public displays for small, co-located groups. CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 49–56). New York, NY, USA: ACM. doi:http://doi.acm.org/10.1145/642611.642622

45. SmartKiosk: Rehg, J. M., Loughlin, M., & Waters, K. (1997). Vision for a smart kiosk. Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97) (p. 690). IEEE Computer Society. Retrieved from http://dl.acm.org/citation.cfm?id=794189.794413

46. Spalendar: Chen, X. A., Boring, S., Carpendale, S., Tang, A., & Greenberg, S. (2012). SPALENDAR: Visualizing a Group's Calendar Events over a Geographic Space on a Public Display. Design. Calgary, AB, Canada. Retrieved from http://grouplab.cpsc.ucalgary.ca/grouplab/uploads/Publications/Publications/2012-Spalendar.Report2012-1018-01.pdf

47. Sweep: Ballagas, R., Rohs, M., & Sheridan, J. G. (2005). Sweep and Point & Shoot: Phonecam-Based Interactions for Large Public Displays. CHI '05: CHI '05 extended abstracts on Human factors in computing systems (pp. 1200–1203). New York, NY, USA: ACM. doi:10.1145/1056808.1056876

48. Touch & Interact: Hardy, R., & Rukzio, E. (2008). Touch & interact: touch-based interaction of mobile phones with displays. In G. H. ter Hofte, I. Mulder, & B. E. R. de Ruyter (Eds.), Mobile HCI (pp. 245–254). ACM. Retrieved from http://dblp.uni-trier.de/db/conf/mhci/mhci2008.html#HardyR08

49. UBI-hotspot: Ojala, T., Kukka, H., Lindén, T., Heikkinen, T., Jurmu, M., Hosio, S., & Kruger, F. (2010). UBI-Hotspot 1.0: Large-Scale Long-Term Deployment of Interactive Public Displays in a City Center. 2010 Fifth International Conference on Internet and Web Applications and Services (pp. 285–294). IEEE. doi:10.1109/ICIW.2010.49

50. VisionWand: Cao, X., & Balakrishnan, R. (2003). VisionWand: Interaction Techniques for Large Displays using a Passive Wand Tracked in 3D. Proceedings of the 16th annual ACM symposium on User interface software and technology - UIST '03 (Vol. 5, pp. 173–182). New York, New York, USA: ACM Press. doi:10.1145/964696.964716

51. Vista: Wichary, M., Gunawan, L., Van Den Ende, N., Hjortzberg-Nordlund, Q., Matysiak, A., Janssen, R., & Sun, X. (2005). Vista: interactive coffee-corner display. CHI 05 CHI 05 extended abstracts on Human factors in computing systems (pp. 1062–1077). ACM. doi:10.1145/1056808.1056818

52. Visual code widgets: 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. doi:10.1109/ICDCSW.2005.140

53. Vodafone Cube: Ydreams. (2003). Vodafone Cube. Retrieved from http://www.ydreams.com/#/en/projects/publicurbanexperiences/giantinteractivebillboardsvodafone/

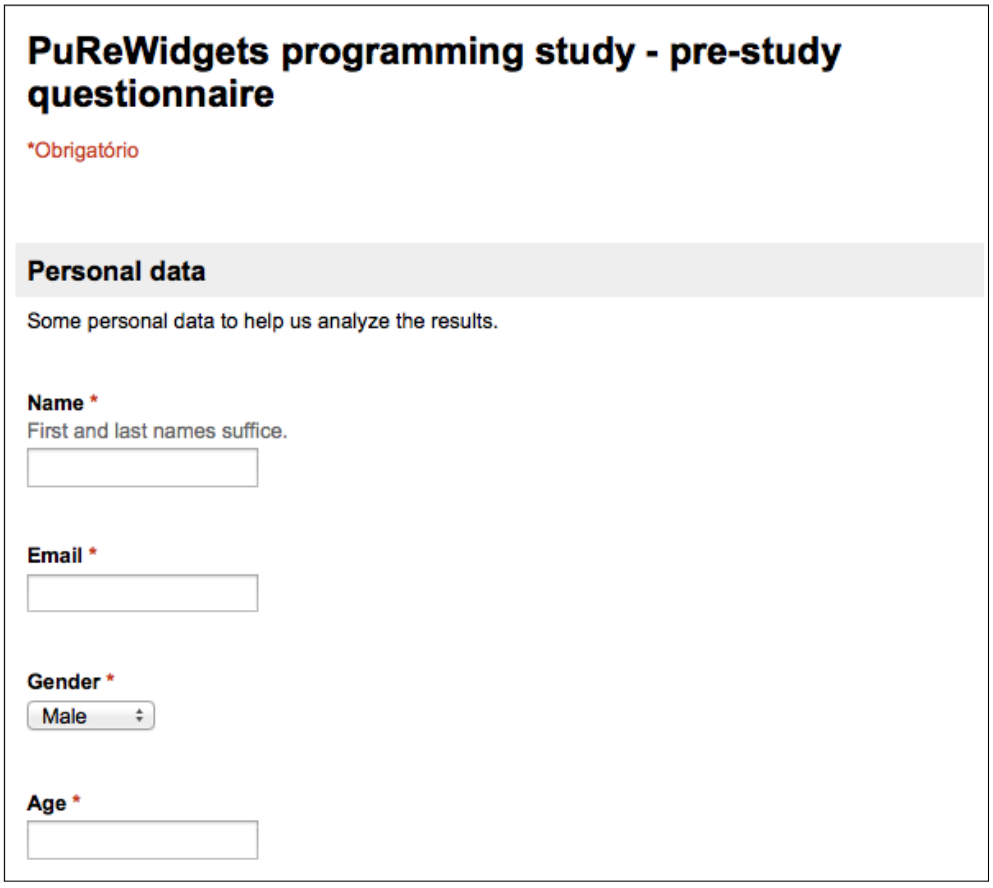
54. WebGlance: Paek, T., Agrawala, M., Basu, S., Drucker, S., Kristjansson, T., Logan, R., Toyama, K., et al. (2004). Toward universal mobile interaction for shared displays. CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work (pp. 266–269). New York, NY, USA: ACM. doi:10.1145/1031607.1031649

55. Webwall: Ferscha, A., Kathan, G., & Vogl, S. (2002). WebWall - An Architecture for Public Display WWW Services. The Eleventh International World Wide Web Conference. Honolulu, Hawaii, USA. Retrieved from http://www2002.org/CDROM/alternate/701/

Chapter B

Questionnaires

B.1 Programming Study: Screening Questionnaire



PuReWidgets programming study - pre-study questionnaire

*Obrigatório

Personal data

Some personal data to help us analyze the results.

Name *
First and last names suffice.

