A Prototyping and Evaluation Framework for Interactive Ubiquitous Systems

Christine Keller, Romina Kühn, Anton Engelbrecht, Mandy Korzetz, and Thomas Schlegel

TU Dresden - Junior Professorship in Software Engineering of Ubiquitous Systems
{christine.keller,romina.kuehn,mandy.korzetz, thomas.schlegel}@tu-dresden.de, anton.engelbrecht@mailbox.tu-dresden.de

Abstract. Ubiquitous systems often come with innovative design ideas and interaction concepts. To enhance and ensure the user's acceptance, it is necessary to test and evaluate those ideas in early design stages. In addition, early tests also validate the feasibility of those concepts. Rapid prototyping of ubiquitous systems enables researchers and practitioners to quickly test and implement new ideas, but is also necessary in iterative system development. We introduce a framework that supports rapid prototyping and evaluation of ubiquitous interactive systems using a modular approach, incorporating different interaction modes.

Keywords: Rapid Prototyping, Framework, Ubiquitous Systems, Interaction.

1 Introduction and Motivation

Ubiquitous computing aims at building intelligent environments, where computing devices of all sorts are pervasive but unobtrusive, as first envisioned by Mark Weiser [12]. A ubiquitous environment is supposed to support its users by providing easy information and computing access as well as usable interfaces. The key is *"Getting the computer out of the way"* [13]. Weiser's vision turned 20 in 2011 and although our computers are not out of the way yet, computing devices of all shapes and sizes become increasingly pervasive. However, most computing devices are standalone systems, lacking intelligent mechanisms to exchange data or incorporate context information, but also missing interaction concepts that ease the user's access to the system. Research for ubiquitous systems involves the design of innovative interaction concepts. It is necessary to test and evaluate those ideas in early design stages to avoid design errors. Prototypes are essential for developing and evaluating interaction concepts for ubiquitous environments. As Mark Weiser already stated 1993, *"the research method for ubiquitous computing is [...] the construction of working prototypes [...]"* [13]. Prototyping interactive ubiquitous systems facilitates the user-centered design process in ubiquitous computing and supports the development of systems that "get out of the way".

We developed a prototyping and evaluation framework for ubiquitous interactive systems, named *ProtUbique*. Our framework was designed and implemented to support rapid prototyping of interaction concepts for ubiquitous environments. It provides several components that implement different interaction channels to support a variety of

© Springer-Verlag Berlin Heidelberg 2013

modalities. Our goal is to enable a prototyping engineer to rapidly assemble different interaction channels, to provide any level of background code and then to evaluate this prototype with user tests using the same framework. In the following paragraph, we will take a look on related work in the field of prototyping for ubiquitous systems. We will then present our prototyping tool for interactive systems and describe our realization. After that, we give an example of a prototype that was developed using ProtUbique and show, how the framework can be used. We conclude with the discussion of our framework and future work.

2 Related Work

A *prototype* is a partially realized system that serves as example of a planned system. It can be used to test implemented functionality, to assess design decisions or to evaluate system concepts. *Rapid prototyping* tools and frameworks allow software engineers and system designers to quickly assemble prototypes and are often used in iterative software development processes [7]. In order to design usable systems it is important to involve potential users in early stages of the development process, to evaluate design ideas and to improve the system concept [4], [2]. In the user-centered design process, prototyping is a key technique to evaluate and improve an interactive system [8]. Paper prototypes and the like can serve as a starting point for the requirements analysis [5]. However, it is also important to build technically mature prototypes, in order to evaluate and test ubiquitous interaction [6].

