

# The WUI-Toolkit: A Model-Driven UI Development Framework for Wearable User Interfaces

Hendrik Witt, Tom Nicolai, Holger Kenn  
TZI, Wearable Computing Lab.  
University of Bremen, Germany  
{hwitt, nicolai, kenn}@tzi.de

## Abstract

*We introduce the “WUI-Toolkit” as a framework to support and ease the development of wearable user interfaces (WUIs). The toolkit presents a first step towards a model-driven UI design approach in wearable computing that allows even non UI experts the generation of usable and context-aware WUIs.*

*Based on an abstract model of an envisioned user interface that is independent of any concrete representation, the toolkit is able to generate a device- and context-specific UI for a given wearable computing system at runtime. The toolkit features the ability to use available context sources and can automatically adapt generated interfaces to maintain their usability. We present the current architecture including the capabilities of the abstract model and introduce a renderer developed to generate graphical WUIs suitable for monocular head-mounted displays in industrial applications.*

## 1. Introduction

The design of wearable computers with all their aspects is challenging, as is the design of their human-computer interfaces. Today, WIMP (Windows, Icons, Menus, Pointing) user interfaces are ubiquitous. They dominate desktop computers and even mobile devices. Although the use of the WIMP metaphor on mobile devices is often a burden, it is usually directly reused though. Consequently, its use becomes almost impossible when being constantly preoccupied with a primary task [3, 12]. For example, to control a mouse pointer constant visual attention to the interface is required. Spending almost all visual attention on the interface is, however, impossible when a primary task has to be performed simultaneously. Therefore, wearable computer call for new user interfaces that regard its unique characteristics of being involved in a mobile, dual-task situation while having always the possibility to make use

of available context information of the user, its activities, and environmental conditions.

Today, application designers can rely on a great amount of UI widget libraries, known and tested UI designs, and usability guidelines. Unfortunately, these can often not be applied to wearable UI design [3]. So far, known design knowledge for WUIs only exists as system descriptions of more or less successful implementations or comparative studies for single UI elements which make any further reuse difficult (see e.g. [2, 7, 11]). Because there are no common tools or frameworks available yet, wearable applications are currently still implemented from scratch.

Since we are involved in the wearIT@work project [6], we observed that application developers are often not aware of basic UI design rules. For wearable computers, they are even not aware of its most obvious properties, e.g., how limited presentation capabilities of a wearable device typically are. To overcome such problems but to also offer rapid prototyping of WUIs (e.g., needed in user studies) the Wearable User Interface Toolkit (WUI-Toolkit) is currently developed as a part of the European Wearable Computing Framework (EWCF) by the wearIT@work project consortium. The toolkits primary purpose is to provide a tool support for WUI development with reusable components and to reduce implementation efforts.

## 2. Related Work

There are various toolkits to interface physical artefacts to software. Some are related to robotics (playerstage, fastrobots), others are related to pervasive computing (iStuff, Phidgets). The Georgia Tech Gesture Toolkit (GT<sup>2</sup>K) [13] provides tools that support gesture recognition research based on hidden markov models (HMMs). For pointer-based ubiquitous computing devices, systems such as the SUPPLE toolkit [5] or Huddle [8] facilitate the automatic generation of UIs.

The Context-Toolkit [4] was proposed for the general integration of context into applications. Although some of these toolkits can support application development for wearable computers, to the best of our knowledge, there are no integrated solution available that facilitate context-aware WUI development with reusable components in a similar way.

Creating reusable context-aware WUIs is, however, a challenging process. It might include the design, choice, and layout of suitable UI components, the processing of observed context events, and the rendering of UIs. For parts of these aspects some work is available and also good WUI examples are known: The interface of the VuMan [1] is designed around a dial on the device. The graphical UI reflects the input device and arranges elements in a circle. A similar interface, reduced to eight selectable elements was proposed by Schmidt et al. [11]. In [2] a list oriented GUI operated by a special data glove device was shown. KeyMenu [7] is a UI component to be used in conjunction with the Twiddler chording keyboard. What is common to all these UIs is that they depend on special input devices which make any further reuse difficult.