Email *

Gender *
Male

Age *

Programming experience

What programming languages are you experienced with? *
List the ones you are most experienced with.

For how long (years) have you programmed? *

Do you have Web development experience with any of the following:? *
If you don't know what an item means, don't mark it.

- [ ] HTML
- [ ] CSS
- [ ] JavaScript
- [ ] JSON
- [ ] REST

Have you ever used external web APIs to get data from, in your web applications? *
For example, from Facebook, Twitter, Google, ... If you don't have web programming experience, mark "No".

- ( ) Yes
- ( ) No

If you answered "Yes" in the previous question, list some of the latest APIs you remember to have used.

Have you programmed in Google Web Toolkit (GWT) before? *

- ( ) Yes
- ( ) No

List other web development frameworks (server, or client side) you have experience with.
GWT, jQuery, Djando, Ruby on Rails, ASP.NET, ...

Have you ever used Eclipse before? *

- ( ) Yes
- ( ) No

What other IDEs have you used?
List up to three other.

Enviar

Tecnologia do Google Docs

B.2 Programming Study: Final Questionnaire

PuReWidgets post-study questionnaire

*Obrigatório

Concepts

I understood the concept of place as used in PuReWidgets *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

I understood the concept of application as used in PuReWidgets *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

I understood the concept of widget as used in PuReWidgets *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

I understood the concept of input feedback as used in PuReWidgets *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

Continuar »

Tecnologia do Google Docs

Denunciar abuso - Termos de Utilização - Termos adicionais

PuReWidgets post-study questionnaire

*Obrigatório

Tasks

Tasks

I was able to complete satisfactorily the first task (modifying helloworld) *

- ◯ True
- ◯ False

I was able to complete satisfactorily the second task (SlideShow) *

- ◯ True
- ◯ False

I was able to complete satisfactorily the third task (Multi-user) *

- ◯ True
- ◯ False

I was able to complete satisfactorily the fourth task (Enhanced feedback) *

- ◯ True
- ◯ False

I understood how to create widgets *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

I understood how to delete widgets *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

I understood how to modify the input feedback shown on the public display. *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

Any comments or suggestions regarding the tasks or widget/input feedback operations

« Anterior Continuar »

PuReWidgets post-study questionnaire

*Obrigatório

Documentation

During the tasks I used the wiki documentation available at the project's site *

- ◯ Yes
- ◯ No

During the tasks I used the javadoc documentation available at the project's site *

- ◯ Yes
- ◯ No

I think the wiki documentation was clear *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

I think the javadoc documentation was clear *

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

I think the wiki and javadoc documentation was sufficient to complete the tasks. *

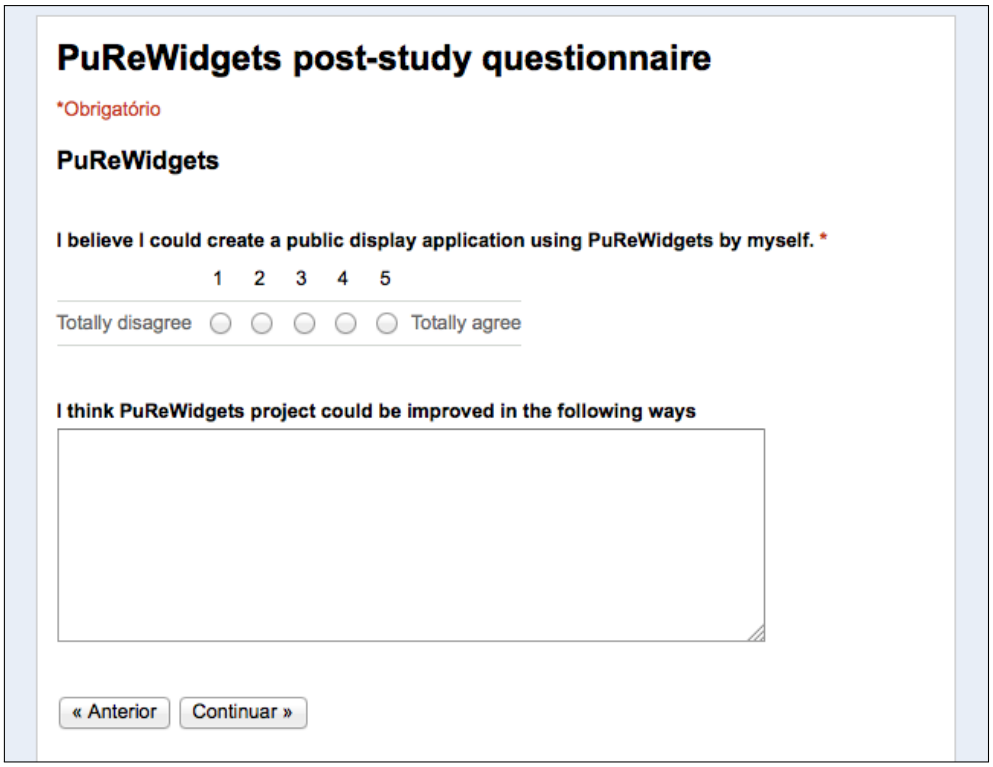
	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

Any comments regarding the documentation

« Anterior Continuar »

Tecnologia do Google Docs

Denunciar abuso - Termos de Utilização - Termos adicionais



PuReWidgets post-study questionnaire

*Obrigatório

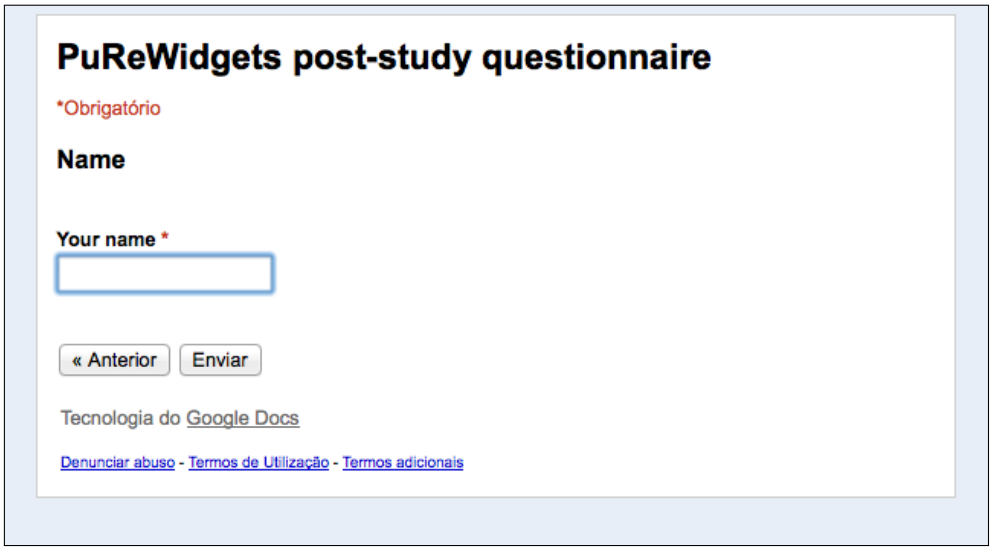
PuReWidgets

I believe I could create a public display application using PuReWidgets by myself. *