Several research efforts aim at supporting rapid prototyping for ubiquitous systems. The Context-Toolkit from Dey, Abowd and Salber is a distributed context-aquiring and handling toolkit [3]. The Context-Toolkit provides context-widgets, interpreters and aggregators to abstract, hide and reassemble sensor-data for context-aware applications. Because of its service-oriented architecture the different components can be implemented in various programming languages. The Toolkit supports rapid prototyping of context-aware ubiquitous systems, and although they focus on providing context-aware interaction, the implementation of the interaction itself is not supported, in contrast to our framework. More focused on prototyping interaction is the iStuff toolkit developed by Ballagas et al. [1]. It supports interaction on displays via physical tangible devices. The toolkit allows any physical object or device that has a wireless interface to be an input or output device by defining it as an *iStuff* component. An iStuff component then is connected to a central system. The toolkit supports multiple users, devices and applications and is therefore very adaptable to different scenarios and fields of application. The authors write, that *"event communication takes only a few lines of code"* in order to utilize iStuff [1]. They also provide some output devices, for example a vibrating device for haptic output, they call *iVibe*. However, the toolkit does not support other interaction techniques but tangible devices, whereas we focus on facilitating different interaction modalities.

The Distributed Wearable Augmented Reality Framework (DWARF) by Christian Sandor and Gudrun Klinker is a "software infrastructure that allows the rapid exchange of interaction styles" [11]. According to Sandor and Klinker there are currently three aspects of interaction of future human-computer interfaces: mobility, multichannel-communication and interactions embedded in the real world. DWARF adresses these three interaction-styles and is built of loosely coupled distributed components. It has a layered architecture that includes a hardware layer, which manages the sensors and recieves the input-data. This data is then processed and interpreted by an interaction-management layer. On top of that, a media-design-layer takes care of the output or routing of the data. The implementation and utilisation of interaction components however is more complex as our approach, where interaction channels are modified via graphical user interface.

The proximity toolkit by Marquardt et al. facilitates the creation of so-called proxemic-aware applications [9]. These are applications that use the distance and orientation towards entities to realize interaction, where enities can be people, digital devices and objects. Orientation, distance, motion, identity and location information between entities is captured via marker-tracking and cameras in an laboratory. These measures are provided to prototype engineers for usage in the implementation of interactions. The toolkit is also able to record and replay events that are generated by the tracked entities. The proximity toolkit provides a visual monitoring tool, that allows to observe and record the proxemic relationships of entities in three dimensional space. The architecture of the proximity toolkit separates sensing hardware from the processing layer. This way, different sensing technologies can be used. The layered architecture provides flexibility and extensibility, which is also supported by the plugin-concept that enables a prototyping engineer to use predefined templates for interactions to support rapid prototyping. The installation of cameras is fixed and therefore not portable. The proximity toolkit only provides one means of interaction based on the described relationships between entities, other interaction techniques are not planned. Our framework, in contrast, focuses on providing different interaction channels and on extensibility. Norrie and Murray-Smith show that the Microsoft Kinect sensor can simulate a proximity sensor for spacial interaction without the installation of special hardware [10]. Furthermore, the authors suggest additional sensor types that can be simulated using Microsoft Kinect data. Those are an accelerometer, a pose sensor, an occupancy sensor, a motion sensor, a light sensor and a sound meter. These sensors can then be used in prototypes for interactive ubiquitous systems. Their implemented tool simulates a proximity sensor by tracking the spatial position of the user's hand. The authors use it for a mobile application that utilizes the proximity data and displays useful information if the user points his mobile phone to different spots. Although the authors suggest the usage of kinect data for additional sensors and possible interactions, they did not implement these yet. Therefore, only one interaction technique is supported at the moment.

It is our opinion that toolkits or frameworks that support prototyping of interactive ubiquitous systems need to be highly flexible to support many different domains, since there are numerous fields of applications for ubiquitous systems. As multimodal interaction is very important for ubiquitous systems, they need to provide different means of interaction for the prototype. In the following sections we will therefore present our framework *ProtUbique*, that allows for rapid prototyping of interactive ubiquitous systems. It supports different interaction techniques and easy adaptation via graphical user interface. Its architecture is modular, encapsulating the different interaction channels and therefore facilitates the extension by additional interaction channels.

3 A Rapid Prototyping Framework for Ubiquitous Systems

We developed *ProtUbique*, a framework that facilitates the user-centered design of ubiquitous systems. It supports easy and rapid prototyping of interactive ubiquitous systems and also enables the evaluation of these prototypes. ProtUbique focuses on interaction prototypes and for this purpose provides an abstraction layer for the prototyping engineer that encapsulates different interaction channels for supporting a variety of modalities. The prototyping engineer therefore doesn't have to implement several interaction techniques but can use the readily implemented interaction components and easily plug them into his backend code, as displayed in figure 1. Because the backend code provides the program logic and is not restricted in any way, various prototypes for different interactive ubiquitous fields of application become possible.

