A Framework for Rapid Prototyping of Multimodal Interaction Concepts

Ronny Seiger
Technische Universität Dresden
Dresden, Germany
ronny.seiger@tu-dresden.de

Florian Niebling
Technische Universität Dresden
Dresden, Germany
florian.niebling@tu-dresden.de

Mandy Korzetz
Technische Universität Dresden
Dresden, Germany
mandy.korzetz@tu-dresden.de

Tobias Nicolai
Technische Universität Dresden
Dresden, Germany
tobias.nicolai@mailbox.tu-dresden.de

Thomas Schlegel
Technische Universität Dresden
Dresden, Germany
thomas.schlegel@tu-dresden.de

ABSTRACT

Ubiquitous systems provide users with various possibilities of interacting with applications and components using different modalities and devices. To offer the most appropriate mode of interaction in a given context, various types of sensors are combined to create interactive applications. Thus, the need for integrated development and evaluation of suitable interaction concepts for ubiquitous systems increases. Creation of prototypes for interactions is a complex and time consuming part of iterative software engineering processes, currently not well supported by tools as prototypes are considered to be short-living software artifacts. In this paper, we introduce the framework *Connect* that enables rapid prototyping of interaction concepts with a focus on software engineering aspects. The framework allows the definition and modification of event-to-action mappings for arbitrary interaction devices and applications. By applying *Connect*, model-based prototypes of multimodal interaction concepts involving multiple devices can be created, evaluated and refined during the entire engineering process.

ACM Classification Keywords

H.5.2 User Interfaces: User-centered design

Author Keywords

interaction framework; interaction concepts; multimodal interaction; rapid prototyping; software engineering

INTRODUCTION

Ubiquitous Systems as coined by Weiser [22] describe user-centered systems at the intersection of mobile and pervasive computing combined with ambient intelligence. In addition to the high degree of embeddedness of pervasive systems, ubiquitous systems are characterized by a high level of mobility, often consisting of a large number of heterogeneous and possibly resource-limited devices, which are loosely-coupled into dynamic network infrastructures. The emergence of smart spaces (smart homes, smart factories, smart offices) shows the increasing importance and spreading of ubiquitous systems throughout different areas of everyday life.

As user interaction with devices that disappear into the background often cannot be realized using traditional metaphors, new ways of interaction have to be explored during the software engineering process for creating ubiquitous system components. The development and improvement of multimodal interaction concepts (i. e., interactions using one or more input modalities [21]) is thereby not limited to initial prototyping, but equally important during implementation, testing and evaluation stages.

Existing tools for interaction design and rapid prototyping of ubiquitous user interaction can be successfully employed during the initial prototyping phases. In later development phases as well as in iterative engineering processes such as user-centered design (UCD), their applicability is often restricted due to their limited ability to automatically propagate changes in prototypes to subsequent stages of development. The mostly informal descriptions and implementations of interaction concepts and interactive applications limit their extensibility with respect to new interaction devices and modalities. The lack of models and formalism also prevents prototypes for interactions from being used and matured in later stages of the development process, which is why prototypes are usually considered to be short-living software artifacts.

In this paper, we propose a model-driven framework for prototyping of interaction concepts that can be applied throughout the different phases of iterative software engineering processes. The focus of the introduced *Connect* framework for interaction design is placed on software engineering aspects

Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.
Copyright © 2015 for the individual papers by the papers' authors. Copying permitted only for private and academic purposes. This volume is published and copyrighted by its editors.

and models. It enables the rapid development of prototypes and enhancement at runtime during expert evaluation and user studies. Extensibility concerning new types of interaction devices as well as interactive components is easily achieved by inheritance in the object-oriented framework architecture. As a result of a high-level, model-based design of interaction concepts, modifications to the interactions–even of very early prototypes–can be reused and advanced from beginning to end of the development cycle. The framework supports individualizations concerning different groups of users as well as distinct scenarios by customizing interaction concepts using specialization and configuration. We introduce a tool based on the *Connect* framework that facilitates the creation and customization of interaction concepts at runtime even by non-programmers. The framework is demonstrated developing an interaction prototype within a Smart Home–a ubiquitous environment consisting of various devices for multimodal interactions with physical and virtual entities.

