Lightweight Editing of Distributed Ubiquitous Environments: The CollaborationBus Aqua Editor

Maximilian Schirmer, Bauhaus-University Weimar, Germany

Tom Gross, University of Bamberg, Germany

ABSTRACT

Cooperative ubiquitous environments support user interaction and cooperative work by adapting to the prevalent situation of the present users. They are typically complex and have many environment components—interconnected devices and software modules—that realise new interaction techniques and facilitate collaboration. Despite this complexity, users need to be able to easily adapt their environments to the respective needs of the workgroups. In this paper, the authors present the CollaborationBus Aqua editor, a sophisticated, yet lightweight editor for configuring ubiquitous environments in groups. The CollaborationBus Aqua editor simplifies the configuration and offers advanced concepts for sharing and browsing configurations among users.

Keywords: *CollaborationBus Aqua Editor, Configuration, Cooperative Ubiquitous Environments, Editor, Sharing and Browsing*

INTRODUCTION

Cooperative ubiquitous environments reach beyond single-user interaction and facilitate cooperation and collaboration among their users. They leverage interaction between users, artefacts, and devices, with the goal of softening or even eliminating the barrier between local and remote participants. For instance, a conference room can capture the positions of present persons and their actions, and then adapt the computer and projector configuration, the lighting, and the window shutters; and it could store these settings to support easy later resumption of a meeting.

The configuration of a cooperative ubiquitous environment describes the settings of the environment's components, as well as the degree and shape of the individual interaction between the components. Typically, the task of configuring an environment is realised by programmers or administrators, because it requires great insight into the underlying infrastructure and system architecture, and adequate programming skills. For instance, the rules for the adaptation

DOI: 10.4018/jdst.2011100105

behaviour of the above conference room are rather difficult to configure.

The configurations should cover the needs of the end-users and their workgroups. However, despite the progress in base technologies such as data acquisition, processing, and machine learning, creating and adapting configurations is still a complex process. In order to facilitate this process, users need empowerment for end-user configuration.

In this paper we present *CollaborationBus Aqua*—a sophisticated, yet light-weight editor for cooperative ubiquitous environments that supports elegant capturing and storing of data from the physical as well as electronic world, visual composition of configurations, and sharing and browsing of configurations among groups of configuration authors. In the next sections we discuss related work. We then present the concept and implementation of *CollaborationBus Aqua* and report on its user interaction. We exemplify the user interaction in a scenario. Finally, we conclude the paper.

END-USER EDITORS FOR UBIQUITOUS ENVIRONMENTS

There are several end-user editors for editing and managing configurations of ubiquitous environments. They provide inspiring concepts with respect to their enabling middleware (e.g., *eGadgets*), their scheme of the configurations (e.g., *iCAP*), and easy user interaction (e.g., *Jigsaw*). As a limiting factor they mostly focus on individual end-users editing configurations of single-user settings.

In *eGadgets* (Mavrommati et al., 2004) a *Gadgetware Architectural Style* (*GAS*) framework for interconnecting reusable components in the form of devices, and a GAS editor for building custom compositions were developed. While an enabling middleware manages and controls all components within the framework, the editor hides complexity from users. The editor retains insight to the dataflow to avoid behaving like a black box for users. By means of connecting the components' inputs and outputs, users generate a range of scenarios consisting of home appliances that have been adapted to be accessible through the *GAS* platform. The *GAS* framework models individual components following a plug-synapse model, where each component offers a set of abilities and requests services from other components. Devices in the physical world are represented as plugs. When different plugs are instantiated and connected, they form synapses. This model abstracts and represents compatible data types and data flows, and thus effectively helps users understand which components can be interconnected. In contrast to the *eGadgets* editor, *CollaborationBus Aqua* focuses on a cooperative composing process for ubiquitous computing environments, and offers a sharing and browsing mechanism with synergy notifications.

Another related editor is the *iCAP* (Lim & Dey, 2009; Sohn & Dey, 2003) editor that allows users to prototype applications and scenarios for context-aware environments. Following a pen-based interaction technique, the system's components (input and output devices) may be interconnected to form a conditional rule-based construct in a user-friendly way. The *iCAP* editor allows users to draw their own sketches, which are used to represent the underlying devices within the editor environment. These sketches help to generate a deeper understanding of the constructed prototype and the interrelations between devices. When components are connected, their rule-based interaction can be tested in the editor's run mode that allows the simulation of certain input states as well. Similar to the *eGadgets* editor, *iCAP* realises a single-user concept. In contrast, *CollaborationBus Aqua* aims at leveraging cooperative editing of ubiquitous computing compositions and offers synergy notifications.

The *Jigsaw* editor (Dey & Newberger 2009; Humble et al., 2003) is a graphical front-end to a user-oriented framework that supports users in configuring domestic ubiquitous environments. Users move dragging components (represented as jigsaw pieces) from the editor's list view onto a canvas to create compositions that interconnect hardware sensors and devices from a domestic environment. Differences among the jigsaw pieces (either output port, or input port, or both) reflect the connection properties of the underlying devices and help users to identify what devices are compatible and can be connected. The editor provides both visual and auditory feedback when interactions occur, and visualises the dataflow to help users keep track of sensor updates. In contrast to the Jigsaw editor, *CollaborationBus Aqua* relies on a sophisticated sensor-based ubiquitous computing event notification infrastructure with multifarious environment components and offers powerful mechanisms for filtering or further processing of gathered data.

The *UBI-Designer toolkit* (Vastenburg et al., 2009) is a web-based graphical editor for sensor networks and infrastructures for ubiquitous computing. The toolkit represents a high-level abstraction of the environment's underlying components and addresses designers of context-aware ubiquitous computing environments as end-users. The toolkit provides access to sensors, processors, and rules. *Sensors* are sources of information in the ubiquitous computing environment, gathered by either hardware sensors, or virtual sensors. Users can explore all available sensors and their current state, filter sensors according to projects, and simulate sensor values. *Processors* represent software algorithms for the low-level interpretation of sensor data. Users can choose from a selection of pre-defined processors that implement algorithms such as pattern recognition. *Rules* trigger actuators in the environment, based on data gathered by sensors and processed by processors. Users create rules for their ubiquitous computing scenarios with a rule-editor that abstracts from the complexity of the underlying *JESS* rule engine. In the UBI-Designer toolkit, configurations of sensors, processors, and rules can be saved as *Projects*. Each project consists of links to the components and simplifies their clustering. In contrast to the *UBI-Designer toolkit*, *CollaborationBus Aqua* provides a rich graphical user interface with visual programming that allows users to interact directly with graphical representations of an environment's components.