	1	2	3	4	5	
Totally disagree	○	○	○	○	○	Totally agree

I think PuReWidgets project could be improved in the following ways

« Anterior Continuar »



PuReWidgets post-study questionnaire

*Obrigatório

Name

Your name *

« Anterior Enviar

Tecnologia do Google Docs

Denunciar abuso - Termos de Utilização - Termos adicionais

References

Alt, F., Kubitza, T., Bial, D., Zaidan, F., Ortel, M., Zurmaar, B., Lewen, T., Shirazi, A. S., and Schmidt, A. Digifieds: Insights into Deploying Digital Public Notice Areas in the Wild. In *Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia - MUM '11*, pages 165–174, New York, New York, USA, December 2011. ACM Press. ISBN 9781450310963. DOI 10.1145/2107596.2107618. URL `http://dl.acm.org/citation.cfm?id=2107596.2107618`.

Alt, F., Shirazi, A. S., Kubitza, T., and Schmidt, A. Interaction techniques for creating and exchanging content with public displays. In *CHI '13 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1709–1718. ACM, April 2013. ISBN 978-1-4503-1899-0. DOI 10.1145/2466110.2466226. URL `http://dl.acm.org/citation.cfm?id=2466110.2466226`.

Amador, G. and Gomes, A. TouchAll: A Multi-Touch, Gestures, and Fiducials API for Flash/Action Script 3.0. In *2011 Fifth FTRA International Conference on Multimedia and Ubiquitous Engineering*, pages 53–58. IEEE, June 2011. ISBN 978-1-4577-1228-9. DOI 10.1109/MUE.2011.21. URL `http://www.computer.org/portal/web/csdl/doi/10.1109/MUE.2011.21`.

Ballagas, R., Ringel, M., Stone, M., and Borchers, J. iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments. In *Proceedings of the conference on Human factors in computing systems - CHI '03*, CHI '03, page 537, New York, New York, USA, 2003. ACM Press. ISBN 1581136307. DOI 10.1145/642611.642705. URL `http://portal.acm.org/citation.cfm?doid=642611.642705`.

Ballagas, R., Rohs, M., and Sheridan, J. G. Sweep and Point & Shoot: Phonecam-Based Interactions for Large Public Displays. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1200–1203, New York, NY, USA, 2005. ACM. ISBN 1-59593-002-7. DOI 10.1145/1056808.1056876.

Ballagas, R., Rohs, M., Sheridan, J. G., and Borchers, J. The Design Space of Ubiquitous Mobile Input. In Lumsden, J., editor, *Handbook of Research on User Interface Design and Evaluation for Mobile Technology*, volume 1, chapter 24, pages 386–407. IGI Global, 2008. URL `http://www.igi-global.com/bookstore/chapter.aspx?TitleId=21843`.

Bass, L. and Coutaz, J. *Developing Software for the User Interface.* Addison Wesley, 1991. ISBN 0201510464.

Beach, A., Gartrell, M., Akkala, S., Elston, J., Kelley, J., Nishimoto, K., Ray, B., Razgulin, S., Sundaresan, K., Surendar, B., Terada, M., and Han, R. WhozThat? evolving an ecosystem for context-aware mobile social networks. *IEEE Network*, 22(4):50–55, July

2008. ISSN 0890-8044. DOI 10.1109/MNET.2008.4579771. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4579771.

Bellotti, V., Back, M., Edwards, W. K., Grinter, R. E., Henderson, A., and Lopes, C. Making sense of sensing systems. In *Proceedings of the SIGCHI conference on Human factors in computing systems Changing our world, changing ourselves - CHI '02*, page 415, New York, New York, USA, 2002. ACM Press. ISBN 1581134533. DOI 10.1145/503376.503450. URL http://portal.acm.org/citation.cfm?doid=503376.503450.

Bellucci, A., Malizia, A., Diaz, P., and Aedo, I. Human-Display Interaction Technology: Emerging Remote Interfaces for Pervasive Display Environments. *IEEE Pervasive Computing*, 9(2):72–76, April 2010. ISSN 1536-1268. DOI 10.1109/MPRV.2010.30. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5437545.

Blanchette, J. The Little Manual of API Design. Technical report, Trolltech, a Nokia company, 2008. URL http://www4.in.tum.de/~blanchet/api-design.pdf.

Bohmer, M. and Muller, J. Users' Opinions on Public Displays that Aim to Increase Social Cohesion. In *2010 Sixth International Conference on Intelligent Environments*, pages 255–258. IEEE, July 2010. ISBN 978-1-4244-7836-1. DOI 10.1109/IE.2010.53. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5673865.

Bollen, J., Pepe, A., and Mao, H. Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena. *CoRR*, abs/0911.1, 2009. URL http://dblp.uni-trier.de/db/journals/corr/corr0911.html#abs-0911-1583.

Bragdon, A., DeLine, R., Hinckley, K., and Morris, M. R. Code space: touch + air gesture hybrid interactions for supporting developer meetings. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS '11*, page 212, New York, New York, USA, November 2011. ACM Press. ISBN 9781450308717. DOI 10.1145/2076354.2076393. URL http://dl.acm.org/citation.cfm?id=2076354.2076393.

Brignull, H. and Rogers, Y. Enticing People to Interact with Large Public Displays in Public Spaces. In Rauterberg, M., Menozzi, M., and Wesson, J., editors, *INTERACT'03*, pages 17–24. IOS Press, 2003. ISBN 1-58603-363-8. URL http://dblp.uni-trier.de/db/conf/interact/interact2003.html#BrignullR03.

Brignull, H., Izadi, S., Fitzpatrick, G., Rogers, Y., and Rodden, T. The introduction of a shared interactive surface into a communal space. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work - CSCW '04*, page 49, New York, New York, USA, 2004. ACM Press. ISBN 1581138105. DOI 10.1145/1031607.1031616. URL http://portal.acm.org/citation.cfm?doid=1031607.1031616.