RELATED WORK

Prototypes are useful tools for assessing design decisions and concepts in early but also in advanced stages of the software engineering process. Especially in iterative design processes for creating usable systems, future users have to be involved continuously to provide feedback and to improve concepts and software artifacts [9]. According to Weiser, one of the essential research and development methods for interactive ubiquitous systems is the creation of prototypes [23]. The rapid prototyping technique aims at creating prototypes in a time and resource efficient way to mature artifacts in agile software engineering processes [15]. The focus of our work is on providing a framework for the rapid development and evaluation of multimodal interactions for ubiquitous systems. Especially within the UCD process, prototypes are needed to evaluate design ideas and improve the usability of interactive systems [17]. Complex interaction scenarios involving multimodal interactions require the use of technically mature prototypes to improve the usability of the system or application [14].

The basis for our prototyping framework is the *OpenInterface* platform developed by Lawson et al. [13]. The platform allows the integration of arbitrary heterogeneous components as interaction devices and applications. OpenInterface provides a lightweight framework and runtime environment leveraging the prototyping of multimodal interaction concepts using high-level data fusion and pipeline concepts to connect interaction devices with interactive applications. In contrast to other frameworks for prototyping of interactions [7, 4], OpenInterface is based on a platform and technology independent description of a component's functionality and interface. Our framework adapts these concepts with a stronger focus on the underlying models and component descriptions in order to facilitate the extension, runtime adaptation and reuse of components and interactions during the iterative stages of UCD. Other existing interaction and prototyping frameworks (e. g., *CrossWeaver* [20] and *STARS* [16]) are realized in a more informal way for specific scenarios and therefore lack extensibility of software components, interaction devices and modalities as well as reusability. The OpenInterface workbench [13] provides developers with a comprehensive toolset for configuring components and interactions. Our aim is to provide an easy to use tool also enabling non-programmers to rapidly create interaction prototypes.

In [10] Hoste et al. describe the *Mudra* framework for fusing events from multiple interaction devices in order to leverage multimodal interactions. Mudra provides means for processing low-level sensor events and for inferring high-level semantic interaction events based on a formal declarative language. The framework is primarily focused on combining multiple interaction devices to enable advanced forms of interaction input. It can be used complementary to the *Connect* framework as part of defining and integrating low-level and high-level sensor components. However, the application of the formalism and semantic models that Mudra is based on increases the effort and complexity for rapidly prototyping interaction concepts and introduces a number of additional components to the lightweight *Connect* framework.

In addition to the work of Dey et al. [6], one of the first conceptual frameworks for the rapid prototyping of context-aware applications predominant in ubiquitous systems, the *iStuff* toolkit [3] and its mobile extension by Ballagas et al. [2] represent further related research our *Connect* framework is based on. These toolkits offer a system for connecting interactive physical devices and interaction sensors with digital applications and software components. The iStuff toolkit suite supports multiple users, devices and applications as well as interaction modalities. However, due to the limited software models applied within these tools, the set of supported interaction devices is rather static. A model-based approach for dynamically creating multimodal user interfaces composed of several components is described by Feuerstack and Pizzolato [8] as part of their *MINT* framework. Mappings between user interface elements, user actions and actions to be executed can be formally defined with help of this framework and used for dynamically generating interactive applications. Both the iStuff and MINT framework are intended to be used in the design and development process of user interfaces whereas our focus lies on the prototyping and evaluation of interaction concepts (i. e., event-to-action mappings) in different stages of the UCD process. However, in order to prototype and develop interactive applications including interaction concepts and user interfaces, the iStuff and *Connect* framework can be used complementary.

The *ProtUbique* framework by Keller et al. facilitates the rapid prototyping of interactive ubiquitous systems [11]. It supports an extensible set of interaction channels transmitting low-level interaction events from heterogeneous devices and sensors. These interaction events are unified by the framework and accessible programmatically in order to prototype interactive applications. As ProtUbique offers interfaces to access its interaction channels, it is possible to directly combine both the ProtUbique and *Connect* framework. Interaction channels are integrated into *Connect* in the form of sensors supplying interaction events. Connect can then be used to map these events to actions that will be executed by actuators or applications.

With the emergence of ubiquitous systems, users play a central role in the software development process. Prototypes are well suited for involving users in the design process and for improving concepts and software artifacts based on user feedback. Various frameworks for the prototyping of interactive applications including user interfaces and interaction concepts exist. These frameworks are often focused on the use of prototyping techniques in early development stages and limited in the set of supported software components and interaction modalities. However, agile and iterative software engineering processes are required for developing interactive ubiquitous systems. Therefore, we propose a model-driven framework for the rapid prototyping of multimodal interaction concepts. By applying models for the definition of interactive components and their interrelations, extensibility and reusability of interaction concepts and interactive prototypes in multiple design stages is facilitated. In that way, the usability and user experience of applications and systems for ubiquitous environments can be increased.