The *RePlay* system (Newman et al.. 2010) aims at designers of ubiquitous environments and enables them to recreate states of the environment. By simulating sensor data, replaying helps to test how a ubiquitous environment adapts to a prevalent context. Designers and developers can then adapt the configuration of the environment according to the simulation results. RePlay provides a graphical user interface that is very similar to multi-track audio or video editors and follows a scenario-based approach. Gathered sensor data is represented as *Clips* in a clip library. Users organise several clips from different sensors in *Episodes*. Furthermore, *RePlay* supports *Transforms* that can be applied on clips. Transforms are processing units that modify sensor data. Several pre-defined transforms are available, for example the *Identity Transform* for changing the associated user of sensor data, the *Dwell Transform* for creating delays in sensor data, and the *GPS Noise Transform* for manipulating GPS sensor data. In contrast to *RePlay*, the *CollaborationBus Aqua* editor allows users to configure the interconnections between components of a ubiquitous environment and provides a sharing mechanism for configurations.

The *Topiary* system (Li et al., 2004) is related to *CollaborationBus Aqua* because it presents an interesting approach for prototyping ubiquitous, location-enhanced applications. In Topiary, users model location contexts in a graphical user interface that consist of entities, which can be *people*, *places*, or *things*. People represent users in a location-enhanced ubiquitous application. Places are defined by boundaries that users draw on a map. Things are generic entities that can be associated with places. Topiary supports two distinct types of location contexts: *presence contexts* that describe the spatial relation of persons and things at places, and *proximity contexts* that model the adjacency of entities. Several location contexts form a scenario that represents a complex situation with a number of entities and location contexts. In contrast to the *Topiary* system, *CollaborationBus Aqua* goes beyond the prototyping of location-enhanced applications, and allows users to configure existing real-world components of ubiquitous environments.

Fokidou et al. (2008) have presented two interesting graphical user interface prototypes for configuring pervasive environments, which specifically address the requirements of elderly people (*Bee Prototype*) and young adolescents (*Haring's World Prototype*). The graphical user interface of the *Bee Prototype* follows the metaphor of a beehive that abstracts from the components of the ubiquitous computing environment. The actual configuration is created with the help of wizards and forms. The Haring's World Prototype provides a graphical user interface that leverages creativity. During the configuration process, users are able to communicate with others using the integrated instant messaging service. The prototype allows users to discover present artefacts of the ubiquitous environment, and to define associations between these artefacts. These associations are realised as a set of rules. In contrast to these prototypes, *CollaborationBus Aqua* is fully functionally and has been implemented and deployed as a graphical desktop application.

COLLABORATIONBUS AQUA CONCEPT

The requirements for *CollaborationBus Aqua* were from our own experience of developing cooperative ubiquitous environments for many years, and lessons learned from related work such as the examples described in the previous section. In this section we focus on the following three core concepts of *CollaborationBus Aqua:* advanced and easy capturing of data, composing configurations visually, and sharing and browsing configurations.

Advanced and Easy Capturing of Data

The *CollaborationBus Aqua* editor includes an ubiquitous sensor-based platform that distributes and processes gathered data in the form of sensor events. The powerful sensor-based platform *Sens-ation* (Gross et al., 2006) manages all the capturing, processing, and storing of the data for the users in the background. The combination of *CollaborationBus Aqua* and *Sens-ation* provides access to the environment components: sensors that gather data, inference engines that process gathered data, and actuators that trigger feedback in the user environment. Furthermore, *Sens-ation* offers a broad range of gateways as interfaces for the easy management of components and access to both raw and processed sensor data.

Sensors are either hardware sensors for light, movement, temperature, noise; or software sensors for applications such as email, Web browser, and office applications. The gathered data is used to abstract awareness information about the users in a cooperative ubiquitous environment.

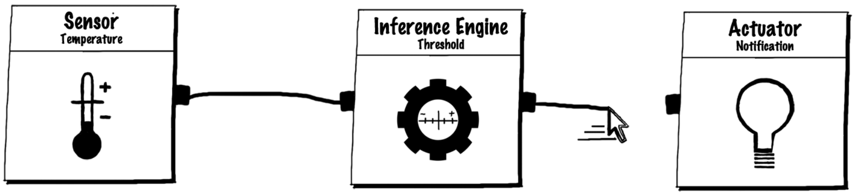
Inference engines in the *Sens-ation* platform process incoming sensor data. This processing mechanism allows inferring higher-order information from the raw sensor data. Processing results vary from simple mathematical calculations (e.g., mean values) up to complex interdependent processing chains that involve multiple inference engines' results. Results from inference engines are transferred back to the platform as sensor events, so clients and actuators can access them through all available gateways.

Actuators realise actions within the environment according to the results of the inference process. Just like sensors, actuator components are either software applications or hardware devices. While hardware actuators change physical settings within the environment, software actuators typically serve as means of presenting notifications on a computer monitor.

Composing Configurations Visually

In *CollaborationBus Aqua* users visually compose configurations using the components of the platform. The editor follows the visual programming paradigm that supports configuration tasks by means of visually appealing graphical representations (Myers, 1986). These graphical representations abstract programmatic

Figure 1. Scheme of configurations including a sensors, inference engine, and actuator



behaviour, yet still provide an indication of the underlying technology. The *CollaborationBus Aqua* editor uses distinct graphical elements for sensors, inference engines, and actuators.

Figure 1 shows our scheme of configurations consisting of one or more sensors, one or more inference engines, and one or more actuators. In this exemplary configuration, a user wants to be notified when the temperature measured by a temperature sensor has reached a defined threshold. The user has connected the sensor's output to an inference engine's input, and the inference engine's output to an actuator's input. The inference engine evaluates the incoming temperature and notifies the actuator.

Environment components are instantiated by drag-and-drop. Users create connections among them by drawing lines between two individual representations. The editor handles the necessary technical procedures in the background and provides an indication whether the established connections are correct on a technical as well as on a semantic level. The data type validation mechanism evaluates compatibilities and notifies users with a warning if they create connections that form incompatible relationships between components. This avoids that the composition results in unpredictable behaviour within the environment.