Cardoso, J. C. S. and Jose, R. A Framework for Context-Aware Adaptation in Public Displays. In Meersman, R., Herrero, P., and Dillon, T., editors, *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, volume 5872/2009 of *Lecture Notes in Computer Science*, pages 118–127, Vilamoura, Portugal, 2009. Springer Berlin / Heidelberg. DOI 10.1007/978-3-642-05290-3_21. URL http://jorgecardoso.eu/publications/2009-cams-digitalfootprintsframework.pdf.

Cardoso, J. C. S. and José, R. Assessing Feedback for Indirect Shared Interaction with Public Displays. In Meersman, R., Dillon, T., and Herrero, P., editors, *On the Move to Meaningful Internet Systems: OTM 2011 Workshops*, volume 7046 of *Lecture Notes*

in Computer Science, pages 553–561. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-25125-2. DOI 10.1007/978-3-642-25126-9_67. URL http://dx.doi.org/10.1007/978-3-642-25126-9_67.

Cardoso, J. C. S. and José, R. Creating web-based interactive public display applications with the PuReWidgets toolkit. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia - MUM '12*, page 1, New York, New York, USA, 2012a. ACM Press. ISBN 9781450318150. DOI 10.1145/2406367.2406434. URL http://dl.acm.org/citation.cfm?doid=2406367.2406434.

Cardoso, J. C. S. and José, R. PuReWidgets: a programming toolkit for interactive public display applications. In José Creissac Campos, Simone D. J. Barbosa, Philippe Palanque, Rick Kazman, Michael Harrison, S. R., editor, *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '12*, page 51, New York, NY, USA, June 2012b. ACM Press. ISBN 9781450311687. DOI 10.1145/2305484.2305496. URL http://dl.acm.org/citation.cfm?doid=2305484.2305496.

Cardoso, J. C. S. and José, R. The PuReWidgets Toolkit for Interactive Public Display Applications. In *The International Symposium on Pervasive Displays*, Porto, June 2012c. URL http://dx.doi.org/10.6084/m9.figshare.92165.

Carter, S., Churchill, E. F., Denoue, L., Helfman, J., and Nelson, L. Digital graffiti: public annotation of multimedia content. In *CHI '04 extended abstracts on Human factors in computing systems*, pages 1207–1210, New York, NY, USA, 2004. ACM. ISBN 1-58113-703-6. DOI http://doi.acm.org/10.1145/985921.986025.

Cheverst, K., Dix, A. J., Fitton, D., Friday, A., and Rouncefield, M. Exploring the Utility of Remote Messaging and Situated Office Door Displays. In *Mobile HCI*, pages 336–341, 2003. URL http://www.springerlink.com/content/g27g1vnedqd2llmp/.

Cheverst, K., Dix, A. J., Fitton, D., Kray, C., Rouncefield, M., Sas, C., Saslis-Lagoudakis, G., and Sheridan, J. G. Exploring bluetooth based mobile phone interaction with the hermes photo display. In *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services - MobileHCI '05*, page 47, New York, New York, USA, 2005. ACM Press. ISBN 1595930892. DOI 10.1145/1085777.1085786. URL http://portal.acm.org/citation.cfm?doid=1085777.1085786.

Churchill, E. F., Nelson, L., and Denoue, L. Multimedia fliers: information sharing with digital community bulletin boards. In M.H. Huysman Etienne Wenger, V. W., editor, *Communities and technologies*, pages 97–117. Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 2003a. ISBN 1-4020-1611-5.

Churchill, E. F., Nelson, L., Denoue, L., and Girgensohn, A. The Plasma Poster Network: Posting Multimedia Content in Public Places. In Rauterberg, M., Menozzi, M., and Wesson, J., editors, *Human-Computer Interaction INTERACT '03*, pages 599–606. IOS Press, 2003b. URL http://www.fxpal.com/publications/FXPAL-PR-03-197.pdf.

Churchill, E. F., Nelson, L., Denoue, L., Helfman, J., and Murphy, P. Sharing multimedia content with interactive public displays. In *Proceedings of the 2004 conference on Designing interactive systems processes, practices, methods, and techniques - DIS '04*, pages 7–16, New York, New York, USA, 2004. ACM Press. ISBN 1581137877. DOI 10.1145/1013115.1013119. URL http://portal.acm.org/citation.cfm?doid=1013115.1013119.

Cooper, A., Reimann, R., and Cronin, D. *About face 3: the essentials of interaction design*. John Wiley & Sons, Inc., New York, NY, USA, 2007. ISBN 9780470084113.

Cox, D., Kindratenko, V., and Pointer, D. IntelliBadge : Towards Providing Location-Aware Value-Added Services at Academic Conferences. In Dey, A., Schmidt, A., and McCarthy, J., editors, *UbiComp 2003: Ubiquitous Computing*, pages 264–280. Springer Berlin / Heidelberg, 2003. DOI 10.1007/978-3-540-39653-6_21. URL http://www.springerlink.com/content/wddnwhf8e9mdb0tl.

Davies, N., Friday, A., Newman, P., Rutlidge, S., and Storz, O. Using bluetooth device names to support interaction in smart environments. In *Proceedings of the 7th international conference on Mobile systems, applications, and services - Mobisys '09*, pages 151–164, New York, New York, USA, 2009. ACM Press. ISBN 9781605585666. DOI 10.1145/1555816.1555832. URL http://portal.acm.org/citation.cfm?doid=1555816.1555832.

Davies, N., Langheinrich, M., Jose, R., and Schmidt, A. Open Display Networks: A Communications Medium for the 21st Century. *Computer*, 45(5):58–64, May 2012. ISSN 0018-9162. DOI 10.1109/MC.2012.114. URL http://www.computer.org/csdl/mags/co/2012/05/mco2012050058-abs.html.

Dearman, D. and Truong, K. N. BlueTone: a framework for interacting with public displays using dual-tone multi-frequency through bluetooth. In *Proceedings of the 11th international conference on Ubiquitous computing - Ubicomp '09*, pages 97–100, New York, New York, USA, 2009. ACM Press. ISBN 9781605584317. DOI 10.1145/1620545.1620561. URL http://portal.acm.org/citation.cfm?doid=1620545.1620561.

Deecker, G. F. P. and Penny, J. P. Standard input forms for interactive computer graphics. *ACM SIGGRAPH Computer Graphics*, 11(1):32–40, April 1977. ISSN 00978930. URL http://dl.acm.org/citation.cfm?id=988655.988659.

Dey, A. K. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, 2000. URL http://www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf.

Dix, A. J. and Sas, C. Public displays and private devices: A design space analysis. In *Workshop on Designing and evaluating mobile phone-based interaction with public displays. CHI2008*, Florence, 2008.

Dix, A. J. and Sas, C. Mobile Personal Devices meet Situated Public Displays : Synergies and Opportunities. *International Journal of Ubiquitous Computing*, 1(1):11–28, 2010. URL http://www.alandix.com/academic/papers/MPD-SPD-2010/.