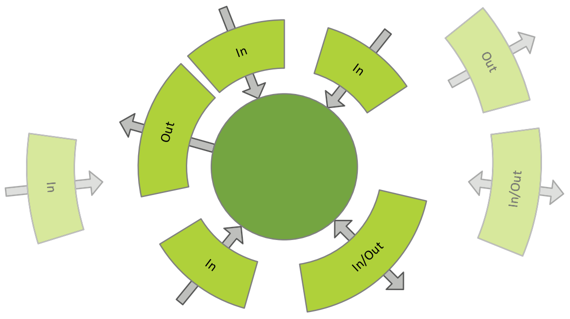


Fig. 1. Plugging together different implemented interaction channels for a new prototype

With our approach we are able to reduce the effort needed for programming. The user can quickly develop mixed or high-fidelity prototypes depending on the maturity of the backend code. The developed prototypes are then able to provide more or less functionality depending on the implemented backend code. In doing so several interaction techniques can be combined by using pre-implemented channels or by adding self-implemented channels to the framework.

Our goal is to facilitate the usage of different interaction channels without limiting the flexibility of their application. The interaction channels are therefore made available through a graphical user interface that allows customizing them. The interaction channels generate events that can be used to trigger responses to the specific interaction. These events have to be used in backend code. We intend to support as much customizing as possible through the graphical user interface. This way, prototyping engineers can concentrate on designing the prototype and providing backend code rather than implementing interaction techniques. In order to support the user-centered design of interactive ubiquitous systems, the ProtUbique framework also enables user tests. Prototypes that are realized within the ProtUbique framework can be evaluated within the framework itself. With the help of the framework interface user tests can be recorded and played back, showing all captured interactions. The following sections present our modular system architecture, the graphical user interface (GUI) and current implemented interaction channels in detail.

3.1 Architecture and Graphical User Interface

The first few interaction channels were implemented using the Microsoft Kinect sensor for the Xbox 360, which allows for various input modes. The ProtUbique framework is implemented in C#. We focused, however, on keeping the framework extensible and on allowing the easy development of additional interaction channels, as displayed in figure 2.

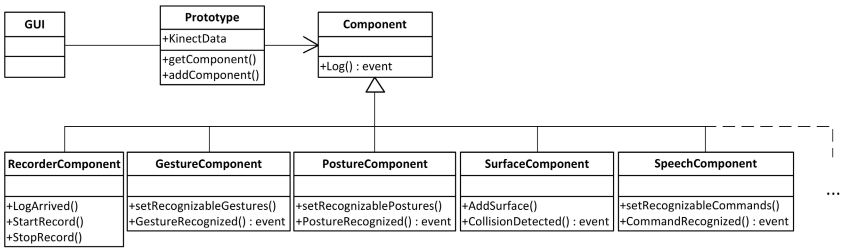
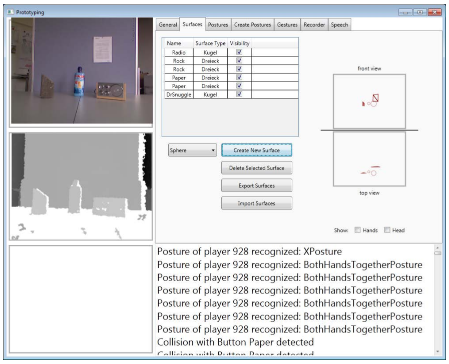


Fig. 2. The modular architecture of ProtUbique

Flexibility in ProtUbique is provided by the `Prototype` class. Its purpose is to mediate between the different hard- and software that realizes the interaction channels and the backend code that is provided by the user. An interaction channel can either provide output or input capabilities or both. An output component would consume data to deliver it to the user in the defined way, where an input component creates events whenever an interaction is recognized. The `Component` class represents general interaction channels. Individual channels can be realized as specializations of the `Component` class. If an implemented interaction channel should be applied in a certain prototype, the corresponding object is registered in the `Prototype` class, which then initializes the interaction channels and delegates resulting events to the backend code or delivers output data to output channels. A registered interaction channel can be tailored for the prototype it should be used for.