### 3. The WUI-Toolkit

The ability to ease WUI development is the overall requirement of the WUI-Toolkit. A number of additional requirements to be fulfilled by such a toolkit were determined in cooperation with partners in the wearIT@work project and are discussed in [14]. For the scope of this paper, we will concentrate on the following subset of these requirements:

- **Device independent UI specification:** Usually, wearable systems have no fixed or default I/O device configuration like desktop computers where the presents of display, keyboard, and mouse can always be assumed. Wearable applications can make, e.g., use of gesture devices or chording keyboards and a HMD instead of an audio headset. Instead of relying on special devices, a UI for a wearable system should be specified independent of any specific I/O devices or modalities.
- **Support for context-awareness and automatic adaptation:** One strength of wearable computing is the availability of context information by using sensors to detect environmental conditions or user activities. Research has shown how sensors can be successfully applied to recognize a wide range of contexts. Special purpose context-awareness of applications is often achieved by attaching available context information to the ap-

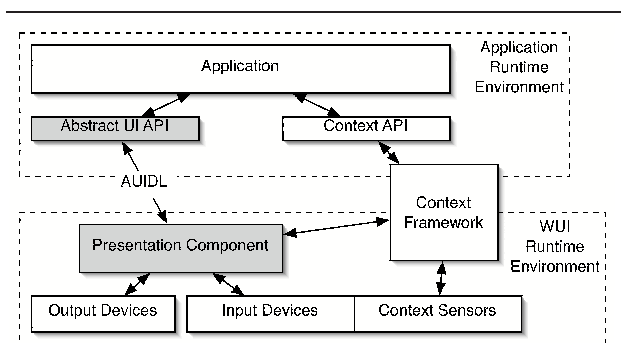


Figure 1. Architecture Overview of the Toolkit

plication in a “hard-wired” manner though. Unfortunately, there are a lot of reoccurring dynamic contexts or environmental conditions that can adversely affect the usability of a WUI. To achieve reusability, some of these factors can be adjusted automatically by the UI, e.g., changing the color contrast according to illumination. Thus, a global knowledge repository about such general WUI properties should be established that is reusable in different renderings.

#### 3.1. Architecture Overview

The overall design of the WUI-Toolkit architecture follows an event driven approach. Figure 1 shows the basic components of the architecture (shaded boxes) and their interfaces to other systems. The current version is implemented in Java 2 Standard Edition 1.3 which makes it available for a wide range of devices.

The envisioned UI for a wearable application is specified by the application developer through the *Abstract UI API*. While doing so, an *abstract model* of the UI is formed that holds its properties and capabilities. The abstract model can be serialized into another representation, the so-called *abstract user interface description language* (AUIDL), for remote communication and internal management purposes.

The *Presentation Component* receives information about the envisioned user interface to be rendered in the form of AUIDL. Its purpose is to mediate between the application and the user through a generated user interface. By interpreting the abstract model and consecutively mapping abstract entities to suitable concrete interface components, the *presentation component* delivers the generated UI to the user and handles user input. If user input is received through input devices that is relevant for the applications, e.g., triggering a database query that changes the applications data model, it is forwarded to notify the application.

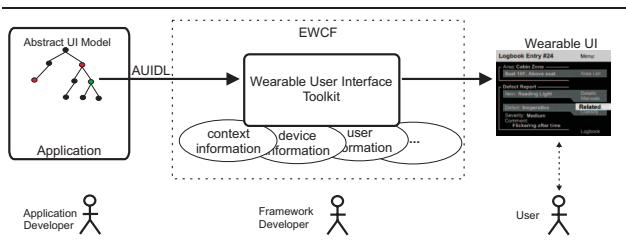


Figure 2. Actors Using the WUI-Toolkit

If there are multiple options during the mapping of abstract entities to concrete interface components, the *presentation component* has to decide which components and devices to use. For example, a text string can be rendered as written text on a display device or as speech output on an audio output device. To decide this, rules and constraints encoded as expert knowledge in the *presentation component* are used. Here, the user’s context may be considered during the decision process, too. For instance, light conditions of the environment are used to determine colors and font sizes for text rendering on display devices.