Fass, A., Forlizzi, J., and Pausch, R. MessyDesk and MessyBoard: two designs inspired by the goal of improving human memory. In *DIS '02: Proceedings of the 4th conference on Designing interactive systems*, pages 303–311, New York, NY, USA, 2002. ACM. ISBN 1-58113-515-7. DOI http://doi.acm.org/10.1145/778712.778754.

Ferscha, A., Kathan, G., and Vogl, S. WebWall - An Architecture for Public Display WWW Services. In *The Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA, 2002. URL http://www2002.org/CDROM/alternate/701/.

Foley, J. D., Chan, P., and Wallace, V. L. The human factors of computer graphics interaction techniques. Technical report, U. S. Army Research Institute for the Behavioral and Social Sciences, 1980. URL http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA136605.

Glaser, B. and Strauss, A. *The Discovery of Grounded Theory*. AldineTransaction, 1967. ISBN 0202302601. URL http://books.google.com/books?id=tSi7KiOHkpYC.

Google. Google App Engine. Webpage: http://code.google.com/appengine/, 2011a. URL http://code.google.com/appengine/.

Google. Google Web Toolkit. Webpage: http://code.google.com/webtoolkit/, 2011b, Accessed June 2010. URL http://code.google.com/webtoolkit/.

Grasso, A., Muehlenbrock, M., Roulland, F., and Snowdon, D. Supporting communities of practice with large screen displays. In O'Hara, K., Perry, E., Churchill, E., and Russel, D. M., editors, *Public and Situated Displays - Social and Interactional Aspects of Shared Display Technologies*, pages 261–282. Kluwer, 2003.

Greenberg, S. Toolkits and interface creativity. *Multimedia Tools and Applications*, 32(2):139–159, 2007. ISSN 1380-7501 (Print) 1573-7721 (Online). DOI 10.1007/s11042-006-0062-y. URL http://www.springerlink.com/content/b5760748p3233316/.

Greenberg, S. and Rounding, M. The Notification Collage: Posting Information to Public and Personal Displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 515–521, Seattle, Washington, United States, 2001. ACM. ISBN 1-58113-327-8. DOI 10.1145/365024.365339.

Hardy, J. and Alexander, J. Toolkit support for interactive projected displays. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia - MUM '12*, page 1, New York, NY, USA, December 2012. ACM Press. ISBN 9781450318150. DOI 10.1145/2406367.2406419. URL http://dl.acm.org/citation.cfm?id=2406367.2406419.

Hardy, R. and Rukzio, E. Touch & interact: touch-based interaction of mobile phones with displays. In ter Hofte, G. H., Mulder, I., and de Ruyter, B. E. R., editors, *Mobile HCI*, ACM International Conference Proceeding Series, pages 245–254. ACM, 2008. ISBN 978-1-59593-952-4. URL http://dblp.uni-trier.de/db/conf/mhci/mhci2008.html#HardyR08.

Heer, J., Card, S. K., and Landay, J. A. prefuse: A Toolkit for Interactive Information Visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05*, page 421, New York, New York, USA, April 2005. ACM Press. ISBN 1581139985. DOI 10.1145/1054972.1055031. URL http://dl.acm.org/citation.cfm?id=1054972.1055031.

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

Huang, E. M., Russell, D. M., and Sue, A. E. IM here: public instant messaging on large, shared displays for workgroup interactions. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 279–286, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. DOI 10.1145/985692.985728.

Huang, E. M., Koster, A., and Borchers, J. Overcoming Assumptions and Uncovering Practices: When Does the Public Really Look at Public Displays? In *Pervasive*, pages 228–243, 2008.

Janrain. Janrain - user management platform for the social web, 2013. URL http://www.janrain.com/.

Jansen, M., Uzun, I., Hoppe, U., and Rossmanith, P. Integrating Heterogeneous Personal Devices with Public Display-Based Information Services. In *IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE'05)*, pages 149–153. IEEE, 2005. ISBN 0-7695-2385-4. DOI 10.1109/WMTE.2005.37. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1579254.

Johanson, B. and Fox, A. The Event Heap: a coordination infrastructure for interactive workspaces. In *Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on*, pages 83–93, 2002. DOI 10.1109/MCSA.2002.1017488.

Jose, R. and Cardoso, J. C. S. Opportunities and Challenges of Interactive Public Displays as an Advertising Medium. In Mueller, J., Alt, F., and Michelis, D. E., editors, *Pervasive Advertising*, Human-Computer Interaction Series, chapter 3, pages 139–157. Springer-Verlag London Limited, 2011. ISBN 978-0-85729-351-0. DOI 10.1007/978-0-85729-352-7_7. URL http://dx.doi.org/10.1007/978-0-85729-352-7_7.

José, R., Otero, N., Izadi, S., and Harper, R. Instant Places: Using Bluetooth for Situated Interaction in Public Displays. *IEEE Pervasive Computing*, 7(4):52–57, October 2008. ISSN 1536-1268. DOI 10.1109/MPRV.2008.74. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4653472.

José, R., Pinto, H., Silva, B., Melro, A., and Rodrigues, H. Beyond interaction: Tools and practices for situated publication in display networks. In *Proceedings of the 2012 International Symposium on Pervasive Displays - PerDis '12*, pages 1–6, New York, New York, USA, June 2012. ACM Press. URL http://dl.acm.org/citation.cfm?id=2307798.2307806.

José, R., Cardoso, J., Alt, F., Clinch, S., and Davies, N. Mobile applications for open display networks: common design considerations. In *Proceedings of the 2nd ACM International Symposium on Pervasive Displays – PerDis '13*, pages 97–102. ACM, June 2013. ISBN 978-1-4503-2096-2. DOI 10.1145/2491568.2491590. URL http://dl.acm.org/citation.cfm?id=2491568.2491590.

JQuery. jQuery. Webpage, 2013. URL http://jquery.com/.

Ju, W., Lee, B. A., and Klemmer, S. R. Range: Exploring Implicit Interaction through Electronic Whiteboard Design. In *Proceedings of the ACM 2008 conference on Computer supported cooperative work - CSCW '08*, page 17, New York, New York, USA, 2008. ACM Press. ISBN 9781605580074. DOI 10.1145/1460563.1460569. URL http://portal.acm.org/citation.cfm?doid=1460563.1460569.

Kaviani, N., Finke, M., Fels, S., Lea, R., and Wang, H. What goes where?: designing interactive large public display applications for mobile device interaction. In *Proceedings of the First international Conference on internet Multimedia Computing and Service*, pages 129–138. ACM, 2009. ISBN 9781605588407. DOI 10.1145/1734605.1734637. URL http://portal.acm.org/citation.cfm?id=1734637.

Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. Papier-Mâché: Toolkit Support for Tangible Input. In *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*, pages 399–406, New York, New York, USA, April 2004. ACM Press. ISBN 1581137028. DOI 10.1145/985692.985743. URL http://dl.acm.org/citation.cfm?id=985692.985743.

Kostakos, V. and O'Neill, E. Cityware: Urban Computing to Bridge Online and Real-world Social Networks. In Foth, M., editor, *Handbook of Research on Urban Informatics: The Practice and Promise of the Real-Time City*, chapter XIII, pages 195–204. Information Science Reference, IGI Global, 2008a.

Kostakos, V. and O'Neill, E. Capturing and visualising Bluetooth encounters. In *adjunct proceedings of the conference on Human factors in computing systems (CHI 2008)*, Florence, Italy, 2008b.

Kubitza, T., Clinch, S., Davies, N., and Langheinrich, M. Using mobile devices to personalize pervasive displays. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(4):26, February 2013. ISSN 15591662. DOI 10.1145/2436196.2436211. URL `http://dl.acm.org/citation.cfm?id=2436196.2436211`.

Kwon, Y. Age Classification from Facial Images. *Computer Vision and Image Understanding*, 74(1):1–21, April 1999. ISSN 10773142. DOI 10.1006/cviu.1997.0549. URL `http://linkinghub.elsevier.com/retrieve/pii/S107731429790549X`.

Li, Y., Groenegress, C., Strauss, W., and Fleischmann, M. Gesture Frame – A Screen Navigation System for Interactive Multimedia Kiosks. *Computer*, pages 380–385, 2004. DOI 10.1007/978-3-540-24598-8_35.

LocaModa. LocaModa App Store. Webpage, 2010, Accessed September 2010. URL `http://locamoda.com/apps/`.

Mackinlay, J. D., Card, S. K., and Robertson, G. G. A Semantic Analysis of the Design Space of Input Devices. *Human-Computer Interaction*, 5(2&3):145–190, 1990.

Martin, K., Penn, A., and Gavin, L. Engaging with a situated display via picture messaging. In *CHI '06 extended abstracts on Human factors in computing systems - CHI '06*, page 1079, New York, New York, USA, 2006. ACM Press. ISBN 1-59593-298-4. DOI 10.1145/1125451.1125656. URL `http://portal.acm.org/citation.cfm?doid=1125451.1125656`.

Maunder, A., Marsden, G., and Harper, R. Creating and sharing multi-media packages using large situated public displays and mobile phones. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services - MobileHCI '07*, pages 222–225, New York, New York, USA, September 2007. ACM Press. ISBN 9781595938626. DOI 10.1145/1377999.1378010. URL `http://portal.acm.org/citation.cfm?doid=1377999.1378010`.

McCarthy, J. F. Using Public Displays to Create Conversation Opportunities. In *CSCW 2002 Workshop on Public, Community and Situated Displays*, New Orleans, 2002.

McCarthy, J. F., Costa, T. J., and Liongosari, E. S. UniCast, OutCast & GroupCast: Three Steps Toward Ubiquitous, Peripheral Displays. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 332–345, London, UK, 2001. Springer-Verlag. ISBN 3-540-42614-0.

McCarthy, J. F., Farnham, S. D., Patel, Y., Ahuja, S., Norman, D., Hazlewood, W. R., and Lind, J. Supporting community in third places with situated social software. In *Proceedings of the fourth international conference on Communities and technologies - C&T '09*, pages 225–234, New York, New York, USA, 2009. ACM Press. ISBN 9781605587134. DOI 10.1145/1556460.1556493. URL `http://portal.acm.org/citation.cfm?doid=1556460.1556493`.

McCormack, J. and Asente, P. An overview of the X toolkit. In *Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software - UIST '88*, pages 46–55, New York, New York, USA, 1988. ACM Press. ISBN 0897912837. DOI 10.1145/62402.62407. URL http://portal.acm.org/citation.cfm?doid=62402.62407.

McDonald, D. W., McCarthy, J. F., Soroczak, S., Nguyen, D. H., and Rashid, A. M. Proactive displays: Supporting awareness in fluid social environments. *ACM Transactions on Computer-Human Interaction*, 14(4):1–31, January 2008. ISSN 10730516. DOI 10.1145/1314683.1314684. URL http://portal.acm.org/citation.cfm?doid=1314683.1314684.

Michelis, D. and Müller, J. The Audience Funnel: Observations of Gesture Based Interaction With Multiple Large Displays in a City Center. *International Journal of Human-Computer Interaction*, 27(6):562–579, June 2011. ISSN 1044-7318. DOI 10.1080/10447318.2011.555299. URL http://dx.doi.org/10.1080/10447318.2011.555299.

Miraglia, E. The Yahoo! User Interface Library. Blog post, 2006. URL http://www.yuiblog.com/blog/2006/02/13/the-yahoo-user-interface-library/.

MoDAL. MoDAL (Mobile Document Application Language). Webpage, 2011, Accessed January 2011. URL http://www.almaden.ibm.com/cs/TSpaces/MoDAL/.

Montiel-Hernandez, J. and Cuayahuitl, H. SUIML: A Markup Language for Facilitating Automatic Speech Application Development. In Sheremetov, L. and Alvarado, M., editors, *Workshop on Intelligent Computing*, pages pp. 376–387, 2004. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.9666.

Müller, J. and Krüger, A. MobiDiC: Context Adaptive Digital Signage with Coupons. In Tscheligi, M., de Ruyter, B., Markopoulus, P., Wichert, R., Mirlacher, T., Meschterjakov, A., and Reitberger, W., editors, *Ambient Intelligence*, volume 5859 of *Lecture Notes in Computer Science*, pages 24–33. Springer Berlin / Heidelberg, 2009. DOI 10.1007/978-3-642-05408-2_3. URL http://dx.doi.org/10.1007/978-3-642-05408-2_3.

Müller, J., Paczkowski, O., and Krüger, A. Situated Public News and Reminder Displays. In Schiele, B., Dey, A., Gellersen, H., de Ruyter, B., Tscheligi, M., Wichert, R., Aarts, E., and Buchmann, A., editors, *Ambient Intelligence*, volume 4794 of *Lecture Notes in Computer Science*, pages 248–265. Springer Berlin / Heidelberg, 2007. DOI 10.1007/978-3-540-76652-0_15. URL http://dx.doi.org/10.1007/978-3-540-76652-0_15.