INTERACTION FRAMEWORK

Structure

We designed the *Connect* framework from a software engineering point of view using abstract models and their building blocks as a starting point. The framework adheres to a basic class structure consisting of multiple types of components, which are interconnected with each other. Fig. 1 shows the class diagram using UML notation. A *Component* is a software entity having a defined interface and known behavior [13]. In analogy to control systems linking physical sensors with actuators, we distinguish between *SensorComponents* representing entities that are able to produce interaction events and *ActuatorComponents* able to consume interaction events and trigger subsequent actions. *ComplexComponents* combine these capabilities. In addition, specializations of complex components are used to enable the logical and temporal combination of sensor and actuator components. Ports describe the components' interfaces in order to define interactions and connections between multiple components. *EventPorts* define the types of events a sensor component is able to produce and *ActionPorts* represent the types of actions an actuator component is able to perform. The activation of an event port leads to the activation of the action ports the event port is connected to. A central Manager class handles the instantiation of components and maintains a list of all active component instances, which are accessible from within the scope of the framework.

Sensor Components

Sensor components represent devices and applications that are able to detect interactions and produce corresponding interaction events. The *SensorComponent* is an interface sensors have to implement, e. g., by an adapter connecting the sensor device via its API to the *Connect* framework. An *EventPort* is a wrapper for every type of event the sensor component is able to trigger. The sensor component maintains a list of all its events and creates corresponding event ports. By implementing the sensor component interface, new types of

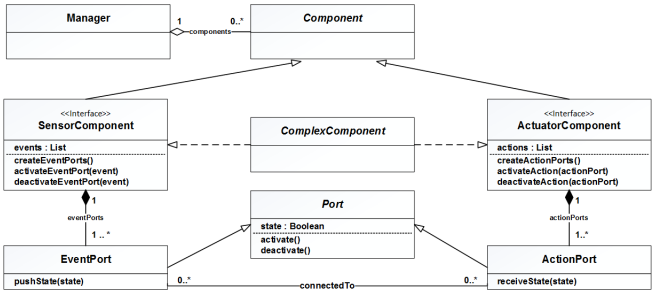

Figure 1. Class diagram of the *Connect* framework

interaction devices and arbitrary event sources can be integrated into the framework. In order to integrate new types and corresponding instances of sensor components into the framework, an adapter for receiving the sensors' events has to be implemented. On receiving an event from the sensor, the state of the corresponding event port is updated, i. e., the port is activated or deactivated. An event port can be connected to one or more action ports of one or more actuator components. The event port's activation leads to the activation of the connected action ports. As arbitrary event sources are supported, interactive devices independent of modality and number can be combined to be used for multimodal interactions, i. e., using one or more input modalities. Currently, only binary states for events (active/inactive) without additional data payload are supported by the *Connect* framework.

An example of a locally integrated sensor component is the computer's keyboard. The event ports correspond to the set of the individual keys. A smartphone device sending touch events via a dedicated app to an instance of the *Connect* runtime is an example of a remotely integrated sensor component. The set of touch events provided by the smartphone and supported by the app represent the event ports.

Actuator Components

Actuator components represent devices and applications that are able to actively perform and execute actions. Analogous to a sensor component, the *ActuatorComponent* is an interface actuators have to implement in order to connect the actuator to the *Connect* framework. An *ActionPort* is a wrapper for an action or method the actuator component is able to execute. New types of actuator components can be integrated into the framework as implementations of the interface *ActuatorComponent*. Adapters for calling the actuator components' particular operations from inside the framework have to be implemented for every type of actuator. An action port can be connected to one or more event ports of one or more sensor components. Upon receiving an activation from an event port connected to an action port, the actuator component activates the action port and executes the corresponding method. Arbitrary local and remote devices and applications can be integrated into the framework as actuator components. Currently, we support the activation of methods without the processing of input or output parameters.

An example of an actuator component is a service robot whose movement functionality is provided in the form of

directed movement actions (e. g., forward, backward, left, right). For each direction there is a corresponding action port.

Complex Components

Complex components represent devices and applications that combine sensor and actuator functionalities. These entities can contain multiple event and action ports, i. e., they are able to produce events for actuator components and receive events from sensor components. The *ComplexComponent* class is viewed as an abstract class that has to implement both the *SensorComponent* and the *ActuatorComponent* interfaces in order to be integrated into the *Connect* framework.