While the *presentation component* is running, the context of the user may change. To get notified if such changes occurred, an interface to a *context framework* is needed, that triggers events whenever sufficiently large context changes happened. Then, the decision process may be restarted which might change the configuration of already instantiated UI components, but may also change the interaction style by switching I/O from one device to another.

For all low-level aspects such as communication with I/O devices, event queuing, or configuration management, a *core framework* layer (not depicted in figure 1) is available within the *presentation component* that abstracts from low-level details.

## 4. Using the WUI-Toolkit

The WUI-Toolkit is organized in a way that makes it easy to integrate in any application development process. Its separation of concerns approach offers a separation between the *use* of the toolkit and the actual *development* of the toolkit itself. Figure 2 shows the roles of different actors involved in the process of building a WUI with the toolkit:

*Application developers* specify the abstract model of the required UI without paying attention to specific I/O devices, layouts, or computing paradigms to be used. Optionally, custom application specific knowledge, e.g., on context-awareness can be specified to augment context awareness capabilities that are integrated by default in generated UIs of the WUI-Toolkit.

The *framework developers* implement and maintain the actual WUI-Toolkit including its core functionalities such as layout of interface components, used interaction, and basic context awareness. They, e.g., integrate specific knowledge about wearable computing, its I/O devices, and interaction constraints. Finally, a *user* can interact with the application through the UI generated by the toolkit.

### 4.1. Abstract Model Specification

Because the abstract UI model is independent of I/O devices and interaction styles, it does not contain objects like buttons or windows, which are common to WIMP desktop applications, but not to WUIs. Instead, the envisioned user interface’s structure is specified with abstract elements. The designer specifies the envisioned interface in a dialog or task based style, i.e. thinking in hierarchy of steps in sequential order. This is very similar to GUI programming where different panels (dialogs) are usually grouped in separate classes that are afterwards recursively linked together to a tree structure in a main window class. There are currently seven elements available to specify such dialog:

An **Information** represents an atomic piece of information presented to the user. In most cases it contains text data, but may also contain multimedia content such as an image, a sound, a video, or a combination of alternate contents. With the latter mechanism, the toolkit can offer different representations of the same piece of information appropriate in a particular situation.

A **Group** is derived from an **Information** and is used to structure and semantically relate information. With its abstract design, it can group arbitrary abstract elements, in particular, also other **Groups**. For example, each dialog presented to the user might be on the top level of a group; indicating that a set of elements belongs together.

A **SelectionList** is a specialization of a **Group**. It can be composed of a set of **Information** elements and a set of **Triggers** which apply a function to an item in the list.

An **ExplicitTrigger** represents an action by the user. To activate the trigger, explicit interaction of the user is required. An example would be an action that involves changes in the data model of the business logic of an application. For instance, a “save changes” option would be specified by using an **ExplicitTrigger**.

In contrast to an **ExplicitTrigger** that facilitates boolean input only, an **ExplicitTextInput** can be used to gather text input from the user. Note, that the use of an **ExplicitTextInput** element is re-

stricted to real text input entered by the user. The actual input device used to gather the input can be any device that allows a transformation of the input into a text representation. Devices such as speech recognizers, keyboards, or sign language recognizers are all examples for suitable devices.

**4.1.1. Modelling Asynchronous Behaviors** All elements introduced so far can be used to specify a range of standard interfaces. However, neither of those can involve complex processing or significant computation load that would delay feedback nor can they notify the user or the rendering system about events occurring asynchronously in the background of a running business logic. Both aspects are crucial though. The first calls for a concept similar to threads that allows the application to continue without freezing the UI. Since the *application* and *presentation component* communicate only through the abstract UI model and may sit on different systems, simple threads are not sufficient.

