Interactive Prototyping for Ubiquitous Augmented Reality User Interfaces

Otmar Hilliges
University of Munich
LFE Media Informatics
otmar.hilliges@ifi.lmu.de

Christian Sandor
Technische Universität
München
sandor@in.tum.de

Gudrun Klinker
Technische Universität
München
klinker@in.tum.de

ABSTRACT

User interfaces for ubiquitous augmented reality incorporate a wide variety of concepts such as multi-modal, multi-user, multi-device aspects and new input/output devices. In this paper we present a twofold approach that consists of an execution engine for ubiquitous augmented reality user interfaces and a runtime development environment that enables rapid prototyping and live system adaption for such advanced user interfaces.

Categories and Subject Descriptors: H.5.2 User Interfaces, D.1.7 Visual Programming

General Terms: Management, Experimentation

Keywords: Software architectures, visual programming, augmented reality, tangible user interfaces, ubiquitous computing, multi-user, multi-modal

1. MOTIVATION

User interfaces in ubiquitous augmented reality (UAR) [1, 2] arise from the convergence of augmented reality and ubiquitous computing. The expressiveness of UAR is powerful as it inherits a variety of concepts such as multiple users, input devices and displays. In contrast to WIMP user interfaces, in which much research and usability evaluations have helped to establish its paradigm with idioms such as *Drag and Drop* or *Point and Click*, the interaction elements in UAR have not yet been formalized.

We present an approach for interactive prototyping of interaction management for UAR user interfaces. Our approach comprises two parts. First, the development of an execution engine for UAR user interfaces. Second, the creation of a runtime development system to alter the behavior of the user interface at system runtime.

Since designers are often less tech savvy, one of our main goals was to reduce the amount of programming necessary to create functional UAR prototypes. We choose a visual programming approach based on the formal model of Petri nets to explicitly model flexible adaption of I/O devices at runtime and data flow through the user interface.

2. RELATED WORK AND CONTRIBUTION

Papier-Mâché [3] is a toolkit for building tangible user interfaces (TUI) without having expertise in hardware design and computer vision. The toolkit consists of a library to adapt various sensor techniques and to abstract input hardware. Further it provides a high-level event model and an API for incorporating tangible input. Our approach contains a similar device abstraction and event model. We do not limit ourselves to TUI but support all kinds of input devices that are able to emit a defined set of events. We support a formal model to describe the flow of events and a visual programming environment while Papier-Mâché is a Java class-library.

Jacobs et.al propose in [4] a user interface management system. Likewise in our approach, continuous data streams and discrete events are distinguished, and a graphical editor for the specification of changes in the media design layer according to those events is provided. In virtual reality, where input hardware is to a certain degree standardized and known in advance, dynamic device exchange is not really an issue. With UAR, in contrast, we are concerned with semantical equivalence and flexible exchange of I/O devices and distribution of software components over several hosts. Additionally, we consider output fission and control of multiple output components to be an issue.

3. INTERACTION MANAGEMENT

The WIMP assumption that input occurs sequentially does not hold for UAR any longer. In UAR several modalities are used by multiple users in a truly parallel manner to interact with the system. Furthermore, the highly dynamic nature of UAR requires a way to flexibly adapt and exchange devices, possibly at runtime. Hence, a special-purpose mechanism has to enable UAR systems to give users the ability to interact with the system in the way they want, with their preferred devices. We call this interaction management.

In addition, interaction management has to allow UAR systems to adapt themselves to the current user and her preferences, as well as to the her context (e.g. current activity, users' inter-social engagement or the available devices). The system has to present content in the preferred way and optimized to the currently available display devices.

To meet these requirements, we have developed a user-interface architecture [1, 2] for UAR systems that enables multi-modal, parallel-user input as well as flexible device adaptation at runtime. For communication purposes we utilize DWARF [5], but the presented functionalities could be implemented on top of any sophisticated publisher/subscriber system.

We arrange the user interface components in three layers, *Media Analysis*, *Interaction Management* and *Media Design*. Most data flow linearly from the *Media Analysis* layer, which

Copyright is held by the author/owner.
IUI'06, January 29–February 1, 2006, Sydney, Australia.
ACM 1-59593-287-9/06/0001.

contains input components, to the *Interaction Management* layer, where the tokens are interpreted. From there the data flow continues to *Media Design* layer where the output components reside.

We have developed a standardized format for tokens that are sent from the input components to the *Interaction Management* layer. Due to this standardized format, we can exchange one input device for another - as long as they emit the same type of tokens.

Similarly, the *Interaction Management* layer sends commands to the *Media Design* layer. The commands consist of actions that have to be executed by the output components. One can interchange output components in the same way as with input components. Due to the flexible DWARF component model the exchange of I/O components works even at system runtime.

