iScreen: A Toolkit for Interactive Screens

Marcus Handte, Stephan Wagner, Wolfgang Apolinarski, Pedro Jose Marron
Networked Embedded Systems Group
University of Duisburg-Essen, Germany
{marcus.handte|stephan.j.wagner|wolfgang.apolinarski|pjmarron}@uni-due.de

ABSTRACT

With the advent of touch-centric operating systems such as iOS and Android, an ever-increasing number of mobile devices are equipped with touch screens. Several integrated computer systems extend this trend to traditional computers and they provide a cost-efficient hardware platform for a pervasive deployment of interactive displays in a variety of environments. However, to be useful for a particular application scenario, their software needs to be customized. Without further support, this customization can be a time-consuming and costly undertaking which may ultimately limit their applicability. To mitigate this, we have developed the iScreen software toolkit that aims at minimizing the development effort by providing a set of reusable building blocks for interactive applications. In this paper, we present the toolkit's architecture and we describe a number of components that we have implemented on top of it. To validate our work, we present four example applications some of which we have been using on a day-to-day basis over the last two years.

Categories and Subject Descriptors

H.4.2 [**Information Systems Applications**]: Communications Applications—*Bulletin Boards*; D.2.13 [**Software Engineering**]: Reusable Software—*Reusable libraries*

General Terms

Design, Experimentation, Management

Keywords

Pervasive Computing, Interactive Displays, Prototyping

1. INTRODUCTION

With the advent of touch-centric operating systems such as iOS and Android, an ever-increasing number of mobile devices are equipped with screens that enable touch-based interaction. Lately, the enormous success of these systems has not only changed mobile computing but it is increasingly making its way back into traditional computing as well.

Integrated computer systems such as the Asus EEE Top series or the HP Touchsmart series provide a cost-efficient hardware platform for a pervasive deployment of interactive displays in a variety of environments. Besides from providing touch screens, they are also equipped with processing and storage capabilities as well as wireless networking which simplifies deployment and maintenance. Furthermore, using additional accessories, they can be attached to a wall easily.

Yet, to be useful for a particular application scenario, their software needs to be customized. For a bulletin board application, for example, the customization might entail the design of a simple user interface to access the relevant information as well as the development of the gluing code that retrieves the information from a web page or a database. Furthermore, in cases where the computer is mounted in a public area, it might entail the development of additional software to limit the device capabilities.

Without further support, the development of application-specific customizations can be a time-consuming and costly undertaking. In fact, in cases where only few computer systems are to be deployed, the cost resulting from application development can often exceed the hardware and deployment cost. Consequently, in order to increase the feasibility of using touch-enabled computer systems as a low-cost application development platform, it is necessary to reduce the associated software development effort.

The iScreen software toolkit aims at minimizing the development effort by providing a set of reusable building blocks for touch-based, interactive applications. In this paper, we present the overall architecture of the toolkit and we describe a number of components that we have implemented on top of it. To validate our work, we present four example applications some of which we have been using on a daily basis over the last two years.

The remainder of this paper is structured as follows. Next, we introduce the design goals for the toolkit. Thereafter, in Section 3, we describe its architecture and implementation. To evaluate the toolkit, we discuss a number of example applications in Section 4. Finally, we discuss related work in Section 5 and we conclude the paper in Section 6.

2. DESIGN GOALS

The design goals for the iScreen toolkit can be derived directly from the goal of minimizing the effort for the development of arbitrary interactive applications running on

PerDis 2012, June 4–5 2012, Porto, Portugal.

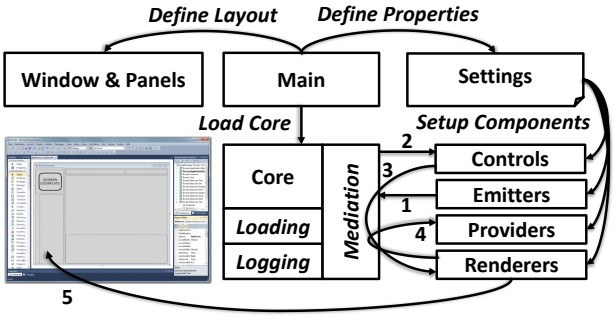


Figure 1: iScreen Architecture

integrated touch-enabled computer systems. In summary, we derive the following three goals:

- Genericity: The toolkit should enable the development of arbitrary applications. Thereby, it should not restrict the application designer, e.g. by mandating a particular screen layout or interaction sequence. Instead, the toolkit should simplify the development of custom layouts as well as interaction sequences.
- Reusability: To ease development, the tookit should support the development of reusable building blocks for common application functionality. Thereby, the reuse should be enabled in such a way that it does not require additional programming effort.
- Extensibility: To support additional application domains that may be emerging over time, the toolkit should be extensible. For this, it must be easy to implement additional functionality in such a way that it can then be integrated into different applications.

