COLLABORATIONBUS AQUA: EASY COOPERATIVE EDITING OF UBIQUITOUS ENVIRONMENTS

Maximilian Schirmer and Tom Gross
Bauhaus-University Weimar

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, we 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

Cooperative Ubiquitous Environments; Configuration; Editor; Sharing and Browsing.

1. 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 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 describe the concept and implementation of *CollaborationBus Aqua* and report on its user interaction. We then present related work.

2. COLLABORATIONBUS AQUA CONCEPT

The requirements for *CollaborationBus Aqua* emerge from our own experience of developing cooperative ubiquitous environments for many years, and lessons learned from related work such as the examples below. In this section we focus on the three core concepts of *CollaborationBus Aqua*.

2.1 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, 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 to infer 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.

2.2 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 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.

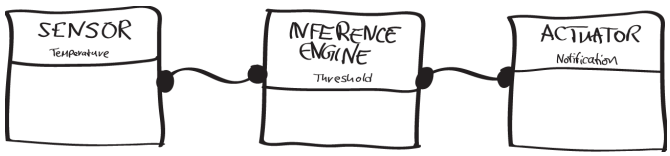


Figure 1. Scheme of configurations including example.

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), or connecting components with incompatible data types (e.g., connecting a temperature sensor with a Boolean inference engine).

2.3 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 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.

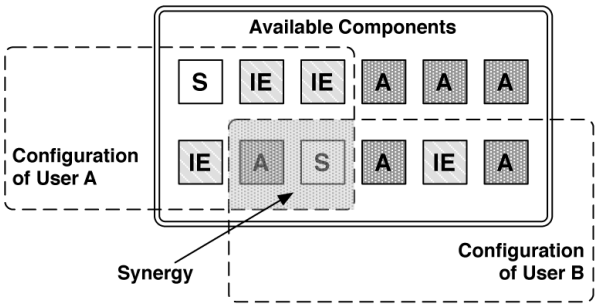


Figure 2. Synergies in shared cooperative ubiquitous environment configurations.

The underlying mechanism works as follows (cf. Figure 2): 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.

3. COLLABORATIONBUS AQUA IMPLEMENTATION

The *CollaborationBus Aqua* editor is a stand-alone application based on Java 1.5.0_13, MySQL 5.0.41, and 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 3). 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.

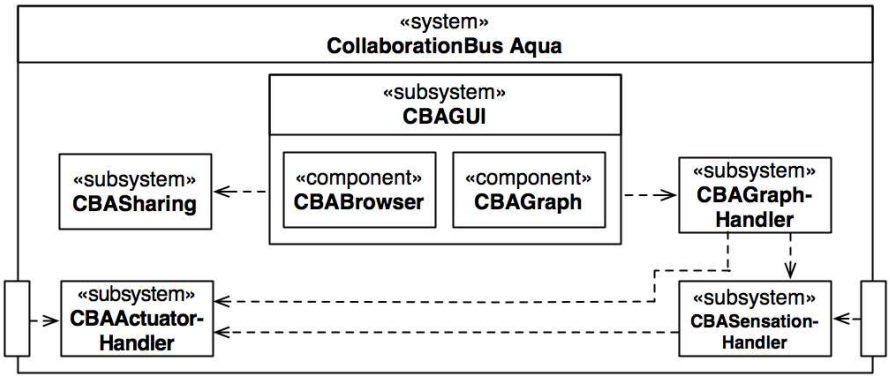


Figure 3. Component diagram of *CollaborationBus Aqua*.

Subsequently, we explain how the concepts from Section 2 are implemented with the five subsystems. Please note that *CollaborationBus Aqua* has been completely implemented and deployed; yet, due to page limitations we cannot provide a deployment diagram.

3.1 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 condition without any noticeable delay. The *CBASensationHandler* subsystem of *CollaborationBus Aqua* implements communication management and initiation. It encapsulates connections to various *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.

3.2 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.

3.3 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.

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.

4. 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.