The abstract **Process** element is designed for this purpose. It provides a structure to encapsulate complex and time consuming code execution sequences. If a **Process** is used, its contained code will be executed once activated in a separate thread on the system running the application. Entities interpreting or querying the abstract UI model will be informed about the progress of a running process. With this, the rendering system can provide necessary user feedback once an operation executed lasts for a longer period. Additionally, the semantic information of knowing about a potentially time consuming task a priori of rendering may change rendering outcomes.

In wearable computing the availability of context information is one of the major advantages. Implicit interaction is what may change the user interface but is not explicitly triggered by the user [10]. To make use of it, asynchronous communication is essential. Context systems usually acquire and infer sensor information continuously. Thus, context changes are likely to occur asynchronously to the execution of a program that makes use of them and call for a notification mechanism similar to a publisher/subscriber design pattern. An **ImplicitTrigger** element is available to handle implicit interaction in the abstract model. It can be activated either indirectly by the user or independent of the user based on its specific behavior. The behavior of a trigger, i.e. what has to happen to activate the trigger, is encapsulated in a **Behavior** object. To implement a particular context awareness of the envisioned user interface custom behaviors can be designed.

## 4.2. The Rendering Process

The interpretation of an abstract model and the rendering of a corresponding concrete user interface is the actual core functionality of the WUI-Toolkit. After instantiation, the presentation component detects available I/O devices and reads user profiles and system configurations. Based on this it loads appropriate device drivers and selects an appropriate so-called UI *rendering scheme* from a repository. A rendering scheme consists of a number of I/O device adapter classes, concrete UI components, and a layout and interaction manager that combines interface components and available I/O devices. Besides the actual rendering that involves a certain rendering scheme, the *adaptation module* is involved. The adaptation module allows to equip generated user interfaces with some basic context-awareness by default. It is described in section 5.

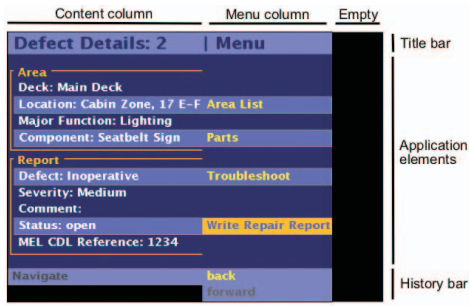
During the rendering process the presentation component identifies, for each entity in the abstract model, an equivalent UI component of the current rendering scheme. For instance, an **Information** containing text might be rendered through a text-to-speech UI component for an “audio headset” rendering scheme. To identify a suitable UI component for an abstract entity, the toolkit tries to find a corresponding representation by solving a constraint satisfaction problem (CSP) originated from constraints defined in the rendering scheme. For this, environmental and user contexts as well as design constraints are queried and evaluated.

Each concrete UI component may adapt itself to the current device, user, and context, based on a certain rendering scheme. A text-to-speech component might choose the appropriate sound output level based on context, the output speed based on user preferences and the sample rate based on the available sound output device. Because context may change over time, both the components and the presentation component communicate with a context observer. Periodic updates of the context are received when relevant context changes occur.

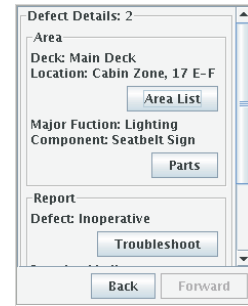
To support large scale context changes, that require, e.g., a change from a head-mounted display to a hand-held display, the state of the individual UI components is not kept in the components themselves. Instead, it is kept by the *presentation component* that manages all available rendering schemes and allows to change the current scheme. Figure 3 shows two renderings based on the same abstract model but with two different rendering schemes changed at runtime.

### 4.2.1. The “Wearable UI” Rendering Scheme

A number of schemes have been implemented so far. Apart from schemes based on the default Java GUI



(a) Wearable UI Rendering Scheme



(b) Hand-Held UI Rendering Scheme

**Figure 3. Different Interface Renderings of the same Abstract Model**

frameworks for hand-held devices (see figure 3(b)), a “Wearable UI” scheme was implemented (see figure 3(a)). The latter is optimized for wearable applications that use monocular head-mounted displays (HMDs). The graphical components and their layout are currently implemented for a MicroOptical SV-6 monocular “see-around” HMD. With a few changes, the scheme can be adopted to HMDs that have different resolutions, color depths, or are of “see-through” type.