3. ISCREEN TOOLKIT

In the following, we first provide an overview of the iScreen toolkit's architecture. Thereafter, we discuss how the individual components interact with each other at runtime. Finally, we outline some implementation issues and we describe a number of generic application building blocks that we implemented on top of it.

3.1 Architecture

The iScreen toolkit follows a component-based approach to application development to achieve it's design goals with respect to reusability and extensiblity. Thereby, components are providing reusable functionality that is orchestrated during application development. As depicted in Figure 1, a generic core is responsible for loading components and mediating their interactions. The core itself is loaded and initialized through a main class which defines the visual layout of the application as well as basic application settings. As hinted in Figure 1, the development of such a main class can be simplified by using existing development tools such as Visual Studio's Form and Settings Designer.

In order to achieve generic composability, application components are loosely coupled and they interact with each other by emitting and consuming events. Similar to Intents[1] in the Android platform, events in iScreen can either target a particular component (in a unicast fashion) – that may be bound at runtime – or they can be announced to all components (in a broadcast like fashion).

Components in iScreen may implement arbitrary application functionality at a developer-defined level of granularity. However, in order to simplify reuse, it is preferable to rather implement fine-granular components. Depending on the functionality implemented by a component, the toolkit provides four separate extension points. Each is defined by means of a component interface as well as a core interface. The component interface defines the set of methods that must be implemented by a component. The core interface exposes a specific subset of the core functionality. To use it, the core passes a handle implementing the core interface to the component during load time. Similar to the model-view-controller design pattern [4], the iScreen toolkit realizes three extension points called renderer (i.e. view), provider (i.e. model), control (i.e. controller). In addition, a forth extension point – the so-called emitter – enables the reception of sensory inputs. For simplicity, a single component may implement multiple extension points. In the following, we briefly outline the idea behind each extension point:

- Renderer: A renderer component is capable of managing a particular type of user interface control on the screen. If the renderer should be reusable in different applications, the underlying model must not be hard coded but instead it must be retrieved dynamically through a provider component. As an example, consider a renderer that is able of drawing and managing a toolbar. Instead of knowing the icons that shall be displayed, it should rather be able to support arbitrary sets of icons in order to be customizable.
- Provider: A provider component is capable of providing a particular model to other components. Thereby, it abstracts from the way how that model is generated. A simple provider may just provide access to a hard coded model. More complex providers may load them from a web page or a file. To allow the reuse of provider components, they may provide access to multiple models. To address a particular model, the iScreen toolkit uses URIs. To support late binding, providers register the supported URIs or URI patterns during load time.
- Emitter: An emitter generates events based on some input. Depending on the type of emitter, the input can either represent some touch event to a visual element generated by a renderer or it can represent some sensor input such as a picture taken by a camera.
- Control: A control component takes care of interpreting the events that are generated by the other components and executing appropriate actions. These actions may entail the activation of new renderers, the removal of existing renderers, the reloading or manipulation of models, etc. Control components can be organized hierarchically, such that a master control component activates its children depending on the state of the application. However, in contrast to other types of components, controls are often application specific and thus, they are hard to reuse.

[1]See http://dev.android.com/guide/topics/intents/intents-filters.html for more details.

3.2 Interaction

To clarify the interaction in an application, we briefly outline a typical scenario depicted in Figure 1. When the main application starts up, it first initializes the core. Thereafter, it registers the application settings as well as the panels contained in the applications main form. When the core is loading, it first initializes the log. Thereafter, it searches for components by scanning a predefined folder for assemblies that exhibit the associated extension points. During component startup, the core issues typed handles to all components which allow them to send and receive events or to initiate other calls depending on the implemented extension points. During component start up, the components may access the application settings in order to retrieve their configuration. Examples for settings may be the resource identifier of the device's webcam or the user name and password to be able to send emails.