Typical examples for incompatibilities are: connecting two outputs of components (e.g., connecting the outputs of two sensors with each other, cf. Figure 2 (a)), or connecting components with incompatible data types (e.g., connecting a temperature sensor with a Boolean inference engine Figure 2 (b)). In the first case, there is no data flow because in the sensor model of *Sens-ation*, sensors only output data to the platform. In the second case, a movement sensor is connected with an inference engine that implements a logical AND operator. Clearly, the data types of both components are not compatible with each other. While the inference engine awaits Boolean values as input, the movement sensor only provides numerical float values as output.

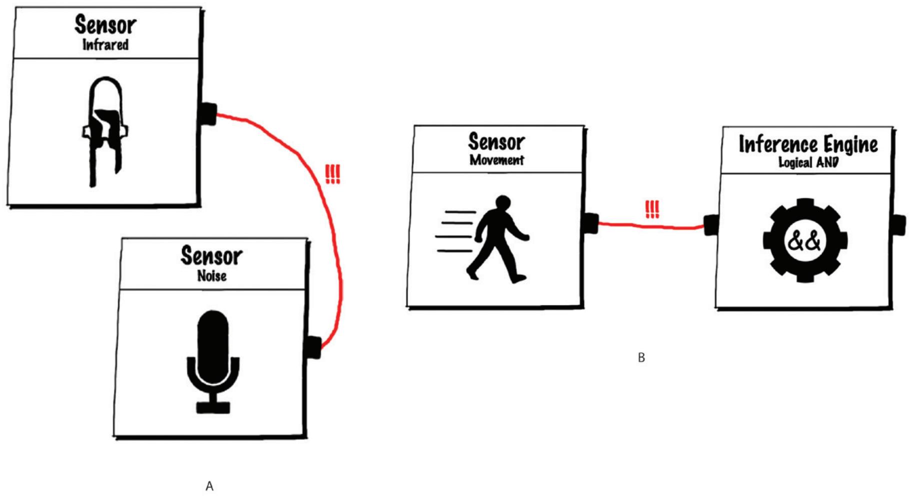
Sharing and Browsing Configurations

CollaborationBus Aqua encourages its users to share, explore, and reuse configurations. In *CollaborationBus Aqua* all configurations are accessible through a shared repository, which allows groups of authors to cooperate during the composing. This repository especially provides beginners, who do not have experience with configuring ubiquitous environments, with an entry point to the system (Mackay, 1990). With a growing number of configurations in the repository, beginners get a good sample of configurations and learn about their cooperative ubiquitous environment and configuration options therein.

Users can choose between sharing and privacy—that is, they can either place their configuration in a shared repository that every user in a group can access, or save their configuration in a private local file (see Greif & Sarin, 1986 for early findings on sharing and privacy).

The shared repository facilitates synergies among users. Components instantiated within the *CollaborationBus Aqua* editor refer to concrete physical artefacts or software instances. We define synergies as the similar use of the same components within the repository of

Figure 2. Incorrectly established Sensor (a) and Sensor-Inference Engine connections



available configurations. When users access components that are already part of other users' configurations, all users involved receive information about their mutual components. The notification encourages them to explore each other's configuration or contact each other to discuss synergies.

The underlying mechanism works as follows (cf. Figure 3): the shared repository of configurations is a set of components. Any configuration therein forms a subset of components. When the intersection of any number of configurations produces a set that is not the empty set, synergies occur.

CollaborationBus Aqua User Interaction

CollaborationBus Aqua consists of the Main Window (cf. Figure 4(a)) and the Inspector (cf. Figure 4(b)). The Main Window provides four parts: (aa) the Operation Mode toolbar on the top end of the window, (ab) the Component Browser below, (ac) the Composer in the centre of the window, and (ad) the Statusbar in the bottom of the window.

Connecting, Editing, and Sharing

The Operation Mode toolbar of the Main Window of *CollaborationBus Aqua* provides on the left side the access to three basic Operation Modes: Connecting, Editing, and Sharing. Switching to one of the modes changes the content of the Main Window. On the right side of the Operation Mode toolbar, two additional buttons allow users to delete components and to open up the Inspector. In the Connecting Mode, users either enter the appropriate connection details of the *Sens-ation* instance they want to connect to or select one from the connection history list. Once users establish a connection, the Editing and Sharing Modes can be accessed. The Editing Mode is the core of the application and provides the Component Browser, the Composer, and the Inspector. From the Component Browser, components are instantiated by simply dragging them to the Composer, where they are transformed into graph nodes. The Inspector allows exploring and configuring selected components. In the Sharing Mode, users browse the repository of available configurations to learn about their

Figure 3. Synergies in shared cooperative ubiquitous environment configurations

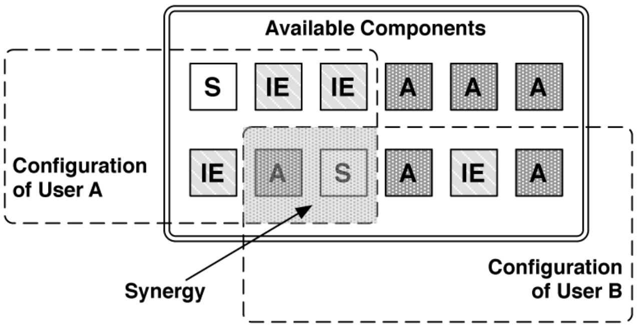
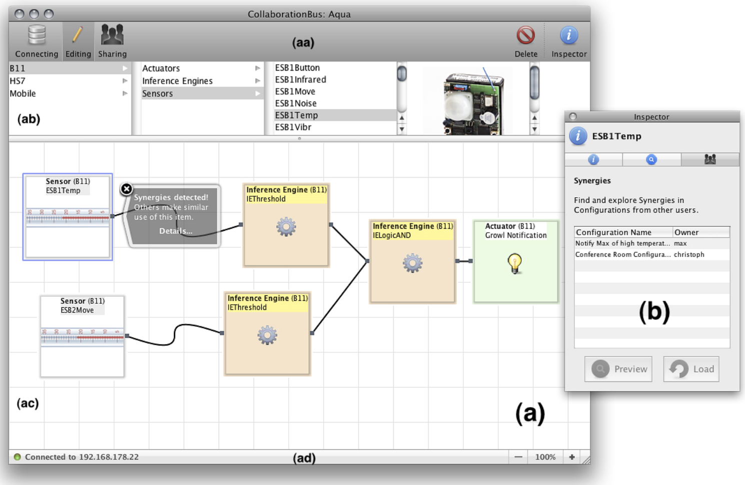


Figure 4. Graphical user interface of CollaborationBus Aqua, with (a) the Main Window, and (b) the Inspector



Synergies detected!
Others make similar use of this item.
Details...