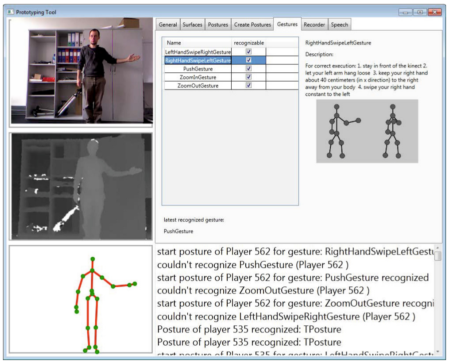
A prototype engineer must plug the events created by the interaction channels into his backend code, but apart from that should not have to write much additional code. This is why we decided to provide a **graphical user interface** (GUI) that facilitates the configuration of the selected interaction channels for a new prototype. The first step of creating a new prototype using ProtUbique is always the generation of a `Prototype` class and the registration of the used components. Each component then initializes a tab in the graphical user interface of ProtUbique, in which it can be configured. Since most of the implemented interaction channels use the Kinect sensor, we used its depth frame and VGA image to display the space that is used for the prototype, as shown



Name	Surface Type	Visibility
Radio	Kugel	✓
Rock	Dreieck	✓
Rock	Dreieck	✓
Paper	Dreieck	✓
Paper	Dreieck	✓
DrSnuggle	Kugel	✓

```
Posture of player 928 recognized: XPosture
Posture of player 928 recognized: BothHandsTogetherPosture
Posture of player 928 recognized: BothHandsTogetherPosture
Posture of player 928 recognized: BothHandsTogetherPosture
Posture of player 928 recognized: BothHandsTogetherPosture
Posture of player 928 recognized: BothHandsTogetherPosture
Posture of player 928 recognized: BothHandsTogetherPosture
Collision with Button Paper detected
```

(a) Drawing interactive surfaces



Name	recognizable
LeftHandSwipeRightGesture	✓
RightHandSwipeLeftGesture	✓
PushGesture	✓
ZoomInGesture	✓
ZoomOutGesture	✓

RightHandSwipeLeftGesture

Description:

For correct execution: 1. stay in front of the kinect 2. let your left arm hang loose 3. keep your right hand about 40 centimeters (in x direction) to the right away from your body 4. swipe your right hand constant to the left

latest recognized gesture:

PushGesture

```
start posture of Player 562 for gesture: RightHandSwipeLeftGestu
couldn't recognize PushGesture (Player 562 )
start posture of Player 562 for gesture: PushGesture recognized
couldn't recognize ZoomOutGesture (Player 562 )
start posture of Player 562 for gesture: ZoomOutGesture recogni
couldn't recognize LeftHandSwipeRightGesture (Player 562 )
Posture of player 535 recognized: TPosture
Posture of player 535 recognized: TPosture
```

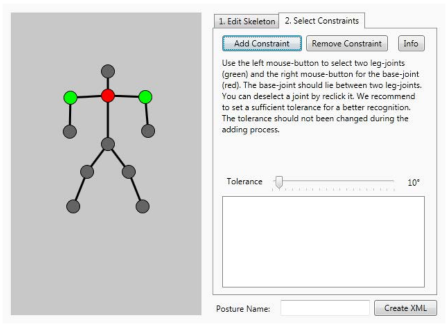
(b) Using implemented gestures

Fig. 3. The GUI of the ProtUbique framework

in figure 3(a). The left column of the graphical user interface consists of the VGA image on the top and the depth frame in the middle. There is also a presentation of the recognized skeletons on the bottom of the column on the left hand side. The left column can easily be omitted, if no Kinect sensor is available. On the bottom of the ProtUbique GUI, there is an output panel. All fired events and therefore all recognized interactions are displayed there, which is also shown in figures 3 and 4(b). They also display the different configuration tabs of the interaction channels.