Once the core has loaded all components, it signals the successful initialization by sending an event. In response, the control component(s) will typically assign a renderer to different layout panels defined by the main application or they may start different emitters, e.g. in order to capture pictures periodically using the device's camera. If an emitter detects an input, such as a person standing in front of the screen or a person pressing a visual element displayed by a renderer, it generates a corresponding event (c.f. Figure 1 (1)). This event is then received and processed by a control (2). If the screen state needs to be modified, the control will call the core to update a particular renderer on a panel with an associated model provided by a provider (3). Once the renderer receives the request, it fetches the model using the specified provider (4) and updates the panel accordingly (5).

3.3 Implementation

To validate the architecture, we implemented it using the .NET framework. The reason for choosing .NET was threefold. First, many integrated computer systems are shipped with Windows which makes .NET a natural choice. Second, using .NET it is simple to access all of the device's peripherals and due to pinvoke it is straight-forward to access libraries written in other non-MSIL languages. Last but not least, since .NET supports dynamic loading of assemblies, it is possible to distribute components in binary format. This not only speeds up compilation but also simplifies the distribution of proprietary code. On top of the core, we developed a number of components. Among others, they include:

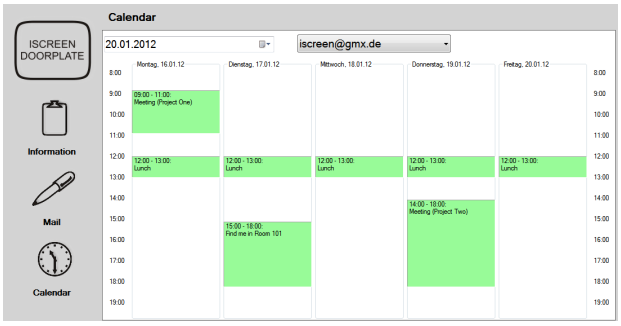
- TouchKeyboard: The TouchKeyboard component implements a customizable renderer and emitter that enables a user to provide text input by typing on the screen. By passing button sequences via a model, the keyboard's keys and layout can be adjusted, for example, to support text or numbers.
- ImageButtonGrid: The ImageButtonGrid component implements a renderer and emitter that shows an arbitrary number of buttons that are organized in a grid that adjusts to its panel size. By pressing a button, the grid can emit a customizable event.
- RTFViewer: The RTFViewer implements a simple renderer that can display a document in rich text format. This can be used, for example, to display instructions on how to use the application.
- HTMLBrowser: The HTMLBrowser wraps Internet Explorer as a renderer and emitter. It can be customized to show only a single web page or to allow browsing. If browsing mode is enabled, it shows an optional bar to specify addresses. A developer can limit the addresses by providing a URL pattern.
- CalendarViewer: The CalendarViewer is an renderer that enables a person to browse a particular Google calendar via Google's Data API. The calendar can be customized to limit the visibility to public events to protect the owner's privacy.
- EmailComposer: The EmailComposer implements a renderer and emitter that enables a person to send an email to a particular (set of) address(es). Using a keyboard input a user can type arbitrary text that is sent once the user hits a send button.
- BluetoothScan: The BluetoothScan component implements an emitter that performs Bluetooth scans. This can be used, for example, to determine the number of persons that are located close to the device.
- CameraSnapshot: The CameraSnapshot component implements an emitter that can take pictures using the device's built-in camera. The snapshots can then either be stored or processed using further components.
- FaceDetector: The FaceDetector component implements a control and emitter that consumes the events from the CameraSnapshot component to detect faces in the camera images. To do that, it uses a simple neural network that is applied to different parts of the image at different resolutions. The output of the component is a set of rectangles which identify the regions with faces.
- FolderTransfer: FolderTransfer is a simple control and emitter component that can transfer a folder from and to the device using Window's built-in file sharing. This allows the transfer of log information to another computer and it enables the retrieval of new content for the device.
- File-/DB-/WebProvider: This set of provider components enable the retrieval of data from different sources. To ease their reuse, it is possible to configure the base URL for the file and web access or the DB connection strings via application settings.

4. EVALUATION

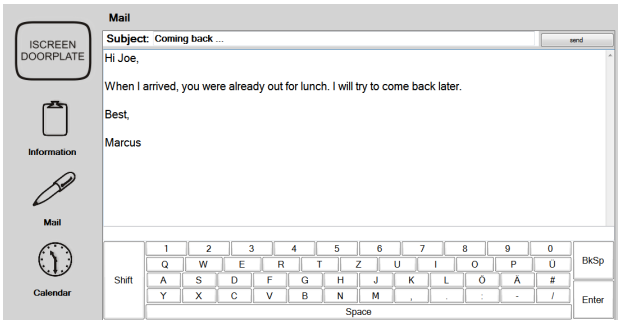
To evaluate the iScreen toolkit, we used it to develop a number of applications. In the following, we first present some of them. Thereafter, we describe our experiences during the application development as well as during day-to-day use.