Find and explore Synergies in Configurations from other users.

Configuration Name	Owner
Notify Max of high temperat...	max
Conference Room Configura...	christoph

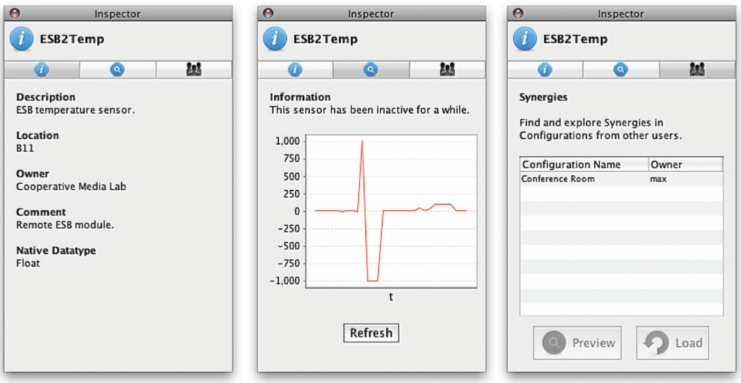
environment or to find a template as a starting point for an editing process.

Exploring and Configuring Components

The Inspector provides detailed information and configuration options for components in the Editing Mode and dynamically changes its content in relation to selected components. For example, if users select a sensor component, the inspector only displays information about it and its recent events; if they select an inference engine or actuator component, the Inspector also provides means of configuring their parameters. The Inspector is a floating palette window always located on top of other windows of the editor. Figure 5 shows the Operation Modes of the Inspector. Changing between Operation Modes follows the pattern of the Main Window: a toolbar with three different toggle buttons representing the associated modes.

The (a) General Information Mode displays common data about the selected component (e.g., its location, owner). This helps users to identify physical components in their environment. Furthermore, they provide a common ground for communication with other users of

Figure 5. The operation modes of the inspector: (a) general information, (b) recent events, and (c) synergy browser



Description
ESB temperature sensor.

Location
B11

Owner
Cooperative Media Lab

Comment
Remote ESB module.

Native Datatype
Float

Information
This sensor has been inactive for a while.

Synergies

Find and explore Synergies in Configurations from other users.

Configuration Name	Owner
Conference Room	max

these components, because they allow explicit identification. The (b) Recent Events Mode provides an overview of the component's recent condition, which is mostly useful for sensor and inference engine components. It displays either a graphical or a tabular visualisation of the recent events, according to the component and its data type. For instance, a temperature sensor produces numerical event values, which can be visualised as a temperature graph, while an inference engine that evaluates a given input value against a threshold will output Boolean values, which require a tabular visualisation. The (c) Synergy Browser Mode allows user to quickly inspect a component's synergies within other configurations in the form of a tabular configuration listing. In the case of existing synergies, the Inspector provides two buttons for either previewing or loading a selected configuration with synergies.

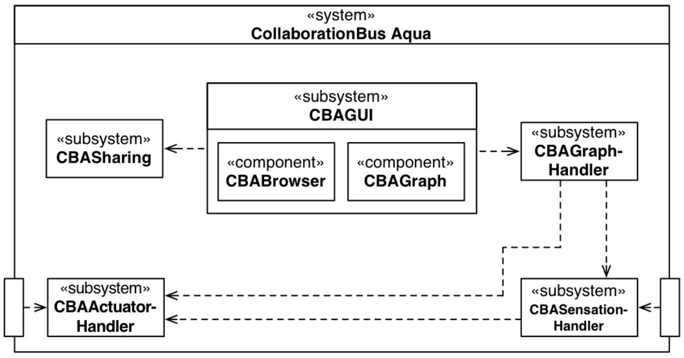
CollaborationBus Aqua Implementation

The *CollaborationBus Aqua* editor is a standalone application implemented in Java 1.5.0_13, MySQL 5.0.41, and with Apache 2.0.59 on Mac OS X 10.4.9 and as such was straight-forward to implement and provides user interaction concepts that are well known to end-users. It acts as a client to the *Sens-ation* sensor platform.

CollaborationBus Aqua is comprised of five core subsystems that implement the main program logic (cf. Figure 6). The *CBAGUI* subsystem is responsible for managing both the *CBABrowser* component as well as the *CBAGraph* component that forms the editor's core. The *CBAGraphHandler* subsystem manages the creation of the visual representations for the components and devices of the environment. The *CBASensationHandler* subsystem communicates directly with the associated *Sens-ation* instance via XML-RPC (Scripting News Inc., 2010) and distributes the gathered data and its available components to the *CBAGraphHandler*. The management and delegation of actuator components is realised by the *CBAActuatorHandler* subsystem. It manages all available and instantiated actuators and communicates directly to *Sens-ation* via XML-RPC. It provides actuator parameters for the graphical representations of actuator components to the *CBAGraphHandler*. The *CBASharing* subsystem directly relates to the *CBAGUI* subsystem. It handles access to the repository of shared configurations by delegating tasks to a database server. It also processes the related data for display within the graphical user interface and implements the synergy finding algorithm.

Subsequently, we explain how the concepts are implemented with the five subsystems.

Figure 6. Component diagram of CollaborationBus Aqua



CollaborationBus Aqua has been completely implemented and deployed.

Deployment

We have deployed and tested *CollaborationBus Aqua* in a cooperative ubiquitous environment in our workgroup. In Figure 7, we introduce a typical setup. This setup consists of two users on different workstation computers at two different buildings of our Cooperative Media Lab at the Bauhaus-University in Weimar. Each *CBAWorkstation* computer (*ccml13* and *ccml26* are 27-inch iMacs with 3.06 GHz Intel C2D, 4 GB of main memory, running Mac OS X 10.6.4) is equipped with the *Collaboration-Bus Aqua* editor. The workstation computers attach to the university's CAN (Campus Area Network) via a 100 MBit/s Ethernet connection. All instances of the *CollaborationBus Aqua* editor communicate with a central *CBAServer* using XML-RPC and HTTP.