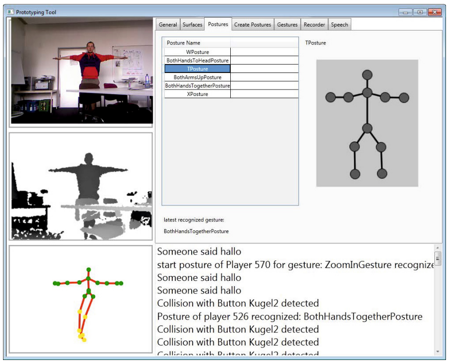
3.2 Implemented Interaction Channels

Up to now, we implemented postures, gestures, speech and interactive surfaces to support rapid prototyping of interactive and possibly multimodal ubiquitous systems. The `PostureComponent` can recognize designated **postures** of the skeleton detected by the Microsoft Kinect sensor. The Kinect is able to detect and distinguish 20 skeleton joints. We only use eleven of the skeleton joints for posture detection. A posture is defined using their relative positions. Constraints restrict the angle between two skeleton joints. The "Posture Creator" tab of the ProtUbique GUI provides a graphical tool for defining postures, as shown in figure 4(a). The skeleton joints that can be used for posture definition, are displayed on the left in the GUI. The skeleton can be edited by clicking on the nodes and dragging them into the required positions. The second page of the Posture Creator serves for editing constraints and angles between the different joints, as shown in figure 4(a). Constraints for postures helps distinguishing different postures from each other. A constraint is defined by selecting a base joint, displayed in red, and two leg joints, displayed in green. Base and leg joints form an angle. The constraint restricts this angle between the two leg and base joints and has a given tolerance, which translates into a minimum and maximum angle. The posture is then saved into an XML file, which can be loaded again at any time. The name given for the file is also the name for the posture. Using this name, the postures can be registered in the `Prototype` class. All registered and therefore available postures are then displayed in the “Posture” tab, where also the name of the last recognized posture is given, as shown in figure 4(b).



Use the left mouse-button to select two leg-joints (green) and the right mouse-button for the base-joint (red). The base-joint should lie between two leg-joints. You can deselect a joint by reclick it. We recommend to set a sufficient tolerance for a better recognition. The tolerance should not been changed during the adding process.

(a) Creating postures using the Posture Creator.



Posture Name	
WPosture	
BothHandsToHeadPosture	
TPosture	
BothArmsUpPosture	
BothHandsTogetherPosture	
XPosture	

latest recognized gesture:
BothHandsTogetherPosture

```
Someone said hallo
start posture of Player 570 for gesture: ZoomInGesture recognize
Someone said hallo
Someone said hallo
Collision with Button Kugel2 detected
Posture of player 526 recognized: BothHandsTogetherPosture
Collision with Button Kugel2 detected
Collision with Button Kugel2 detected
```

(b) Posture tab.

Fig. 4. Posture component: creating and customizing

The `SurfaceComponent` is used to create **interactive surfaces** (any polygons) or volumes (e.g. spheres) defined by their coordinates in space. Interactions are triggered by collisions of a body-joint with the defined surfaces using the Kinect depth sensor. A developer can use interactive surfaces to simulate buttons or interaction with fixed objects. The surfaces can be designed “around” the objects that are meant to be interactive. By touching these real-world objects and consequently interacting with the defined surface, the interaction with the object itself can be emulated. Interactive surfaces can be created by directly drawing them into the depth frame of the Kinect sensor, that is displayed on the left in the GUI. There are two drawing modes - a triangle mode that allows for creation of any kind of polygons, which are automatically split up into triangles. While in triangle mode, the user can click on the “Create New Surface” button and then mark the vertices of the polygon in the depth frame. A mouse click defines a vertex at the current x and y position. The depth information is taken directly from the depth information of the sensor. A right mouse click completes the definition of a new interactive polygon. The second mode is the sphere mode. By clicking into the depth frame, the center of a new interactive sphere is marked. A prompt allows to input the radius of the sphere. Each new interactive surface or sphere is given a name, that is also used to reference it in the underlying code. As shown in figure 3(a), the tab also shows a front view and a top view of the space that is captured by the Kinect sensor. The interactive surfaces are displayed there, in order to give a three dimensional impression. The detected skeletons can also be displayed in the front and top view. Interactive surfaces can be exported and saved to a file for reuse and import in other prototypes.