4.1 Applications

To validate the toolkit, we used it as an application development platform in several student projects at the University of Bonn and Duisburg-Essen. In each of the projects, we were targeting a different use case that seemed applicable and useful in a university context. In the following, we briefly describe four resulting applications.



(a) Calendar



(b) Email

Figure 2: iScreen Doorplate

- iScreen Doorplate: The first application implements an interactive door plate. Similar to other smart door plates [5, 12], the interactive door plate displays information about the person that inhabits a particular room. Due to the fact that many businesses are already maintaining staff information on their home page, the information about the person is retrieved via the web. To do this, the HTMLBrowser component is configured via an application setting to display the person's homepage. In cases where a visitor does not meet the person directly, it is often helpful to know the schedule of the person in order to determine whether it makes more sense to wait or to come back another day. To enable this, the door plate displays the person's public appointments by means of the CalendarViewer (c.f. Figure 2(a)). Once the return time has been planned, it might be beneficial to leave a notice. To do this, the door plate enables the visitor to send an email via the EmailComposer (c.f. Figure 2(b)). To avoid fake emails, the recipient is hard coded via an application setting. Furthermore, the send button of the EmailComposer is only enabled once a face has been detected using the FaceDetector and CameraSnapshot components. The picture containing the face is then attached to the email to identify the sender.

- iScreen Kiosk: The second application is a in-store kiosk application that can be derived directly from the interactive door plate. In order to show in-store information, the HTMLBrowser can be reconfigured to display the associated web site and to allow browsing on it. Furthermore, using the CalendarViewer it is then possible to display special events. Finally, by redirecting the EmailComposer to the customer service, it is possible to facilitate customer feedback.

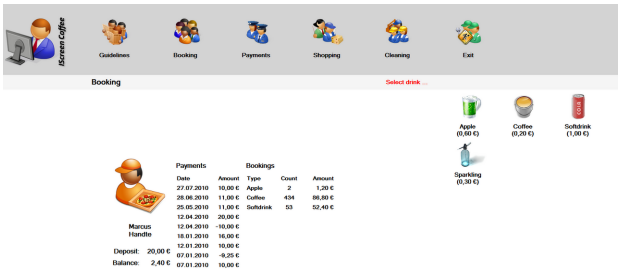
- iScreen Advertise: The third application implements a scenario in which the device is used to display advertisements similar to [10]. Using the FolderTransfer component, advertisements are loaded from a remote server on a daily basis. The advertisements are then shown using a configurable schedule that is downloaded together with the advertisements. In order to capture statistics, the CameraSnapshot and FaceDetector components are used to count the number of viewers for each advertisement. In addition, the BluetoothScan component is used to capture the number of persons in the surrounding of the display.

- iScreen Coffee: The last application serves as a touch-based replacement for tick-lists that we previously used to distribute softdrink and coffee cost among the members of our research group. Using the ImageButtonGrid component, iScreen coffee provides access to a number of features that were previously realized using pen-and-paper. Using the RTFViewer component, iScreen coffee can display a simplified manual for our coffee machine including cleaning instructions that are targeted at new employees. By hitting a booking button and selecting a particular account, a person can book different types of drinks (c.f. Figure 3(a)). Using the cost of the individual drinks as well as the person's consumption and payments, the application computes the total balance. By hitting the payments button and selecting a person, it is possible to add payments via the TouchKeyboard component that is configured to display only numbers (c.f. Figure 3(b)). Similarly, by hitting the shopping button, it is possible to add expenses which is used to compute the global balance of our coffee fund. To save power, the display is deactivated automatically after a short period and the CameraSnapshot and FaceDetector component are used to automatically enable the display as soon as a person is detected in front of it. Furthermore, to avoid data loss, the FolderTransfer component is used to backup the underlying database on a daily basis.