An example of a complex component is a smartphone sending touch interaction events and providing executable operations (e. g., for taking pictures or switching on the its light).

Logical and Temporal Components

Logical Components are viewed as specializations of complex components. They are used for creating logical connections (AND, OR, NOT, XOR, etc.) between multiple event ports of one or more sensor components. The logical component's action ports are used as input ports for events from the sensor components and its event ports are used as output ports producing events for the activation of subsequent actuator components. By cascading these logical components, complex logical circuits for sensor events triggering actions of actuator components can be created. In addition, we integrate flip-flop and trigger components for saving of and switching between states. That way, it is possible to define more complex interaction concepts involving multiple interaction devices in advanced stages of the engineering process and also to introduce modes of interaction (i. e., state-dependent behavior of the interactive prototypes).

Besides logical components, *Temporal Components* for describing temporal dependencies between sensor events are supported as extensions of complex components. That way we are able to define the activation of higher level events, e. g., after a defined number of repeating sensor events or after the appearance of an event within a defined time frame. The functions and algorithms–including additional attributes–that are executed when the logical or temporal component is activated have to be provided for each new type of complex component. As new types of components are introduced into the framework's underlying class model by inheritance, only the base classes' methods have to be overwritten to use instances of these new components.

Dynamic Components

Thus far we are able to extend the set of sensor, actuator and complex components by introducing implementations and specializations of the appropriate classes into the model at design time. In order to add new types of components at runtime, we extend the framework by the concept of *Dynamic Components*. These components are created by *Connect*'s runtime based on a formal model of a component's functionality and ports. Currently, we support the use of a WSDL (Web Services Description Language [5]) document describing the available operations of a service-based actuator component. The WSDL format provides a suitable formalization

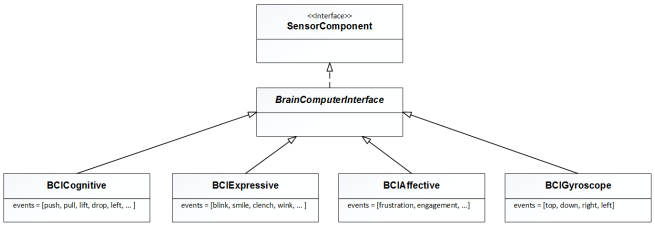

Figure 2. Extensions of the *SensorComponent* to support BCI input modes

of an actuator's callable methods and their parameters, which can be parsed in order to automatically create an actuator component implementing the *ActuatorComponent* interface and the corresponding action ports.

PROTOTYPING MULTIMODAL INTERACTION CONCEPTS

Exemplary Sensors

Brain Computer Interface

In order to show the framework's capability of supporting multimodal interactions and its applicability within the Smart Home scenario, we extended the core sensor component by the Emotiv EPOC[1] EEG brain computer interface (BCI) acting as a source of interaction events [19]. The BCI used in our setting provides interaction modes enabling the recognition of thoughts (*Cognitive*), facial expressions (*Expressive*), emotions (*Affective*) and head movement (*Gyroscope*). Each of these modes is introduced as a subclass of the abstract *BrainComputerInterface* class, which implements the *SensorComponent* interface (see Fig. 2). Event ports are created for every possible type of sensor event produced by the BCI in each mode (e. g., for blink, wink left, look right, smile, and laugh in the Expressive mode). Upon instantiation of an object of one of the interaction mode classes, a listener for event ports corresponding to the sensor events is initialized.

Tablet

The second exemplary sensor component from the Smart Home domain that we integrated into our test setting is an Android-based tablet device. A dedicated app sends interaction events regarding the pressing of specific buttons and events detected by the tablet's gyroscope sensor to an instance of the *Connect* framework. In order to support this event source, we introduce the abstract *Tablet* class implementing the *SensorComponent* interface. From that class, the *Button* and *Gyroscope* modes are derived as subclasses (see Fig. 3). Event ports representing the particular buttons and gyroscope movement directions (i. .e., forward, backward, left, right) enable the detection of the corresponding interaction events and connection to other components.