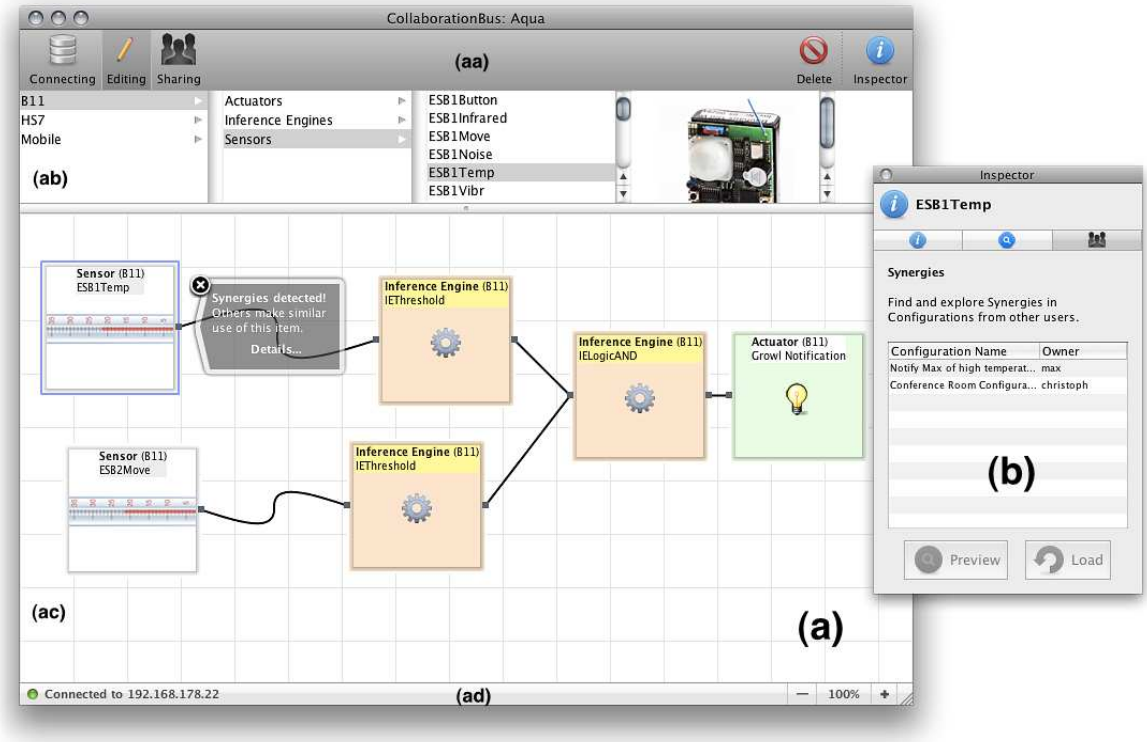


Figure 4. Graphical user interface of *CollaborationBus Aqua*, with (a) the Main Window, and (b) the Inspector.

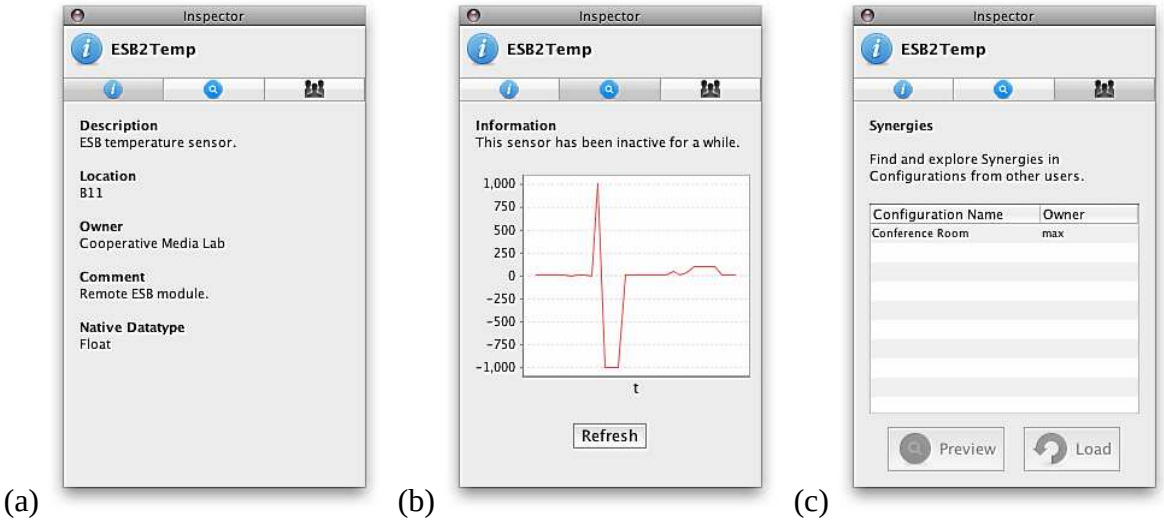
4.1 Connecting, Editing, and Sharing

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

4.2 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.