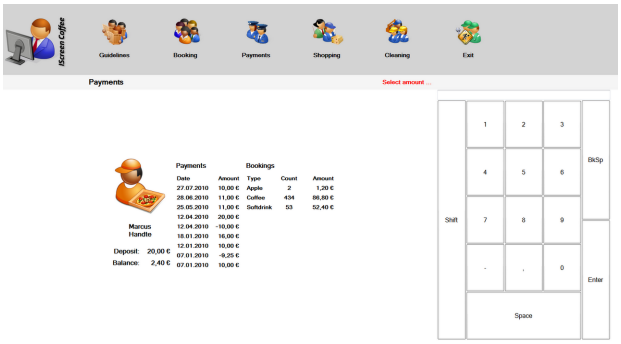
4.2 Experiences

During the development of the components and applications we found that most of the students had no problems in understanding the basic architecture as well as the interaction. Due to the clear separation of tasks introduced by the different extension points of the core system, the emitter, provider and renderer components were fairly easy to reuse. This is also indicated by the fact that the four applications presented previously share a set of common components. For example, both the kiosk as well as the plate application share an almost identical configuration. Similarly, the CameraSnapshot as well as the FaceDetector components are used in multiple applications. Thereby, they may either serve as an unobtrusive way to collect usage statistics or – as in the case of the coffee application – to enable the display on demand.

However, the opposite argument can be made about the control components. Since control components embody the actual application logic, they are hard to reuse in different settings. This is amplified by the fact that they tie to-



(a) Booking



(b) Payments

Figure 3: iScreen Coffee

gether the remaining components which results in a strong coupling. Although, it is possible to organize control components in a hierarchical fashion, we found that the lack of support for this kind of organization in early versions of the iScreen toolkit lead to implementations with large and error-prone conditional statements. To mitigate this, we extended the framework with base classes for wizard-like control components to wrap particular interaction sequences. This improved the code style by simplifying the development of modular (sub) control components. In fact, it was even possible to reuse the logic of some interaction sequences.

Although, we tested all applications for an extended time period, only one of them – the iScreen coffee application – has been successful in terms of long-term deployment. After several weeks of testing, we deployed a dedicated device in our coffee kitchen (c.f. Figure 4) to solely run the iScreen coffee application. Over the last two years, the application has been used by more than 20 people on a daily basis. During this time, the device has been running without interruption. Despite its low cost, we did not experience a single day of downtime yet. Besides from the robustness of the device and application, this is also partly due to the overall architecture. Due to the fact that the device integrates all required processing and storage capabilities, it even remained operational during times when our wireless network and our (redundant) storage servers experienced critical failures.

With respect to usage we found that even less technically versed users had no issues with learning the basic operation of the application within a few minutes. We attribute this partially to the fact that the actual functionality of the application closely mimics the pen-and-paper based solution and partially to the fact that the touch interface is straightforward. In fact, soon after having installed the application in our kitchen, we forwarded the source code to another research group which has been using it ever since. To meet some special requirements, the group extended some of the renderer components to show additional billing information. Since these extension were implemented without additional help or documentation provided by our group, this can be seen as another indicator for the extensibility of the system.

Figure 4: iScreen Coffee Installation

Given these experiences with application development, we argue that the iScreen toolkit meets its design goals with respect to genericity – as indicated by the number of different applications –, reusability – as indicated by the overlapping components used in different applications – and extensibility – as indicated by the external extension of the iScreen application. Furthermore, based on our experiences with application usage, we are convinced that touch-enabled integrated computer systems provide a cost-efficient and robust platform for many interactive pervasive applications.

5. RELATED WORK

Existing research on pervasive displays and applications can be broadly classified in work related to interaction methods and work related to system support. In the area of interaction methods, researchers have developed and studied a multitude of different approaches. This includes interaction through pens [2] or capacitive surfaces [8] which is now common in commercially available displays or smart boards. But it also includes interaction through cameras or sensor devices which capture the location and gestures of users in front of a large scale display that can be fixed [10] or steerable [11]. More recently, a significant amount of this research has focused on the use of mobile devices as a way of interacting with the screen. Examples include [1] which uses Bluetooth communication to share photos with a public display, [7] which proposes the use of visual codes shown on a display to download further information onto the mobile device or [6] which describes approaches to use camera phones as a mouse replacement. The work presented in this paper, does not attempt to propose a novel interaction technique, instead, we present a toolkit that simplifies the realization

of many common use cases proposed by others.

Research on system support can be further categorized into management systems that aim at managing large scale networks of displays and toolkits that aim at the development of applications. Examples for the first include [12] and [3]. These systems address issues resulting from the scale of the deployments such as the proper addressing and identification of devices, etc. Instead, our work addresses the development of applications running on a single autonomous device. Consequently, our work is more closely related to toolkits such as [13] or [9] which are geared towards a simplified development of display-based applications. However, while these are focusing on multi-user interactions, the iScreen toolkit is targeted at simplifying the development of single-user applications by composing common building-blocks.