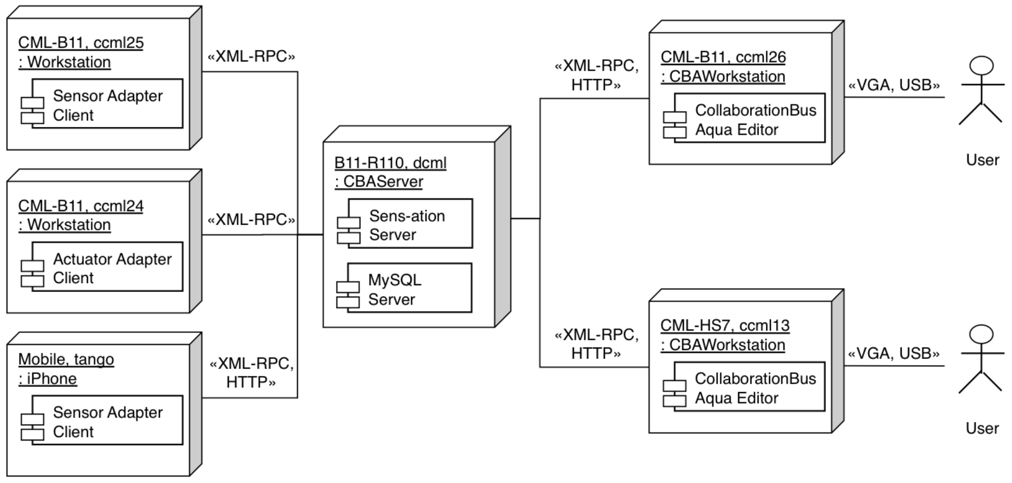
The *CBAServer* is deployed on our *dcml* server, which is a PowerMac G5 with dual 1.8 GHz G5 processors, 1 GB of main memory, running Mac OS X 10.5.8. The *CBAServer* provides the *Sens-ation* platform (version 5.7) as well as a MySQL database server (version 5.0.41). The *Sens-ation* platform manages all environment components (i.e., sensors, inference engines, and actuators) of the cooperative ubiquitous environment. It also delegates the communication between the components, and the data exchange between clients. The database server persistently stores the configuration repository of the *CollaborationBus Aqua* editor. This repository contains XML representations of the environment configurations.

In our setup, *Sensor Adapter Clients* and *Actuator Adapter Clients* are deployed on additional workstation computers (*ccml24* and *ccml25*, also iMac 27-inch C2D), and on an iPhone 3G (*tango*). *Sensor Adapter Clients* provide interfaces for environment sensors and send gathered sensor data to the *Sens-ation* platform of the *CBAServer*. The *Actuator Adapter Client* provides the necessary implementation for triggering environment reactions (e.g., displaying notifications on a computer display, controlling other computing devices, or manipulating real-world objects using servo motors).

Data Capturing

CollaborationBus Aqua requests and obtains data as a client for the *Sens-ation* platform. The editor implementation makes use of the XML-RPC gateways with synchronous communication. *CollaborationBus Aqua* relies on synchronous communication, because it is important to apprise all users of the current condition of the environment without noticeable delay. The *CBASensationHandler* subsystem of *CollaborationBus Aqua* implements communication management and initiation. It encapsulates connections to various

Figure 7. Deployment diagram of a typical CollaborationBus Aqua setup in our environment



Sens-ation platforms and keeps a history. The *CBASensationHandler* acts as a surrogate for the actual *Sens-ation* connection that is active. Instead of interacting directly with *Sens-ation*, all components of the *CollaborationBus Aqua* system direct their requests to the *CBASensationHandler*. The *Sens-ation* connection components make use of the Java XmlRpcClient implementation as well as the Java WebServer implementation (both from the corresponding Apache project framework Apache Software Foundation, 2010). While the XmlRpcClient is used to send requests to *Sens-ation* (e.g., for acquiring information about available sensors), the WebServer component listens for notifications that are sent from *Sens-ation* when a sensor event of an observed sensor occurs.

Visual Composing

Visual composing in the *CollaborationBus Aqua* editor is implemented in the *CBAGraphHandler* subsystem and based on a Model-View-Controller (MVC) pattern. The base of the visual composing graphical user interface is an interactive graph interface element, the *CBAGraph*. This graph implementation bases on the Java JGraph (JGraph Ltd, 2009) framework that provides a graph component for the Java Swing framework. The *CBAGraph* component in the *CBAGraphHandler* subsystem contains the *CBAGraphModel* with the necessary data for each node in the graph, as well as information about relationships between graph nodes. The *CBACellViewFactory*, *CBACellView*, and *CBAVertexRenderer* components realise the visual representations of these graph nodes, in conjunction with the *CBAGraphRouting* component that generates control points for the rendering of smooth spline-based edges between the nodes of the graph.

Sharing and Browsing Configurations

Sharing and browsing configurations is implemented in the *CBASharing* subsystem. Its *CBASharingDatabase* provides an abstraction layer to the underlying MySQL database and implements the functional behaviour to save and load configurations. An identifier string and the creator of the composition uniquely identify every composition. Each composition in the GUI is serialised to an internal XML representation, which facilitates their internal handling, and includes all necessary information to reload, edit, and share configurations. The identifier string, the creator, and the XML representation of the composition are stored persistently in the database.

The XML representation for all components of a configuration follows a simple structure (cf. Figure 8). Every component has a type identifier, a native data type (e.g., Float, String, Boolean), a location identifier, and a unique component identifier.

In order to detect synergies in shared configurations, a set of comparisons across all configurations in the repository is necessary. The components' identifiers and their locations are compared. When both the identifiers and the locations of two components match, a synergy is detected. For this purpose, an XML pull parser sequentially scans all configurations in the repository and evaluates the contained components. When a synergy is detected, a synergy flag is set for the corresponding component in the *CBAGraphModel*. During the graph rendering cycle, the *CBAGraphHandler* triggers the display of a graphical synergy notification for all graph nodes that are marked with the synergy flag. The synergy notification also contains the identifiers of configurations with synergies, as well as information about their authors. Users can directly explore and browse these configurations to find out more about them.

Scenario

In this section, we present a scenario to illustrate a typical editing situation in a cooperative ubiquitous environment. The scenario involves two users (Walter and Henry) who are using *CollaborationBus Aqua* for configuring devices and components in their research facility's cooperative ubiquitous environment.