Exemplary Actuators

Service Robot

A TurtleBot 2[2] service robot plays the role of an actuator in the context of our Smart Home scenario [19]. We abstracted its movement functionality into two operational modes extending the abstract *ServiceRobot* class: *Manual Movement*

[1] https://emotiv.com/epoc.php
[2] http://www.turtlebot.com/

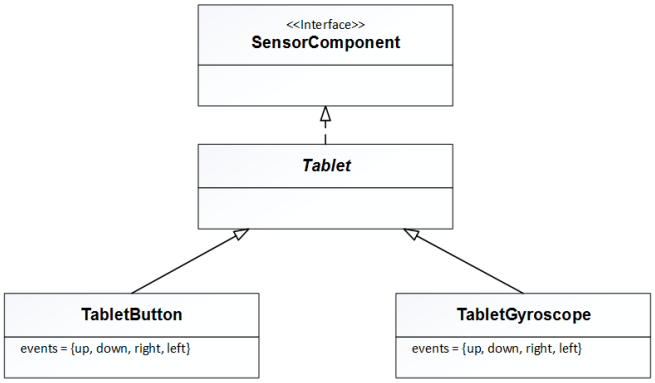

Figure 3. Extensions of the *SensorComponent* to support tablet input modes

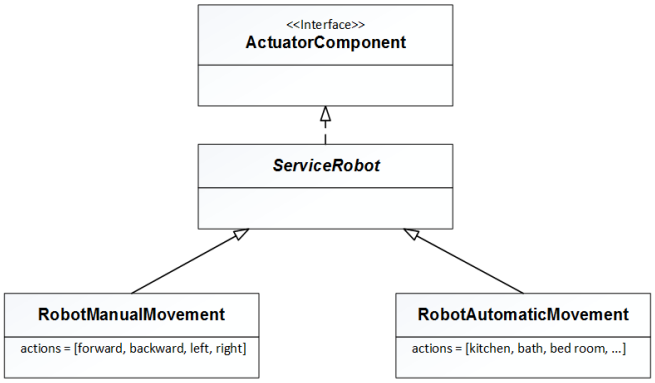

Figure 4. Extension of the *ActuatorComponent* to support a service robot actuator

and *Automatic Movement* (see Fig. 4). The manual movement mode supports the fine-grained control of the robot platform by direct movement commands (i. e., forwards, backwards, to the left, to the right). Using the automatic movement mode, the robot can be send to specific locations in a room or building. In automatic mode, driving, path planning and obstacle avoidance are handled by the robot itself. The action ports for these two actuator component modes correspond to the available movement directions (manual mode) and to the specific target locations (automatic mode).

Service-based Light Switch

The capability of dynamically adding new components at runtime is an important feature of the *Connect* framework. As it supports the automated generation of an actuator component based on a WSDL document, we implemented a web service for the remote control of a light switch providing a *switch on* and a *switch off* operation. Upon parsing of the WSDL file and creation of action ports for both operations, the *Connect* runtime acts as a client sending requests to the web service.

Security Component

In order to prevent incorrect behavior and actions caused by imprecise interaction devices and unintended user interactions at runtime, a *Security Component* is introduced into the framework as an implementation of the actuator component. This component provides an operation for deactivating all event and action ports and thereby disabling the current

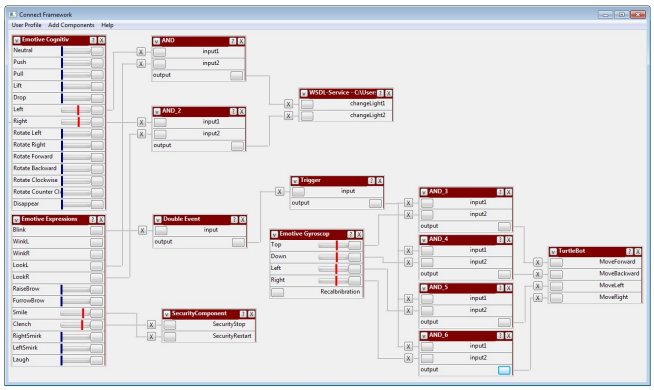

Figure 5. Complex network of input and output components forming an interactive prototype

interactions and listeners for new events. The security component's second operation resets all ports to the inactive state and re-enables the event listeners to continue with the interaction. Both operations can be connected to the event ports of an arbitrary–preferably reliable–sensor component.

Prototyping Tool

We implemented a Java application based on the *Connect* framework. The tool allows the graphical instantiation of known types of sensor, actuator and complex components. The lists of available ports, individual attributes as well as the component's graphical representation are coded into the class structure and component's data model. Instantiated components can be configured using the tool. In addition, it is possible to generate service-based actuator components from a WSDL file. Connections between component ports are created and modified graphically at runtime using drag and drop gestures (cf. Pipeline metaphor [13]). That way, circuits for interactions consisting of sensors, actuators, logical components and temporal components can bes designed. For certain types of sensor events there are sliders that are used for setting activation thresholds. As many interaction devices provide sensor data in the form of numerical values–not just Boolean values for the active/inactive states–the definition of activation thresholds increases the accuracy of event detection/activation and supports individual user configurations.