4. INTERACTION MANAGEMENT WITH PETRI NETS

Petri nets, or place-transition nets, are classical models of concurrency, non-determinism and control flow. Petri nets provide an elegant and mathematically rigorous modeling framework for dynamic systems. They also provide an easy-to-learn and understandable graphical notation.

A Petri net consists of places, tokens, arcs and transitions. The arcs connect places and transitions. Places and arcs may have capacities. A transition fires when all places at the end of incoming arcs contain enough tokens. Transitions execute actions when fired.

Places of Petri nets usually represent states or resources in the system, while transitions model the activities of the system.

Transitions are used to encapsulate atomic interactions. More complex interactions can be modeled by combining several transitions.

The characteristics exhibited by the activities in a multimodal user interface such as concurrency, decision-making and synchronization are modeled very effectively with Petri nets. In Figure 1 some of these characteristics are represented.

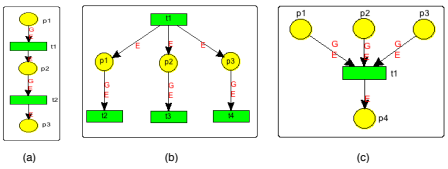

Figure 1: From left to right: Petri nets modeling Sequential Execution, Concurrency and Synchronization

We map input events to tokens that are put into incoming places. Code encapsulated in transitions extracts the content from user input tokens and interprets it. Tokens generated by transitions are sent to output components, thus enabling us to model interactions, and implicitly, the behavior of the user interface.

Transitions link inputs to a semantic entity. Transitions can be seen as *predicates* over input *attributes*. A transition encapsulates actions which are fired when the predicate evaluates to true, which causes a change in the user interfaces state; for example, the addition or removal of an item in a graphical view or a change of properties of some item in an output component.

A set of places, arcs and transitions forms an expression or, in terms of user interaction, a declaration of intent (e.g. insert an element to a view, selection/de-selection).

5. UI EXECUTION ENGINE

The core component of the *Interaction Management* layer is the User Interface Controller (UIC) [6]. It combines the functionalities of *Dialog Control* and *Discrete Integration*. It interprets input tokens sent by the *Media Analysis* components and then triggers actions that are dispatched to components in the *Media Design* layer.

The UIC has been implemented on top of the object-oriented Petri net framework JFern[1]. JFern's Petri net model is based on the traditional model of hierarchical Petri nets with the additional concept of *object-based* tokens - places can contain arbitrary Java objects as tokens.

We take advantage of two of its main features, the ability to use arbitrary objects, including DWARF events, as tokens and the possibility to describe transitions guards and actions in native Java code. That implies a high expressiveness of the single statements and very little learning required for programmers familiar with the Java language.

We have built a communication layer around the JFern kernel, connecting input places with components of the *Media Analysis* layer and connecting output places with components of the *Media Design* layer.

We utilized the described user interface execution engine to build the mixed-reality, multi-player game SHEEP which is described in [7, 8]. That approach showed that it is powerful enough to control the user interface of mixed-reality applications. It also gives developers and technically interested users remarkably easy insight at runtime into the processes occurring by visualizing on screen the Petri net execution.

6. RUNTIME DEVELOPMENT

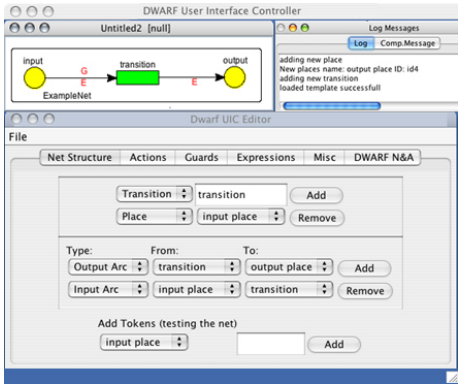

Figure 2: The UIC showing a very simple Petri net and the net structure modification tab.

Since the code-based approach utilized in SHEEP did not scale very well in terms of development time and methodology, we built a visual programming environment to specify Petri nets. This has the advantage of letting the developer

[1] `http://sourceforge.net/projects/jfern`

modify the user interface at runtime based on feedback from users. This also makes it easier to deal with the resulting complexity of the Petri nets by hiding the underlying XML representation in favor of a pure graphical notation.

The basic functionality of our tool is to define the net structure (Figure 2). This means to specify how many inputs and what sort of inputs are needed to execute one task (e.g. a gesture and a speech command) and also what sort of output is generated. With the net structure we also define how different tasks are related to each other.

6.1 Dynamic Code Modifications

To change the behavior of the system we need to modify the data manipulation that is done within the control structure. In our case this means to exchange the code of the *actions* which execute arbitrary Java code and have access to the tokens that reside in the input places. The code executed actually changes the state of attached components from the *Media Design* layer.