Henry is a research and teaching assistant and Walter is the room administrator at the research facility that Tony is working at. It is Walter's responsibility to provide technical and administrative support for a number of conference rooms within the building.

Walter receives many calls from users of the conference rooms reporting a broken video projector. Most of the time he finds the project system perfectly intact, but the conference room users simply forgot to turn on the main power switch. This Switch is necessary to reduce the standby power output of the room's devices. It would simplify Walter's work a lot if there were a system that automatically prepared the conference room for a meeting when a meeting situation is imminent.

With the help of the *CollaborationBus Aqua* editor, Walter creates a configuration of sensors, inference engines, and a simple actuator that controls a power switching relay. The configuration (cf. Figure 9) consists of a movement sensor (MovementRoom42) and a noise sensor (NoiseRoom42). Their gathered data is used by three inference engines in order to determine if there are people present in the conference room. Two of the inference engines are of the type IEThreshold, which means these inference engines evaluate incoming sensor data (in this case noise and movement levels) against user-specified thresholds. The third inference engine has the type IELogicAnd. It realises a logical AND operator for incoming sensor data. If both IEThreshold inference engines determine that their thresholds have been reached, the relay board actuator is triggered. The relay then powers the video projector on.

When Walter placed the actuator component in his configuration, he noticed a small dialogue window that appeared next to the component, stating that there are other configurations that make use of the power switching relay actuator in conference room 42 as well (cf. Figure 10).

Walter clicks the button "Details…", just like it is proposed in the dialogue to open the inspector window. Within the inspector, the synergy browser mode is activated and presents a list of other configurations (cf. Figure 11). These configurations contain components that are also present in Walter's configuration.

From the list of configurations in the synergy browser, Walter learns that a user named Henry has also created an environment configuration for conference room 42. Henry's configuration involves the power switch relay as well. Walter is curious to learn about Henry's

Figure 8. XML representation of a sensor component

```
<de.cmlab.collaborationbus.utility.VertexData
   serialization="custom">
  <map>
   <default>
     <loadFactor>0.75</loadFactor>
     <threshold>12</threshold>
   </default>
   <int>16</int>
   <int>4</int>
   <string>Type</string>
   <string>Sensor</string>
   <string>NativeDataType</string>
   <string>Float</string>
   <string>Location</string>
   <string>B11</string>
   <string>ID</string>
   <string>ESB1Temp</string>
  </map>
</de.cmlab.collaborationbus.utility.VertexData>
```

configuration and uses the synergy browser to load it directly into his *CollaborationBus Aqua* editor application. After exploring the configuration, he goes to see Henry. Henry is happy to introduce his configuration to Walter and explains what he did in great detail.

As it turns out, Henry is a regular user of the conference room and wanted to save the time required for turning on the main power switch over and over again every day. Henry organises all his appointments in a calendar application program on his computer. All the meetings in the conference room are included, as well. So he envisioned a system that would prepare the conference room just before a scheduled meeting begins.

Henry's configuration involved a calendar sensor that provides events right before an appointment is due, an inference engine that evaluates if the appointment is set to take place in the conference room 42, and the same relay board as actuator that Walter used in his configuration.

While they were discussing each other's configurations and the synergies they have created with them, Walter and Henry decide to work together on an even better configuration that works both for spontaneous, unscheduled meetings (like in Walter's configuration), and for scheduled meetings as well.

CONCLUSIONS AND FUTURE WORK

Cooperative ubiquitous environments combine aspects of ubiquitous computing with the general aim of supporting collaboration and cooperation in a shared information space, as envisioned in the core ideas of computer-supported cooperative work (Bannon & Schmidt, 1989). These environments require a lot of interconnected devices and software components in order to realise new interaction techniques and facilitate collaboration through them.

We introduced *CollaborationBus Aqua* that provides mechanisms and easy interfaces for accessing sensors and the event data they capture as well as for composing configurations. It is a continuation of our *CollaborationBus editor* (Gross & Marquardt, 2007) with a special focus on end-users—combining easy handling with complex compositions. In particular, this editor is based on a sophisticated interaction concept that abstracts from the technical complexity of the cooperative ubiquitous environment and its components and allows users to focus on the semantics of their configurations. With the sharing and browsing mechanisms, users can exchange their configurations—this is particularly helpful for novice users who can browse existing configurations and do learning by example.

Figure 9. Walter's conference room configuration

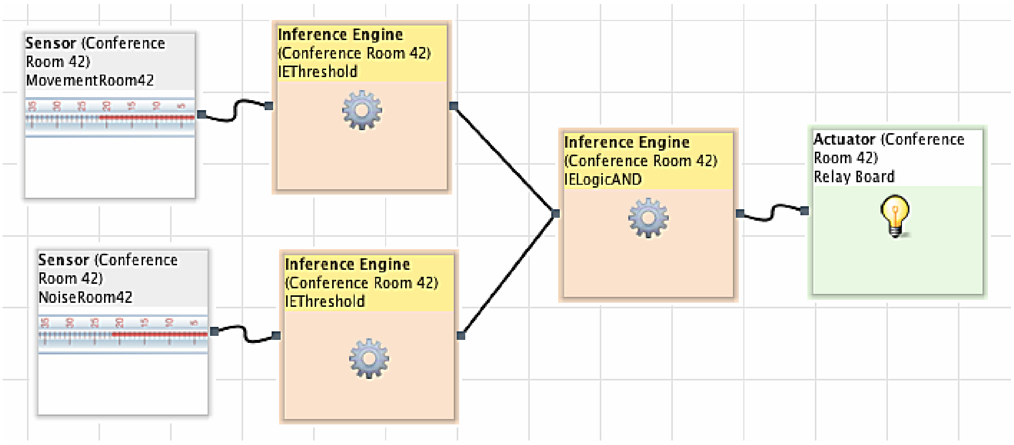
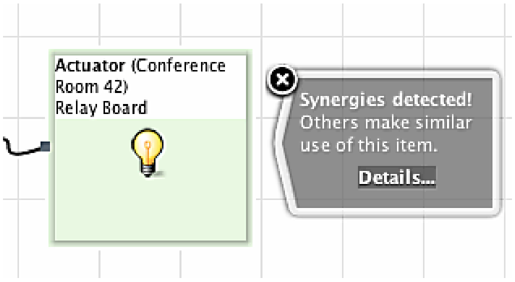