Fig. 5 shows a screenshot of the configuration tool's user interface containing three instances of sensor components (BCI modes), logical components and an actuator component (service robot). The tool's user interface provides visual feedback regarding currently active sensor events, connections, and actions as well as numerical values for sensor input.

Prototype Configurations and User Profiles

The composition of components as well as their interconnections, attributes and port thresholds can be persisted in individual prototype configurations and user profiles based on the class model presented in the previous section. These settings are saved in and loaded from XML-based files. In this way, individual interaction concepts can be created for specific prototypes, component configurations, scenarios and

users according to their capabilities. These model-based configurations can then be used as templates for creating new interaction concepts or for refinement at a later stage of the development process [18]. Listing 1 shows an extract of a prototype configuration describing an event port of a sensor component connected to an action port.

Listing 1. Extract from a prototype configuration

```
<conf>
 <component>
  <class>IO.TabletButtonComponent</class>
  <id>TabletButtonComponent</id>
   <ports>
    <port>
          <class>Core.EventPort</class>
          <id>Up</id>
          <state>false</state>
          <connectedPorts>
           <connectedPort>
            <componentId>TurtleBot</componentId>
            <portId>MoveForward</portId>
           </connectedPort>
          </connectedPorts>
         </port>
   </ports>
  </component>
 <component>
  ...
</conf>
```

Prototyping and Evaluation

With the help of the *Connect* framework and the configuration tool, interaction concepts describing mappings between interaction devices and active controllable components can be created. Once integrated into the framework, multiple instances of multimodal sensor, actuator and complex components are ready to be used and loosely-coupled at runtime for a particular setting. Connections between sensors and actuators and their respective ports are modifiable at runtime (create, update, delete). Compared to hard-wired event-to-action mappings, model-based prototypes of interaction concepts can be created and modified quickly with the help of *Connect* in order to test and evaluate their usability and suitability for concrete use cases as part of the software engineering process.

Connect can be part of various user-centered prototyping and evaluation methods and stages reusing models that describe components and their interrelations. As the prototyping tool follows known metaphors from the WIMP paradigm and integrates easy to understand graphical metaphors (e. g., pipelines and circuits), it is also possible for designers, non-programmers and end-users to understand and define interaction concepts for a given scenario. That way, future users can be involved in early stages of the design process leading to more intuitive interactions and usable applications for ubiquitous systems.

Due to the model-driven approach for describing components and their interrelations applied in *Connect*, it is possible to persist and reload component configurations and their connections. Prototypes of interaction concepts can be repeatedly tested, reused and refined in order to increase the usability of the interactive application that is under development. By creating user profiles for specific users, user groups and scenarios, the corresponding prototypes can be used for user-centered evaluation methods (e. g., user studies and expert evaluation) during various stages of the software development process. In addition, user profiles facilitate the creation of interactive applications according to a user's individual cognitive capabilities and preferences.

The extensibility of the framework's underlying models for components and interactions allows for the integration of new interactive devices and applications at later engineering stages. That way, interaction concepts can be developed starting from simple event-to-action mappings and evolved to more complex models and scenarios for interactive ubiquitous systems. Changes within these models can be propagated to the affected software artifacts. By adding a mechanism for model versioning, traceability for the evolution of these interaction models in iterative stages of the design process is achieved and templates of interaction concepts can be created for reuse in new projects.

DISCUSSION

The introduced *Connect* framework for the creation and iterative advancement of interaction concepts is built on model-based components and configurations. Our chosen design allows for convenient extension through the inclusion of additional arbitrary interaction components (i. e., sensors and actuators) via concepts of object-orientation. Basic design patterns reduce the effort for developers to extend the set available interaction devices and applications. At runtime, interaction components can be rapidly combined into pipelines and circuits to form complex interaction concepts. Modifications of interaction concepts as well as persisting and loading of interaction and component configurations are supported by the framework. Compared to related research, this approach allows for a high level of reusability and refinement of interactive prototype applications and configuration in succeeding development stages. With the help of the graphical configuration tool it is possible to rapidly create working prototypes for multimodal interactive applications and use these prototypes for test and evaluation purposes.