The detection of **gestures** is implemented in the `GestureComponent`. They are not so easy to define via graphical interfaces. Gestures consist of different postures that are performed in sequence. Variations in speed of the executed gesture as well as angles and movement during the transition between postures affect the precision of gesture recognition. The given tolerances have a strong influence on the precision of recognition and they differ for each gesture. Therefore, we decided to implement different gestures that are provided by ProtUbique. The gesture library can be extended by programming additional gesture recognition components and we hope that the future use of ProtUbique leads to the development of components that can be added to the library. Prototyping engineers that do not want to implement new gestures can use the predefined gestures from our library. Using the gesture tab of the GUI (3(b)), all registered gestures can be selected and a picture of the postures that form the start and end of one gesture is displayed as well as a textual description of the gesture.

By using the Windows Desktop **Speech** API from Microsoft, we also realized a `SpeechComponent`. Words and phrases can be defined as commands. The `SpeechComponent` encapsulates the Windows Desktop Speech API and passes the events of recognized commands on to the `Prototype` class. A prototype engineer can add new words and phrases as commands by entering them on the speech tab of our ProtUbique GUI. The last word that was recognized is also displayed there for logging purposes.

To **evaluate** the prototypes created with our system, we additionally integrated a built-in recording-tool which is implemented as a specialized component, called `RecorderComponent`. It uses a camera and microphone to record audio and video data. In addition, the interactions that are recognized and fired events are also logged. In order to study the performance of test users, the prototyping or usability engineer can replay the recorded audio and video and also review the performed interactions. On the "Recorder" tab of the GUI it is possible to insert an ID to identify the current test user. This ID is then used to associate the recordings with the performed tests.

3.3 A Practical Example of How to Develop a Prototype Using ProtUbique

To test the functionality of ProtUbique, two developers, who were unfamiliar with our framework, developed a new prototype for a ubiquitous music player. To implement a new prototype there are only a few steps necessary. At first, a new ProtUbique project has to be created. It contains a `Main` class that initializes the different interaction components and instantiates the `Prototype` object. By instantiating `Prototype`, the prototype engineer can choose if the GUI, the Kinect sensor and the evaluation recorder should be enabled. The chosen interaction components are then added to the `Prototype` object. Once registered, the interaction components can be configured using the GUI, as described above. Afterwards, the events that are generated by ProtUbique and the used interaction components can be plugged into backend code, in order to trigger responses.

The music player's functions comprise a central storage of different types of music and the possibility to play this music. The player is controlled by gestures, surfaces, postures, and speech. The following functions were implemented by the developers: Associate different categories of music with objects through interactive surfaces, start and stop music, adjust volume up and down, forward and rewind, start and stop karaoke mode. The developers worked as a team because one person had to assemble the prototype and to program the backend code, while another person was necessary to test the behaviour of the implemented prototype and also to discuss their concept for the protoype. The developers were asked to assess their programming skills and experience with Microsoft Kinect and the Kinect SDK before starting the implementation. Both stated that they have middle-rated experience with the Microsoft Kinect sensor as interaction device and a lot of experience in programming in general. They rated their C# skills as little to moderate.This information was gathered to figure out whether users have to be experts for using the prototyping framework or not. It turned out that C# knowledge would be an asset but is not mandatory. This is actually due to the fact that the backend code and the processing of events proved difficult, given their moderate expertise using C#. The test programmers stated, that the ProtUbique framework was helpful implementing a prototype for a ubiquitous system. They also judged that ubiquitous prototyping supported by a framework is easy and fast in contrast to complex constructions that they figured would be needed if no framework is provided. Different interaction options are easy to integrate and development cycles are fast, too. Both developers evaluted the framework as usable, operable and easy to learn. In addition, the functions of the offered components were comprehensible. For applying gestures and postures they used the framework's GUI, which was considered suitable. Some suggestions for improvement were given by the developers, too: the realization of different grammars for `SpeechComponent`, allocation of templates, pre-assembled elements and patterns within the framework's GUI to speed up the prototyping and the support of composition by drag-and-drop. As our example shows, our modular concept seems to work and the facilitation of the prototyping process was highly appreciated.