Müller, J., Walter, R., Bailly, G., Nischt, M., and Alt, F. Looking glass: a field study on noticing interactivity of a shop window. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*, page 297, New York, New York, USA, May 2012. ACM Press. ISBN 9781450310154. DOI 10.1145/2207676.2207718. URL http://dl.acm.org/citation.cfm?id=2207676.2207718.

Myers, B. User-interface tools: introduction and survey. *IEEE Software*, 6(1):15–23, 1989. ISSN 07407459. DOI 10.1109/52.16898. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=16898.

Myers, B. A. A new model for handling input. *ACM Trans. Inf. Syst.*, 8(3):289–320, 1990. ISSN 1046-8188. DOI http://doi.acm.org/10.1145/98188.98204.

Myers, B. A., Stiel, H., and Gargiulo, R. Collaboration using multiple PDAs connected to a PC. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported*

cooperative work, pages 285–294, New York, NY, USA, 1998. ACM. ISBN 1-58113-009-0. DOI 10.1145/289444.289503. URL http://doi.acm.org/10.1145/289444.289503.

NEC. NEC Launches Eye Flavor, Japan's First All-in-One Digital Signage Board with Face Recognition Technology. Webpage, 2009, Accessed November 2010. URL http://www.bloomberg.com/apps/news?pid=newsarchive&sid=azUYhTnA4rXk.

Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., and Pignol, M. Generating remote control interfaces for complex appliances. In *Proceedings of the 15th annual ACM symposium on User interface software and technology - UIST '02*, UIST '02, page 161, New York, New York, USA, 2002. ACM Press. ISBN 1581134886. DOI 10.1145/571985.572008. URL http://portal.acm.org/citation.cfm?doid=571985.572008.

Norman, D. A. *The Design of Everyday Things*. Basic Books, 2002.

O'Hara, K., Lipson, M., Jansen, M., Unger, A., Jeffries, H., and Macer, P. Jukola: Democratic Music Choice in a Public Space. In *Proceedings of the 2004 conference on Designing interactive systems processes, practices, methods, and techniques - DIS '04*, page 145, New York, New York, USA, 2004. ACM Press. ISBN 1581137877. DOI 10.1145/1013115.1013136. URL http://portal.acm.org/citation.cfm?doid=1013115.1013136.

Ohlson, M. System Design Considerations for Graphics Input Devices. *Computer*, 11(11): 9–18, November 1978. ISSN 0018-9162. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=1646749&contentType=Journals+&+Magazines.

Olsen, D. R., Jefferies, S., Nielsen, T., Moyes, W., and Fredrickson, P. Cross-modal interaction using XWeb. *Proceedings of the 13th annual ACM symposium on User interface software and technology - UIST '00*, 2:191–200, 2000. DOI 10.1145/354401.354764. URL http://portal.acm.org/citation.cfm?doid=354401.354764.

Paek, T., Agrawala, M., Basu, S., Drucker, S., Kristjansson, T., Logan, R., Toyama, K., and Wilson, A. Toward universal mobile interaction for shared displays. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 266–269, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. DOI 10.1145/1031607.1031649. URL http://doi.acm.org/10.1145/1031607.1031649.

Peltonen, P., Kurvinen, E., Salovaara, A., Jacucci, G., Ilmonen, T., Evans, J., Oulasvirta, A., and Saarikko, P. It's Mine, Don't Touch!: interactions at a large multi-touch display in a city centre. In *CHI 08 Proceeding of the twentysixth annual SIGCHI conference on Human factors in computing systems*, volume 16 of *April 05-10*, pages 1285–1294. ACM, ACM, 2008. ISBN 9781605580111. DOI 10.1145/1357054.1357255. URL http://portal.acm.org/citation.cfm?id=1357255.

Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., and Winograd, T. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Proceedings of the 3rd international conference on Ubiquitous Computing*, UbiComp '01, pages 56–75, London, UK, 2001. Springer-Verlag. ISBN 3-540-42614-0. URL http://portal.acm.org/citation.cfm?id=647987.741344.

Prante, T., Röcker, C., Streitz, N., Stenzel, R., Magerkurth, C., van Alphen, D., and Plewe, D. Hello.Wall - Beyond Ambient Displays. In *Video Track and Adjunct Proceedings of the 5th Intern. Conference on Ubiquitous Computing (UBICOMP'03)*, Seattle, Wash., USA, 2003.

Quividi. Quividi - Automated Audience Measurement of Billboards and Out Of Home Digital Media. Webpage, 2013, Accessed April 2009. URL http://www.quividi.com/.

Raj, H., Gossweiler, R., and Milojicic, D. Contentcascade incremental content exchange between public displays and personal devices. *Mobile and Ubiquitous Systems, Annual International Conference on*, 0:374–381, 2004. DOI 10.1109/MOBIQ.2004.1331744. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1331744.

Rattenbury, T. and Naaman, M. Methods for extracting place semantics from Flickr tags. *ACM Transactions on the Web*, 3(1):1–30, January 2009. ISSN 15591131. DOI 10.1145/1462148.1462149. URL http://portal.acm.org/citation.cfm?doid=1462148.1462149.

Ribeiro, F. and José, R. Autonomous and Context-Aware Scheduling for Public Displays Using Place-Based Tag Clouds. In Augusto, J., Corchado, J., Novais, P., and Analide, C., editors, *Ambient Intelligence and Future Trends-International Symposium on Ambient Intelligence (ISAmI 2010)*, volume 72 of *Advances in Soft Computing*, pages 131–138. Springer Berlin / Heidelberg, 2010. DOI 10.1007/978-3-642-13268-1_16. URL http://dx.doi.org/10.1007/978-3-642-13268-1_16.

Rogers, Y. and Brignull, H. Subtle ice-breaking: encouraging socializing and interaction around a large public display. In *CSCW'02 Workshop Proceedings*, 2002.

Rohs, M. Visual Code Widgets for Marker-Based Interaction. In *25th IEEE International Conference on Distributed Computing Systems Workshops*, pages 506–513, Washington, DC, USA, 2005. IEEE. ISBN 0-7695-2328-5. DOI 10.1109/ICDCSW.2005.140. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1437218.

Roman, M., Beck, J., and Gefflaut, A. A device-independent representation for services. *Proceedings Third IEEE Workshop on Mobile Computing Systems and Applications*, pages 73–82, 2000. DOI 10.1109/MCSA.2000.895383. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=895383.

Russell, D. M. and Gossweiler, R. On the Design of Personal & Communal Large Information Scale Appliances. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 354–361, London, UK, 2001. Springer-Verlag. ISBN 3-540-42614-0.

Salber, D., Dey, A. K., and Abowd, G. D. The context toolkit. In *Proceedings of the SIGCHI conference on Human factors in computing systems the CHI is the limit - CHI '99*, pages 434–441, New York, New York, USA, 1999. ACM Press. ISBN 0201485591. DOI 10.1145/302979.303126. URL http://portal.acm.org/citation.cfm?doid=302979.303126.