Due to the use of known metaphors, e. g., pipelines and circuits, the creation and configuration of complex interaction concepts can be achieved even by non-programmers such as interaction designers or end users, supporting different phases of iterative software engineering processes like UCD. By providing model-based abstractions for component design and data flow, interaction design may be advanced from simple prototypes during requirement analysis to complex multimodal interaction concepts containing numerous different sensors and actuators in subsequent phases of development. This iterative improvement of existing concepts proved to be especially helpful during usability testing using expert evaluation and end user studies.

On the one hand, the introduction of logical and temporal components into more complex interaction concepts allows for the use of multiple interactive input devices. On the other hand, these components enable interaction modes

and the definition of interaction sequences to prevent unintended interaction events. These mechanisms become necessary when using imprecise and "always on" interaction devices (e. g., brain-computer interfaces and eye trackers) to prevent Midas touch. With respect to the BCI, a double-blink within a certain timeframe could, for example, be used to trigger an action instead of a simple "natural" blink. As shown in the prototyping tool, thresholds defining the activation of sensor ports can be defined for sensor events containing Integer or Double values. This mechanism also helps interaction designers with the integration of imprecise devices. Lastly, a security component can be added to the interaction concept and test environment to stop all ongoing interactions in case of any malfunctions. This is especially helpful when interacting with real world physical devices in ubiquitous systems.

The *Connect* framework is a prototyping tool for engineering interactive applications. It can be employed during the development process for designing and testing of interaction concepts. In combination with other frameworks for the prototyping of user interfaces (e. g., MINT [8], iStuff [2]), interactions (e. g., OpenInterface [13]) and interaction devices (e. g., ProtUbique [11]) as well as for the fusion of high-level sensor events (e. g., Mudra [10]), *Connect* represents an addition to the engineering toolchain for interactive ubiquitous systems. Due to its extensibility and model-driven development approach, interaction and configuration models created with *Connect* could be used as input for subsequent tools and phases in the software engineering process.

The current development state of the *Connect* framework still contains some shortcomings that will be improved in consecutive versions. Until now, user defined data types beyond simple Boolean values at event and action ports are not supported. To be able to accommodate analogue sensors, at least some form of floating point data and complex data types will have to be included into the extensible data model. In addition, aggregation of components attached to distributed instances of the framework is not yet possible. With the availability of active components that contain significant processing power themselves such as smartphones, integration of preprocessed sensor values can be simplified by combining multiple networked *Connect* systems.

CONCLUSION & FUTURE WORK

Engineering software for interactive ubiquitous systems requires flexible and iterative development processes. The continuous involvement of future users throughout the entire process is a key aspect of the user-centered design and development methodology for ubiquitous systems. Prototypes are a well suited tool for the development, test and evaluation of theoretical concepts in almost all stages of the software engineering process. We developed an extensible and easy to use framework that supports rapid prototyping, evolution and evaluation of interaction concepts for ubiquitous systems. The *Connect* framework is based on a modular object-oriented software model, which views interaction devices as sensor components and interactive applications as actuator components. Interaction concepts can be defined, modified and tested at runtime by connecting these components. The framework follows a model-driven software engineering approach enabling the extension and integration of new types of components into interactive prototypes as well as the reuse of component and prototype configurations during development. Related frameworks and concepts generally lack extensibility, flexibility and reusability due to the limited use of models and other formalisms. Therefore, interaction frameworks often support only a static set of interaction devices and applications that can be used for the development of short-living interaction prototypes at early design stages. With *Connect*, the set of software components can be easily extended to support new types of devices and applications, which can be combined to create multimodal interactions. *Connect*'s runtime and user-friendly prototyping tool facilitate the use of multiple input and output devices as entities involved in interactions as well as dynamic reconfiguration of interactions at runtime. The model-based descriptions of components and interactions leverage the reuse and iterative refinement of components, concepts and prototypes for the user-centered software engineering process of ubiquitous systems.

Regarding future work, we will extend the component models in order to be able to process non-Boolean input and output values and states for event and action ports. The use of a formal definition language for describing the interfaces of a sensor component will add the ability to also introduce new types of sensor components to *Connect* at runtime. By combining the prototyping framework *ProtUbique* [11] for defining interaction sources and high-level interaction events with service-based communication for sending interaction events to *Connect*, we will create a flexible toolchain for developing prototypes of interactive multimodal applications. In addition, dynamic component platforms (e. g., OSGi [1]) can be employed to introduce additional runtime flexibility concerning the support of new types of software components. We will also look into the distributed communication among several instances of the *Connect* runtime. An instance of *Connect* running on one computer could be used as a sensor component within another *Connect* instance, which allows the preprocessing and derivation of higher order events on local computers in order to save resources and simplify the modeling of complex interactions. To evaluate the framework's applicability, we will use *Connect* for the prototyping of interactive software components as part of the engineering process for Smart Home applications [19, 12].