6. CONCLUSIONS

Touch-enabled integrated computer systems provide a cost-efficient hardware platform for a pervasive deployment of interactive displays in a variety of environments. However, in order to be useful for a particular application scenario, their software needs to be customized. In this paper, we presented the iScreen software toolkit which minimizes the customization effort by providing a number of reusable components that can be composed depending on the application requirements.

As indicated by our example applications, the iScreen toolkit can reduce the development overhead without imposing significant restrictions on the application's design or functionality. Furthermore, as indicated by our experiences with the iScreen coffee application, touch-based interactive applications can provide an easy-to-use alternative to pen-and-paper – even when used by non-expert users on a daily basis.

At the present time, we are thinking about extending the toolkit to support the coordinated cooperation between multiple screens. This will simplify the implementation of more complex distributed applications such as follow-me advertisements or navigation.

Acknowledgments

This work is partially supported by CONET funded by the European Commission under FP7 with contract number FP7-2007-2-224053 and by LIVING++ funded by the BMWi under contract number KF2095019FR0.

We thank the participants of the iScreen student projects at the University of Bonn and Duisburg-Essen for their component implementations and their feedback on the toolkit's abstractions.

7. REFERENCES

[1] K. Cheverst, A. Dix, D. Fitton, C. Kray, M. Rouncefield, C. Sas, G. Saslis-Lagoudakis, and J. G. Sheridan. Exploring bluetooth based mobile phone interaction with the hermes photo display. In *7th int. conf. on HCI with mobile devices & services*, MobileHCI '05, pages 47–54, New York, USA, 2005.

[2] S. Elrod, R. Bruce, R. Gold, D. Goldberg, F. Halasz, W. Janssen, D. Lee, K. McCall, E. Pedersen, K. Pier, J. Tang, and B. Welch. Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. In *SIGCHI conference on Human factors in computing systems*, CHI '92, pages 599–607, New York, NY, USA, 1992. ACM.

[3] A. Erbad, M. Blackstock, A. Friday, R. Lea, and J. Al-Muhtadi. Magic broker: A middleware toolkit for interactive public displays. In *6th IEEE Int. Conf. on Pervasive Computing and Communications*, pages 509 –514, march 2008.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns. elements of reusable object-oriented software, 1994.

[5] F. Hupfeld and M. Beigl. Spatially aware local communication in the raum system. In *7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, IDMS '00, pages 285–296, London, UK, 2000. Springer-Verlag.

[6] S. Jeon, J. Hwang, G. Kim, and M. Billinghurst. Interaction with large ubiquitous displays using camera-equipped mobile phones. *Personal and Ubiquitous Computing*, (2):83–94, 2009.

[7] K. Mitchell, N. J. P. Race, and M. Suggitt. icapture: Facilitating spontaneous user-interaction with pervasive displays using smart devices. In *Workshop on Pervasive Mobile Interaction Devices (PERMID) at Pervasive*, 2006.

[8] J. Rekimoto. Smartskin: an infrastructure for freehand manipulation on interactive surfaces. In *SIGCHI conference on Human factors in computing systems*, CHI '02, pages 113–120, New York, NY, USA, 2002. ACM.

[9] C. Shen, F. D. Vernier, C. Forlines, and M. Ringel. Diamondspin: an extensible toolkit for around-the-table interaction. In *SIGCHI conference on Human factors in computing systems*, CHI '04, pages 167–174, New York, NY, USA, 2004. ACM.

[10] M. Strohbach and M. Martin. Toward a platform for pervasive display applications in retail environments. *IEEE Pervasive Computing*, 10(2):19 –27, feb. 2011.

[11] P. N. Sukaviriya, M. Podlaseck, R. Kjeldsen, A. Levas, G. Pingali, and C. S. Pinhanez. Embedding interactions in a retail store environment: The design and lessons learned. In M. Rauterberg, M. Menozzi, and J. Wesson, editors, *IFIP International Conference on Human-Computer Interaction*. IOS Press, 2003.

[12] W. Trumler, J. Petzold, F. Bagci, and T. Ungerer. Amun - autonomic middleware for ubiquitous environments applied to the smart doorplate project. In *International Conference on Autonomic Computing*, pages 274 – 275, may 2004.

[13] E. Tse and S. Greenberg. Rapidly prototyping single display groupware through the sdgtoolkit. In *5th conference on Australasian user interface*, volume 28 of *AUIC '04*, pages 101–110, 2004.