The chosen layout is a two-column layout based on findings from [15]. The important content is placed in the left column for users wearing the HMD on the right eye. For a left eye usage the situation is reversed. Interaction is reduced to a one-dimensional process in menu selection style. The graphical interface reflects this by interactive items arranged in a vertical list in the right column of the screen (see figure 3(a)). The content is arranged vertically. Borders are used to create groups of elements. Multiple pages arrange content into blocks fitting on the screen if information exceeds the screen estate. Visual elements connect specific parts of the content on the left to menu items on the right to express coherence. Finally, a history function is added at the bottom of each screen that provides navigation to next and previous dialogues similar to a browser.

In contrast to desktop screen design, a dark background is chosen with bright foreground colors. On a “see-around” HMD, dark colors tend to be perceived as being transparent. A bright background might irritate the user. Large fonts are chosen for comfortable reading on the small display with respect to the users visual acuity specified in the user profile. Independent of the default color settings, colors can be automatically changed if, e.g., an illumination context is available (see section 5).

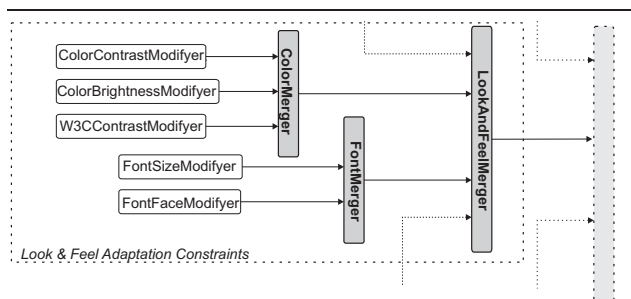
Since menu selection is used as interaction style, I/O devices need to provide only three basic events to control the menu: *cursor\_up*, *cursor\_down*, and *select\_item*.

Although others are imaginable such as touch wheels, the scheme currently supports different speech input devices that allow hands-free operation and a glove based gesture interface for control. Here, a multi-modal interaction, i.e. using speech and gestures at the same time for interaction, is possible.

## 5. Context-Awareness and Adaptation

External context providers offer the toolkit the context data needed to generate context-aware WUIs. Since context-aware UIs can become complex to implement, the toolkit integrates a set of basic reusable adaptation capabilities that let all generated WUIs automatically adapt themselves by default to a number of changing contexts, e.g. different illuminations. At present, “look & feel” related UI constraints are available to optimize colors, fonts, etc. on a HMD. It is planned to extend these sets with more complex constraints once new insight is gained on the coherence of certain properties of a WUI. An open interface allows to add new constraints also by other researchers or application developers.

To achieve adaptation, the toolkit uses a rating approach and tries to satisfy existing adaptation constraints to an optimal level similar to [5]. Because context changes, the optimization is a continuously ongoing process. To determine the current value of a property, different independent *modifier* functions compute a *local* optimal value based on their internal rules and propagate the results to *mergers*. Mergers are responsible for one or more properties. They merge different values of the same property by applying a rating to compute a *global* value and propagate this to the next level. Once values of a property are propagated to the highest level, the rendering process will be invoked to apply the new values on the generated interface. Figure 4 shows a section of such a resulting network used to compute “look & feel” related properties of an interface. By rearranging the order or adding new *mod-*



**Figure 4. Look & Feel Adaptation Network**

ifiers or mergers, it is possible to rapidly test or generate new adaptation behaviors that are immediately applicable during the interface generation without re-implementation.