REFERENCES

1. OSGi Alliance. 2003. *Osgi service platform, release 3*. IOS Press, Inc.

2. Rafael Ballagas, Faraz Memon, Rene Reiners, and Jan Borchers. 2007. iStuff mobile: rapidly prototyping new mobile phone interfaces for ubiquitous computing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 1107–1116.

3. Rafael Ballagas, Meredith Ringel, Maureen Stone, and Jan Borchers. 2003. iStuff: a physical user interface toolkit for ubiquitous computing environments. In

Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 537–544.

4. Jullien Bouchet and Laurence Nigay. 2004. ICARE: a component-based approach for the design and development of multimodal interfaces. In *CHI'04 extended abstracts on Human factors in computing systems*. ACM, 1325–1328.
5. Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, and others. 2001. Web services description language (WSDL) 1.1. (2001).
6. Anind K Dey, Gregory D Abowd, and Daniel Salber. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction* 16, 2 (2001), 97–166.
7. Pierre Dragicevic and Jean-Daniel Fekete. 2001. Input device selection and interaction configuration with ICON. In *People and Computers XVInteraction without Frontiers*. Springer, 543–558.
8. Sebastian Feuerstack and Ednaldo Pizzolato. 2011. Building multimodal interfaces out of executable, model-based interactors and mappings. In *Human-Computer Interaction. Design and Development Approaches*. Springer, 221–228.
9. John D Gould. 2000. How to design usable systems. *Readings in Human Computer Interaction: Towards the Year* (2000), 93–121.
10. Lode Hoste, Bruno Dumas, and Beat Signer. 2011. Mudra: a unified multimodal interaction framework. In *Proceedings of the 13th international conference on multimodal interfaces*. ACM, 97–104.
11. Christine Keller, Romina Kühn, Anton Engelbrecht, Mandy Korzetz, and Thomas Schlegel. 2013. A Prototyping and Evaluation Framework for Interactive Ubiquitous Systems. In *Distributed, Ambient, and Pervasive Interactions*. Springer, 215–224.
12. Suzanne Kieffer, J-YL Lawson, and Benoit Macq. 2009. User-centered design and fast prototyping of an ambient assisted living system for elderly people. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*. IEEE, 1220–1225.
13. Jean-Yves Lionel Lawson, Ahmad-Amr Al-Akkad, Jean Vanderdonckt, and Benoit Macq. 2009. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. 245–254.
14. Linchuan Liu and Peter Khooshabeh. 2003. Paper or interactive?: a study of prototyping techniques for ubiquitous computing environments. In *CHI'03 extended abstracts on Human factors in computing systems*. ACM, 1030–1031.
15. Luqi. 1989. Software evolution through rapid prototyping. *Computer* 22, 5 (1989), 13–25.
16. Carsten Magerkurth, Richard Stenzel, Norbert Streitz, and Erich Neuhold. 2003. A multimodal interaction framework for pervasive game applications. In *Workshop at Artificial Intelligence in Mobile System (AIMS), Fraunhofer IPSI*.
17. Martin Maguire. 2001. Methods to support human-centred design. *International journal of human-computer studies* 55, 4 (2001), 587–634.
18. Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. 2011. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 315–326.
19. Ronny Seiger, Tobias Nicolai, and Thomas Schlegel. 2014. A Framework for Controlling Robots via Brain-Computer Interfaces. In *Mensch & Computer 2014–Workshopband: 14. Fachübergreifende Konferenz für Interaktive und Kooperative Medien–Interaktiv unterwegs-Freiräume gestalten*. Walter de Gruyter GmbH & Co KG, 3.
20. Anoop K Sinha and James A Landay. 2003. Capturing user tests in a multimodal, multidevice informal prototyping tool. In *Proceedings of the 5th international conference on Multimodal interfaces*. ACM, 117–124.
21. Wolfgang Wahlster. 2006. *SmartKom: foundations of multimodal dialogue systems*. Vol. 12. Springer.
22. Mark Weiser. 1991. The computer for the 21st century. *Scientific american* 265, 3 (1991), 94–104.
23. Mark Weiser. 1993. Some computer science issues in ubiquitous computing. *Commun. ACM* 36, 7 (1993), 75–84.