(a) (b) (c)

Figure 5. The operation modes of the inspector: (a) General information, (b) Recent events, and (c) Synergy browser.

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 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.

5. RELATED WORK

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.

6. CONCLUSIONS

Cooperative ubiquitous environments combine ubiquitous computing with the general aim of supporting collaboration and cooperation in a shared information space, as envisioned in 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.

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

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. *ws-xml-rpc - Apache XML-RPC.* http://ws.apache.org/xmlrpc/, 2010. (Accessed 2/3/2010).

Bannon, L.J. and Schmidt, K. CSCW: Four Characters in Search of a Context. In *Proceedings of the First European Conference on Computer-Supported Cooperative Work - ECSCW'89* (Sept. 13-15, Gatwick, UK). Elsevier, Dortrecht, NL, 1989. pp. 358-372.

Dey, A.K. and Newberger, A. Support for Context-Aware Intelligibility and Control. In *Proceedings of the Conference on Human Factors in Computing Systems - CHI 2009* (Apr. 4-9, Boston, MA). ACM, N.Y., 2009. pp. 859-868.

Greif, I. and Sarin, S. Data Sharing in Group Work. In *Proceedings of the 1986 ACM Conference on Computer-Supported Cooperative Work - CSCW '86* (Dec. 3-5, 1986, Austin, TX). ACM, N.Y., 1986. pp. 175-183.

Gross, T., Egla, T. and Marquardt, N. Sens-ation: A Service-Oriented Platform for Developing Sensor-Based Infrastructures. *International Journal of Internet Protocol Technology (IJIPT) 1*, 3 (2006). pp. 159-167.

Gross, T. and Marquardt, N. 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 - PDP 2007* (Feb. 7-9, Naples, Italy). IEEE Computer Society Press, Los Alamitos, 2007. pp. 307-314.

Humble, J., Crabtree, A., Hemmings, T., Akesson, K.-P., Koleva, B., Rodden, T. and Hansson, P. "Playing with the Bits" User-Configuration of Ubiquitous Domestic Environments. In *Proceedings of the Fifth International Conference on Ubiquitous Computing - UbiComp 2003* (Oct. 12-15, Seattle, WA). Springer Berlin/Heidelberg, 2003. pp. 256-263.

JGraph Ltd. *JGraph Home Page.* http://www.jgraph.com, 2009. (Accessed 2/3/2010).

Lim, B.Y. and Dey, A.K. Assessing Demand for Intelligibility in Context-Aware Applications. In *Proceedings of the 11th International Conference on Ubiquitous Computing – UbiComp 2009* (Sept. 30-Oct. 3, Orlando, FL). ACM, N.Y., 2009. pp. 195-204.

Mackay, W.E. Patterns of Sharing Customisable Software. In *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work - CSCW '90* (Oct. 7-10, Los Angeles, CA). ACM, N.Y., 1990. pp. 209-221.

Mavrommati, I., Kameas, A. and Markopoulos, P. An Editing Tool that Manages Device Associations in an In-Home Environment. *Personal and Ubiquitous Computing 8*, 3-4 (2004). pp. 255-263.

Myers, B.A. Visual Programming, Programming by Example, and Program Visualisation: A Taxonomy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI 1986* (Apr. 13-17, Boston, MA). ACM, N.Y., 1986. pp. 59-66.

Scripting News Inc. *XML-RPC Home Page.* http://www.xmlrpc.com/, 2010. (Accessed 2/3/2010).

Sohn, T. and Dey, A. iCap: an Informal Tool for Interactive Prototyping of Context-Aware Applications. In *Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2003* (Apr. 5-10, Fort Lauderdale, FL). ACM, N.Y., 2003. pp. 974-975.