Sensor (Conference Room 42) MovementRoom42
Inference Engine (Conference Room 42) IEThreshold
Inference Engine (Conference Room 42) IELogicAND
Actuator (Conference Room 42) Relay Board
Sensor (Conference Room 42) NoiseRoom42
Inference Engine (Conference Room 42) IEThreshold

Figure 10. A synergy was detected in Walter's configuration



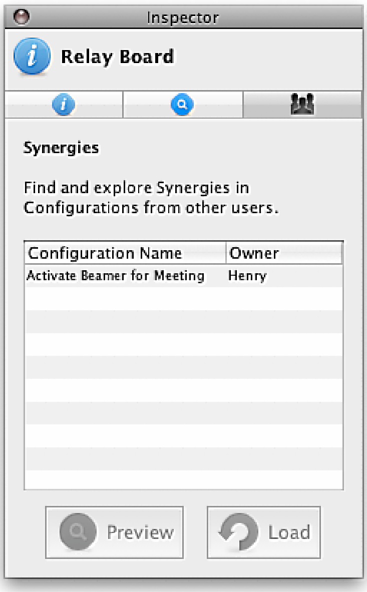
Synergies detected! Others make similar use of this item.

As our overview of related end-user editors for ubiquitous environments has shown, this particular field of research presents a lot of interesting challenges that cover several state-of-the-art topics in ubiquitous computing: sensor infrastructures and frameworks, natural and alternative user interaction, recording and replay of sensor data, use of location data, and new metaphors for graphical user interfaces. With *CollaborationBus Aqua*, we introduce means for cooperative editing of ubiquitous environments and therefore incorporate aspects from computer-supported cooperative work (CSCW). We think that this combination of ubiquitous computing and concepts of CSCW presents an important contribution to end-user editors for ubiquitous environments, and also for ubiquitous computing in general. Allowing users to easily share, discuss, and edit their configurations collaboratively, reduces the complexity of the configuration process.

Besides its regular deployment in our lab's ubiquitous environment, we have successfully deployed *CollaborationBus Aqua* in a media space setting (Gross et al., 2010). In this scenario, the editor allowed users of the media space to define what kind of information they disclose to others, and to specify a granularity level (ranging from undisclosed to full disclosure with two levels in-between). The graphical user interface of the editor, mostly the inspector window, was modified in this setting in order to provide the necessary mechanisms for controlling the disclosure level of media space sensors.

The *CollaborationBus Aqua* editor currently supports the management of sensors,

Figure 11. The synergy browser showing Walter's synergy with Henry

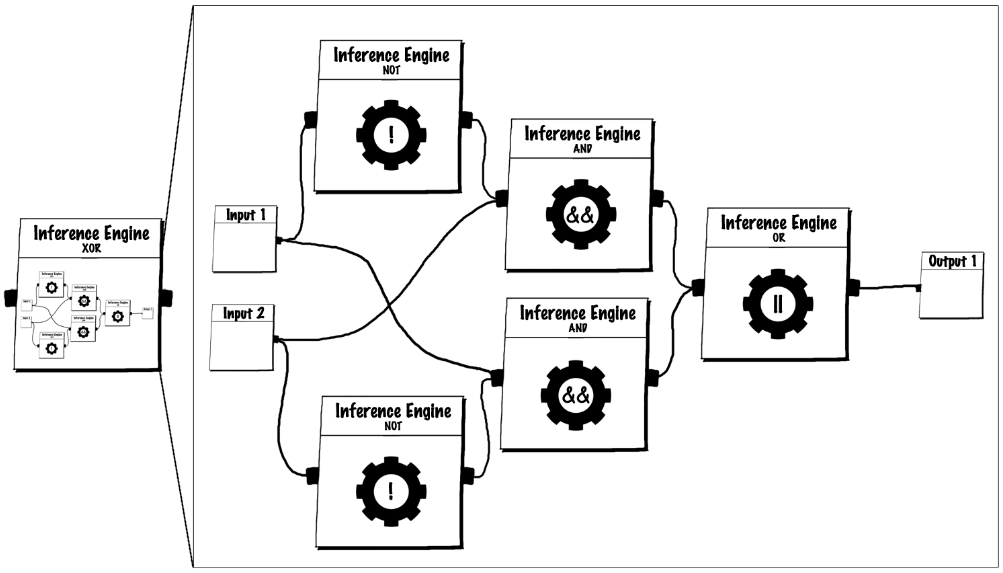


Synergies

Find and explore Synergies in Configurations from other users.

Configuration Name	Owner
Activate Beamer for Meeting	Henry

Figure 12. Nested inference engine components



inference engines, and actuators. For the future, users would benefit from including more capabilities for visualising and simulating sensor data. While users are presented with a simple configuration process, the editor in the current form has limitations concerning the scalability of the presentation of large configurations. This is particularly due to the fact that it can only display flat configurations, where up to ten components can be seen and manipulated at a time on a typical 17 inch screen. Introducing a nesting mechanism for components would allow users to divide and conquer their bigger configurations into multiple levels of abstraction.

Figure 12 presents a sketch of the proposed concept for a nesting mechanism in the *CollaborationBus Aqua* editor. In this case, the behaviour of an XOR inference engine is recreated with several inference engines that realise the logical equivalent to XOR, $(\overline{A} \wedge B) \vee (A \wedge \overline{B})$. In the future, we want to explore suitable user interface metaphors and concepts that make this nesting mechanism easy to use and understandable for end-users.

Recent research activities in ubiquitous computing integrate machine learning techniques to provide a very high degree of adaptability to prevalent user contexts (Fogarty et al., 2005; Patterson et al., 2003; Tapia et al., 2004). The advantage of this approach is that the required complexity for configuring ubiquitous environments decreases quickly once the environment has learned the desired reactions according to user input or user contexts. We see two interesting approaches for the *CollaborationBus Aqua* editor in this context: (1) configuring the machine learning algorithms with the editor, and (2) suggesting users the most suitable components based on their current configuration. In the first approach, *CollaborationBus Aqua* could be used to provide an initial configuration of the ubiquitous environment, but also to create the configuration and composition of the machine learning algorithms. This would allow users to visually compose the processing chain from data acquisition to classification. In the second approach, the editor could be used to propose users the most suitable components, based on the components they are currently using in their configuration. This approach requires an analysis algorithm for detecting typical configurations with respect to commonly found combinations of components.