Sawhney, N., Wheeler, S., and Schmandt, C. Aware Community Portals: Shared Information Appliances for Transitional Spaces. *Personal and Ubiquitous Computing*, 5(1):66–70, February 2001. ISSN 1617-4909. DOI 10.1007/s007790170034. URL http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s007790170034.

Scheible, J. and Ojala, T. MobiLenin combining a multi-track music video, personal mobile phones and a public display into multi-user interactive entertainment. In *Proceedings of the 13th annual ACM international conference on Multimedia - MULTIMEDIA '05*, page 199, New York, New York, USA, 2005. ACM Press. ISBN 1595930442. DOI 10.1145/1101149.1101178. URL http://portal.acm.org/citation.cfm?doid=1101149.1101178.

Schmidt, D., Chehimi, F., Rukzio, E., and Gellersen, H. PhoneTouch: a technique for direct phone interaction on surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology - UIST '10*, page 13, New York, New York, USA, October 2010. ACM Press. ISBN 9781450302715. DOI 10.1145/1866029.1866034. URL `http://dl.acm.org/citation.cfm?id=1866029.1866034`.

Sharifi, M., Payne, T., and David, E. Public Display Advertising Based on Bluetooth Device Presence. *Mobile Interaction with the Real World (MIRW 2006) in conjunction with the 8th International Conference on Human Computer Interaction with Mobile Devices and Services*, 2006. URL `http://www.hcilab.org/events/mirw2006/pdf/mirw2006_sharifi.pdf`.

Shneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8):57–69, August 1983. ISSN 0018-9162. DOI 10.1109/MC.1983.1654471. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1654471`.

Spaeth, J., Singer, S., and Hordeychuk, M. Audience Metrics Guidelines. Webpage, 2008, Accessed November 2010. URL `http://www.dp-aa.org/media/DPAA`.

Streitz, N., Prante, T., Röcker, C., Alphen, D. V., Magerkurth, C., Stenzel, R., and Plewe, D. Ambient Displays and Mobile Devices for the Creation of Social Architectural Spaces: Supporting informal communication and social awareness in organizations. In O'Hara, K., Perry, M., Churchill, E., and Russell, D., editors, *Public and Situated Displays: Social and Interactional Aspects of Shared Display Technologies*, chapter 16, pages 387–409. Kluwer Publishers, 2003.

Sumi, Y. and Mase, K. AgentSalon: facilitating face-to-face knowledge exchange through conversations among personal agents. In *Proceedings of the fifth international conference on Autonomous agents - AGENTS '01*, pages 393–400, New York, New York, USA, May 2001. ACM Press. ISBN 158113326X. DOI 10.1145/375735.376344. URL `http://portal.acm.org/citation.cfm?id=375735.376344`.

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

Terrenghi, L., Quigley, A., and Dix, A. J. A taxonomy for and analysis of multi-person-display ecosystems. *Personal and Ubiquitous Computing*, 13(8):583–598, June 2009. ISSN 1617-4909. DOI 10.1007/s00779-009-0244-5. URL `http://portal.acm.org/citation.cfm?id=1644246.1644265`.

Toye, E., Sharp, R., Madhavapeddy, A., and Scott, D. Using smart phones to access site-specific services. *IEEE Pervasive Computing*, 4(2):60–66, 2005. ISSN 15361268. DOI 10.1109/MPRV.2005.44. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1427650`.

Toye, E., Madhavapeddy, A., Sharp, R., Scott, D., Blackwell, A., and Upton, E. Using camera-phones to interact with context-aware mobile services. Technical report, University of Cambridge, Computer Laboratory, Cambridge, 2004.

TruMedia. TrueMedia: iTALLY: Opportunity to See (OTS) People Counter. Webpage, 2013, Accessed April 2009. URL `http://www.trumedia.co.il`.

Vanderheiden, G. C. and Zimmermann, G. Use of User Interface Sockets to Create Naturally Evolving Intelligent Environments. In *11th International Conference on Human-Computer Interaction*, Las Vegas, Nevada USA, 2005.

Venners, B. The Jini ServiceUI API Specification. Webpage, 2005, Accessed January 2011. URL http://www.artima.com/jini/serviceui/Spec.html.

Ventura, P., Sousa, H., and Jorge, J. Mobile Phone Interaction with Outdoor Advertisements. In *Workshop on Designing and evaluating mobile phone-based interaction with public displays. CHI2008*, Florence, 2008.

Verschae, R., Ruiz-del Solar, J., and Correa, M. A unified learning framework for object detection and classification using nested cascades of boosted classifiers. *Machine Vision and Applications*, 19(2):85–103, October 2007. ISSN 0932-8092. DOI 10.1007/s00138-007-0084-0. URL http://www.springerlink.com/index/10.1007/s00138-007-0084-0.

Vogel, D. and Balakrishnan, R. Interactive Public Ambient Displays: Transitioning from Implicit to Explicit, Public to Personal, Interaction with Multiple Users. In *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04*, pages 137–146, New York, New York, USA, 2004. ACM Press. ISBN 1581139578. DOI 10.1145/1029632.1029656. URL http://portal.acm.org/citation.cfm?doid=1029632.1029656.

Vogl, S. *Coordination of Users and Services via Wall Interfaces*. PhD thesis, University of Linz, Linz, Austria, 2002.

VoiceXML. VoiceXML Forum. Webpage, (year not available), Accessed January 2011. URL http://www.voicexml.org/.

von Ahn, L. and Dabbish, L. Designing games with a purpose. *Communications of the ACM*, 51(8):57, August 2008. ISSN 00010782. DOI 10.1145/1378704.1378719. URL http://portal.acm.org/citation.cfm?doid=1378704.1378719.

Wang, M., Boring, S., and Greenberg, S. Proxemic peddler. In *Proceedings of the 2012 International Symposium on Pervasive Displays - PerDis '12*, pages 1–6, New York, New York, USA, June 2012. ACM Press. ISBN 9781450314145. DOI 10.1145/2307798.2307801. URL http://dl.acm.org/citation.cfm?id=2307798.2307801.

Weiser, M. The Computer for the 21st Century. *Scientific American Special Issue on Communications, Computers, and Networks*, 1991. URL http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html.

Yahoo! YUI Library. Webpage, 2013, Accessed January 2011. URL http://developer.yahoo.com/yui.

Ydreams. Vodafone Cube. Webpage, 2003. URL http://www.ydreams.com/#/en/projects/publicurbanexperiences/giantinteractivebillboardsvodafone/.