In most cases code boils done to a few lines. Essentially actions extract data from input tokens, which are DWARF events, and compose new DWARF events which are sent to the output components. We are able to exchange the code of the *actions* at runtime utilizing the Graham-Kirby-Compiler[2] and in consequence, that enables us to modify the user interface behavior dynamically.

6.2 Connectivity Management

Connections between I/O devices and the UIC are based on DWARF infrastructure and communication channels set up at runtime. Developers can define attributes on outgoing - and respectively predicates on incoming - connections. Within our tool the developer can add new incoming connections by attaching input components to input places of the Petri net. Further the developer can define predicates on that connection to select from different components which offer to produce matching events. The connections will be set up whenever a matching pair of data producers/consumers is present in the network environment. Whenever attributes or predicates change, the regarding connections are disconnected and, if available, new communication partners are connected. Output places can be modified accordingly. This allows us to flexibly adapt different output components and control which components receive what commands. So we can use different modalities to present content to the user or show different content on devices belonging to different users or user groups, e.g. private information vs. publicly available information. Another aspect of our architecture allows developers to keep full control over the granularity of their Petri nets. Since any arc in a Petri net can be replaced by a DWARF connection a developer can model everything in one self-contained component or, on the other end, have several interwoven components each modeling just one single interaction. Such atomic Petri nets can then be reused in different applications.

This implementation stage has been used in the CAR[3] [9] project. Its goal is to provide a collaboration platform for a team of interdisciplinary researchers to experiment with the next generation of car user interfaces. It incorporates concepts of mixed and augmented reality, adaptive user interfaces, multi-modal user interfaces and attentive user interfaces.

[2] `http://www-ppg.dcs.st-and.ac.uk/Java/DynamicCompilation/`

[3] `http://www1.in.tum.de/DWARF/ProjectBar`

7. CONCLUSION AND FUTURE WORK

In this paper we have presented an execution engine for UAR user interfaces that has been shown to meet the requirements for flexibility, adaptivity and powerful interaction management [2, 8].

Further we have presented a runtime development system for such user interfaces. No formal user studies have been carried out yet. Our experiences from building systems with it have shown that it has reached a level where programmers can use it to quickly assemble and tune UAR systems such as CAR and others[4].

While we succeeded in creating a tool for the technical user that is significantly easier and faster than writing code, we found that an even more visual approach would be necessary to enable technically unskilled designers to build such user interfaces since Petri nets and probably any formal model would, for them, be hard to understand and hence unintuitive to use.

8. ACKNOWLEDGEMENTS

Special thanks to DWARF Ph.D. students Thomas Reicher, Martin Bauer, Martin Wagner and Asa MacWilliams.

9. REFERENCES

[1] Christian Sandor. *A Software Toolkit and Authoring Tools for User Interfaces in Ubiquitous Augmented Reality*. PhD thesis, Technische Universität München, München, Germany, 2005.

[2] Christian Sandor and Gudrun Klinker. A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality. *Personal Ubiquitous Comput.*, 9(3):169–185, 2005.

[3] Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. Papier-Mâché: Toolkit Support for Tangible Input. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vienna, Austria, 2004.

[4] Robert J. K. Jacob, Leonidas Deligiannidis, and Stephen Morrision. A Software Model and Specification Language for Non-WIMP User Interfaces. *ACM Transactions on Computer-Human Interaction*, 6(1):1–46, 1999.

[5] Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riss, Christian Sandor, and Martin Wagner. Design of a component-based augmented reality framework. In *Proceedings of the International Symposium on Augmented Reality (ISAR)*, pages 45–54, October 2001.

[6] Otmar Hilliges. Interaction Management for Ubiquitous Augmented Reality User Interfaces. Master's thesis, Technische Universität München, Germany, 2004.

[7] Otmar Hilliges, Christian Sandor, and Gudrun Klinker. A lightweight approach for experimenting with tangible interaction metaphors. In *Proc. of the International Workshop on Multi-user and Ubiquitous User Interfaces (MU3I)*, 2004.

[8] Asa MacWilliams, Christian Sandor, Martin Wagner, Martin Bauer, Gudrun Klinker, and Bernd Brügge. Herding sheep: Live system development for distributed augmented reality. In *ISMAR '03: Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 123–132, Tokyo, Japan, 2003.

[9] Christian Sandor and Gudrun Klinker. Lessons learned in designing ubiquitous augmented reality user interfaces. In Michael Haller, Mark Billinghurst, and Bruce Thomas, editors, *Emerging Technologies of Augmented Reality: Interfaces & Design*. Idea group inc., 2006.

[4] `http://campar.in.tum.de/Chair/ResearchArProjects`