## 6. Conclusion and Future Work

We introduced the WUI-Toolkit as a framework to support and ease the development of WUIs with reusable components. Its model-driven approach allows even non UI experts the design of usable WUIs. Moreover, it offers a mechanism to define different rendering schemes for multiple UI generation. For this, an abstract description of the envisioned UI is needed, that is independent of any specific wearable computing knowledge or interaction paradigm. We presented an overview of the architecture and current capabilities of the toolkit, including the basic steps to specify an abstract UI model and to generate an interface from this model. Besides a graphical WUI rendering scheme optimized for monocular HMDs, we briefly discussed the rendering of simple GUIs for hand held devices with another scheme available. Basic context-awareness of these interfaces is implemented with the extensible constraint based adaptation module and a set of adaptation rules.

Although indicators for both usability of the toolkit and functionality of the model-based approach were gained from various projects that used the toolkit (e.g. [9]), a detailed evaluation is still needed. For this, partners of the wearIT@work project will use it for a long term evaluation. Future work also includes the refinement of the abstract model as well as the design of further rendering schemes, e.g., for audio interfaces. To improve context-awareness of generated interfaces, user studies will be conducted to investigate the kind of automatic adaptation useful in wearable computing environments.

### Acknowledgment

This work has been funded by the European Commission through IST Project wearIT@work: Empowering the Mobile Worker with Wearable Computing (No. IP 004216-2004).

## References

- [1] L. Bass, C. Kasabach, R. Martin, D. Siewiorek, A. Smailagic, and J. Stivoric. The design of a wearable computer. In *Conference on Human Factors in Computing Systems (CHI '97)*. ACM Press, 1997.
- [2] M. Boronowsky, T. Nicolai, C. Schlieder, and A. Schmidt. Winspect: A case study for wearable computing-supported inspection tasks. In *ISWC '01*, pages 163–164, 8–9 October 2001.
- [3] A. F. Clark. What do we want from a wearable user interface? In *Workshop on Software Engineering for Wearable and Pervasive Computing*, page 3, Limerick, Ireland, June 2000.
- [4] A. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HCI Journal*, 16(2-4):97–166, October 2001.
- [5] K. Gajos, D. Christianson, R. Hoffmann, T. Shaked, K. Henning, J. J. Long, , and D. S. Weld. Fast and robust interface generation for ubiquitous applications. In *UBICOMP'05*, Tokyo, Japan, September 2005.
- [6] IST Project WearIT@Work - Empowering the mobile worker with wearable Computing (No. IP 004216-2004). <http://www.wearitatwork.com>, 2007.
- [7] K. Lyons, N. J. Patel, and T. Starner. KeyMenu: A keyboard based hierarchical menu. In *ISWC '03*, pages 240–241, White Plains, NY, October 2003. IEEE Computer Society.
- [8] J. Nichols, B. Rothrock, D. H. Chau, and B. A. Myers. Huddle: automatically generating interfaces for systems of multiple connected appliances. In *UIST '06*, pages 279–288. ACM, 2006.
- [9] T. Nicolai, T. Sindt, H. Kenn, J. Reimerders, and H. Witt. Wearable computing for aircraft maintenance: Simplifying the user interface. In *3rd International Forum on Applied Wearable Computing (IFAWC)*, VDE Verlag, March 2006.
- [10] A. Schmidt. Implicit human computer interaction through context. *Personal and Ubiquitous Computing*, 4(2/3), 2000.
- [11] A. Schmidt, H.-W. Gellersen, M. Beigl, and O. Thate. Developing user interfaces for wearable computers: Don't stop to point and click. In *IMC2000*, Warnemünde, November 2000. Fraunhofer Gesellschaft.
- [12] T. Starner. Attention, memory, and wearable interfaces. *IEEE Pervasive Computing*, 1(4):88–91, 2002.
- [13] T. Westeyn, H. Brashear, A. Atrash, and T. Starner. Georgia tech gesture toolkit: supporting experiments in gesture recognition. In *ICMI '03*, pages 85–92. ACM, 2003.
- [14] H. Witt. A toolkit for context-aware wearable user interface development for wearable computers. In *Doctoral Colloquium at ISWC'05*. IEEE, 2005.
- [15] H. Witt, T. Nicolai, and H. Kenn. Designing a wearable user interface for hands-free interaction in maintenance applications. In *PerCom'06*, Pisa, Italy, 2006. IEEE Computer Society.