ACKNOWLEDGMENTS

The authors would like to thank Christoph Beckmann, Mirko Fetter, Nicolai Marquardt and the other members of CML, as well as the anonymous reviewers for valuable feedback. Part of the work has been funded by the Federal Ministry of Transport, Building, and Urban Affairs and by the Project Management Juelich (TransKoop FKZ 03WWTH018).

REFERENCES

Apache Software Foundation. (2010). *About Apache XML-RPC*. Retrieved from http://ws.apache.org/xmlrpc/

Bannon, L. J., & Schmidt, K. (1989, September 13-15). CSCW: Four characters in search of a context. In *Proceedings of the First European Conference on Computer-Supported Cooperative Wor*k, Gatwick, UK (pp. 358-372).

Dey, A. K., & Newberger, A. (2009, April 4-9). Support for context-aware intelligibility and control. In *Proceedings of the Conference on Human Factors in Computing Systems*, Boston, MA (pp. 859-868). New York, NY: ACM Press.

Fogarty, J., Hudson, S. E., Akteson, C. G., Avrahami, D., Forlizzi, J., & Kiesler, S. (2005). Predicting human interruptibility with sensors. *ACM Transactions on Computer-Human Interaction*, *12*(1), 119–146. doi:10.1145/1057237.1057243

Fokidou, T., Romoudi, E., & Mavrommati, I. (2008, October 13-15). Designing GUI for the user configuration of pervasive awareness applications. In *Proceedings of the 5th IADIS International Conference Cognition and Exploratory Learning in Digital Age*, Freiburg, Germany (pp. 29-37).

Greif, I., & Sarin, S. (1986, December 3-5). Data sharing in group work. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, Austin, TX (pp. 175-183). New York, NY: ACM Press.

Gross, T., Beckmann, C., & Schirmer, M. (2010, February 17-19). The PPPSpace: Innovative concepts for permanent capturing, persistent storing, and parallel processing and distributing events. In *Proceedings of the Eighteenth Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, Pisa, Italy (pp. 359-366). Washington, DC: IEEE Computer Society.

Gross, T., Egla, T., & Marquardt, N. (2006). Sensation: A service-oriented platform for developing sensor-based infrastructures. *International Journal of Internet Protocol Technology*, *1*(3), 159–167.

Gross, T., & Marquardt, N. (2007, February 7-9). CollaborationBus: An editor for the easy configuration of ubiquitous computing environments. In *Proceedings of the Fifteenth Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, Naples, Italy (pp. 307-314). Washington, DC: IEEE Computer Society.

Humble, J., Crabtree, A., Hemmings, T., Akesson, K.-P., Koleva, B., Rodden, T., & Hansson, P. (2003, October 12-15). "Playing with the bits" user-configuration of ubiquitous domestic environments. In *Proceedings of the Fifth International Conference on Ubiquitous Computin*g, Seattle, WA (pp. 256-263).

JGraph Ltd. (2009). *JGraph homepage.* Retrieved from http://www.jgraph.com

Li, Y., Hong, J. I., & Landay, J. A. (2004, October 24-27). Topiary: A tool for prototyping location-enhanced applications. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, Santa Fe, NM (pp. 217-226). New York, NY: ACM Press.

Lim, B. Y., & Dey, A. K. (2009, September 30-October 3). Assessing demand for intelligibility in context-aware applications. In *Proceedings of the 11th International Conference on Ubiquitous Computing*, Orlando, FL (pp. 195-204). New York, NY: ACM Press.

Mackay, W. E. (1990, October 7-10). Patterns of sharing customisable software. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Wor*k, Los Angeles, CA (pp. 209-221). New York, NY: ACM Press.

Myers, B. A. (1986, April 13-17). Visual programming, programming by example, and program visualisation: A taxonomy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Boston, MA (pp. 59-66). New York, NY: ACM Press.

Newman, M. W., Ackerman, M. S., Kim, J., Prakash, A., Hong, Z., Mandel, J., & Dong, T. (2010, October 3-6). Bringing the field into the lab: Supporting capture and replay of contextual data for design. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, New York, NY (pp. 105-108). New York, NY: ACM Press.

Patterson, D. J., Liao, L., Fox, D., & Kautz, H. (2003, October 12-15). Inferring high-level behavior from low-level sensors. In *Proceedings of the Fifth International Conference on Ubiquitous Computing*, Seattle, WA (pp. 73-89).

Scripting News Inc. (2010). *XML-RPC homepage.* Retrieved from http://www.xmlrpc.com/

Sohn, T., & Dey, A. (2003, April 5-10). iCap: An informal tool for interactive prototyping of context-aware applications. In *Proceedings of Extended Abstracts of the Conference on Human Factors in Computing System*s, Fort Lauderdale, FL (pp. 974-975). New York, NY: ACM Press.

Tapia, E. M., Intille, S. S., & Larson, K. (2004, April 18-23). Activity recognition in the home using simple and ubiquitous sensors. In *Proceedings of the Second International Conference on Pervasive Computing*, Vienna, Austria (pp. 158-175).

Vastenburg, M. H., Fjalldal, H., & Mast, C. V. D. (2009, June 9-13). Ubi-Designer: A web-based toolkit for configuring and field-testing UbiComp prototypes. In *Proceedings of the 2nd International Conference on Pervasive Technologies Related to Assistive Environmen*ts, Corfu, Greece (pp. 1-6). New York, NY: ACM Press.

Maximilian Schirmer, MSc is a PhD student and research and teaching assistant at the Mobile Media Group of the Bauhaus-University Weimar, Germany. He received a Master of Science degree in Media Systems at the Bauhaus-University in 2010. His research interests focus on Ubiquitous Computing, Mobile Computing, Context Awareness, and Energy Awareness in Software Systems. For further information (including a list of publications), please visit http://www.maximilianschirmer.net

Tom Gross is full professor and chair of Human-Computer Interaction at the University of Bamberg, Germany. His research interests are particularly in the fields of Human-Computer Interaction, Computer-Supported Cooperative Work, and Ubiquitous Computing. In these areas he has published numerous articles in journals, conference proceedings, books and book chapters. And he has been teaching at various universities across Europe. He has participated in and coordinated activities in various national and international research projects. He is the official representative of Germany in the IFIP Technical Committee on 'Human Computer Interaction' (TC.13). He has been conference co-chair and organiser of many international conferences (e.g., program co-chair of the ACM GROUP 2010 and INTERACT 2009 conference). Further information can be found at: http://www.tomgross.net.