4 Conclusions

In this paper we presented the ProtUbique framework, that facilitates rapid prototyping and evaluation of interactive ubiquitous systems. Since interactive ubiquitous systems come with innovative and for most users unfamiliar interaction concepts, they have to be designed involving users to ensure high usability. Therefore the ProtUbique framework is conceived to support the user-centered design process of ubiquitous systems.

It is extensible so that additional interaction channels can be easily implemented. With this functionality, rapid prototyping of ubiquitous interactive systems becomes easily feasible. A prototyping engineer however, can use the given interaction channels with minimal programming effort. He therefore can focus on providing detailed backend code or on conducting user studies and evaluating his ideas. For this purpose we provide a GUI that can be used to configure the different current implemented interaction channels as far as possible.

Our concept is not fully implemented yet, since there are no dedicated output components at the moment, e.g. a speech output channel. This is part of our future work on ProtUbique. In another iteration, we plan to further extend the available input interaction channels and also to implement new gestures by adding for example the Gesture Authoring Tool of Omek Beckon SDK[1]. So far, implementation of new interaction channels has proven to be easy and quick, due to the modular design. As interaction channels are separated from each other and implemented as individual components, the development of new components is straightforward. We would like to test the next

[1] `http://www.omekinteractive.com/`

version of ProtUbique with some participants, in order to evaluate the usability of the framework. The vision of our work is to evolve the ProtUbique framework from a tool-based rapid prototyping framework for ubiquitous systems into an integrated user interface and software engineering approach for multimodal ubiquitous systems, spanning the whole life-cycle of highly interactive ubiquitous systems.

Acknowledgements. We wish to thank Enrico Hinz, Josefine Zeipelt, Mirko Wolff, Nico Schertler, Philipp Sonnefeld and Ronald Graupner for the contributions made by their practical work.

References

1. Ballagas, R., Ringel, M., Stone, M., Borchers, J.: istuff: A physical user interface toolkit for ubiquitous computing environments. In: CHI 2003, pp. 537–544 (2003)
2. Bäumer, D., Bischofberger, W.R., Lichter, H., Züllighoven, H.: User interface prototyping - concepts, tools, and experience. In: Proceedings of the 18th International Conference on Software Engineering, ICSE 1996, pp. 532–541. IEEE Computer Society, Washington, DC (1996)
3. Dey, A.K., Abowd, G.D., Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction 16, 37–41 (2009)
4. Gould, J.D.: How to design usable systems. In: Handbook of Human-Computer Interaction (1988)
5. Kühn, R., Keller, C., Schlegel, T.: Von modellbasierten storyboards zu kontextsensitiven interaction-cases. i-com - Zeitschrift für Interaktive und Kooperative Medien 10(3), 12–18 (2011) (in German)
6. Liu, L., Khooshabeh, P.: Paper or interactive?: a study of prototyping techniques for ubiquitous computing environments. In: CHI 2003 Extended Abstracts on Human Factors in Computing Systems, CHI EA 2003, pp. 1030–1031. ACM, New York (2003)
7. Luqi: Software evolution through rapid prototyping. IEEE Computer 22, 13–25 (1989)
8. Maguire, M.: Methods to support human-centred design. International Journal of Human-Computer Studies 55(4), 587–634 (2001)
9. Marquardt, N., Diaz-Marino, R., Boring, S., Greenberg, S.: The proximity toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies. In: UIST 2011 (2011)
10. Norrie, L., Murray-Smith, R.: Virtual sensors: Rapid prototyping of ubiquitous interaction with a mobile phone and a kinect. In: MobileHCI 2011 (2011)
11. Sandor, C., Klinker, G.: A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality. Personal Ubiquitous Comput. 9, 169–185 (2005)
12. Weiser, M.: The computer for the 21st century. Scientific American 265, 94–104 (1991)
13. Weiser, M.: Some computer science issues in ubiquitous computing. Communications of the ACM 36(7), 75–84 (1993)