Scalable Computing: Practice and Experience
Volume 11, Number 3, pp. 289–303. http://www.scpe.org
ISSN 1895-1767

CREATING, EDITING, AND SHARING COMPLEX UBIQUITOUS COMPUTING ENVIRONMENT CONFIGURATIONS WITH COLLABORATIONBUS

TOM GROSS* AND NICOLAI MARQUARDT†

Abstract. Early sensor-based infrastructures were often developed by experts with a thorough knowledge of base technology for sensing information, for processing the captured data, and for adapting the system's behaviour accordingly. In this paper we argue that also end-users should be able to configure Ubiquitous Computing environments. We introduce the CollaborationBus application: a graphical editor that provides abstractions from base technology and thereby allows multifarious users to configure Ubiquitous Computing environments. By composing pipelines users can easily specify the information flow from selected sensors via optional filters for processing the sensor data to actuators changing the system behaviour according to their wishes. Users can compose pipelines for both home and work environments. An integrated sharing mechanism allows them to share their own compositions, and to reuse and build upon others' compositions. Real-time visualisations help them understand how the information flows through their pipelines. In this paper we present the concept, implementation, and user interface of the CollaborationBus application.

Key words: ubiquitous computing; editor; configuration

1. Introduction. The development of early sensor-based infrastructures often required expert programmers with a thorough knowledge of base technology for sensing information, for processing the captured data, and for adapting the system's behaviour accordingly [10] [23] [24] [26]. In this paper we argue that also end-users should be able to configure Ubiquitous Computing environments. There are some research projects providing easy-to-use configuration interfaces for non-expert users to create sensor-based Ubiquitous Computing applications, yet mostly only for the private home [4] [8] [15] [18] [25]. Furthermore, most systems lack integrated facilities for the collaborative exchange of users' configurations. Only some systems—typically complex configuration tools [2] [3] [5] [16]—provide enhanced visualisations of the data flow and sensor-network data to support users while creating or configuring applications.

In this paper we introduce CollaborationBus: a graphical editor that provides adequate abstractions from base technology and thereby allows multifarious users—ranging from novice to experts—to easily configure complex Ubiquitous Computing environments.

By composing pipelines users can easily specify the information flows from selected sensors via optional filters for processing the sensor data to actuators changing the system behaviour according to their wishes. Whenever the sensors capture values that are in the range indicated by the users, the actuators perform the specified actions. All pipeline compositions are stored in the respective user's personal repository. A central interface allows users to control their respective repository—they can create new pipeline compositions, or edit, activate or deactivate existing ones.

An integrated sharing mechanism allows users to share their own pipeline compositions with others users. In an analogous manner they can add others' compositions to their own repository, and build new compositions based on these compositions. Real-time visualisations display relations between incoming and outgoing events of the pipeline, and let the user interactively adjust and keep track of the information flow through their pipelines. They help the users understand the information flow through their compositions, which can become quite complex consisting of sets of sensors, filters, and actuators.

In this paper we present the concept, implementation, and user interface of the CollaborationBus application. First, we develop scenarios of configurations for Ubiquitous Computing environments and derive requirements. Then we describe the concept and implementation of CollaborationBus, and present its user interface. We continue with a discussion of related work. Finally, we draw conclusions and report on future work.

2. Requirements. In this section we develop scenarios of configurations for Ubiquitous Computing environments and derive requirements for the CollaborationBus editor.

2.1. Application Scenarios. Users should be able to configure environments in their private homes as well as in their workplaces.

*Faculty of Media, Bauhaus-University Weimar, Germany (e-mail: tom.gross(at)medien.uni-weimar.de).
†Department of Computer Science, University of Calgary. Calgary, AB, CANADA, T2N 1N4. (e-mail: nicolai.marquardt(at)ucalgary.ca).

Smart Telephone. In a first scenario users wish to control the sound volume of their music players and start their calendar application in dependence of their office telephones' state. A simple binary sensor attached to the telephone is the first input source of this pipeline. The second input source checks whether the user is currently logged in at the office computer. The condition modules check the telephone sensor state as well as the login information. Finally, the user specifies the desired information flow: if the attached sensor detects that the phone is used, a script is started (e.g., AppleScript on a Mac OS X computer, or a shell script on Windows) and mutes the volume of the computer (e.g., Mac, or PC), an infrared control (e.g., on a sensor board) mutes the sound system, and another script starts the user's calendar application (e.g., iCal, or Outlook), so that the user can input new appointments during the phone call. When the phone call ends, the application fades the music back in again after a few seconds.

Personal Notification Selection. In a second scenario, users want to get information about the current activities of their remote co-workers and friends. Users can add a state sensor to the instant messaging application as well as movement and noise sensors as sources of their pipeline. Then users can specify queries with keyword filters that analyse the sensor data of the instant messaging sensor and check if they match the names of their remote co-workers or project descriptions. As actuators the users might wish to specify that all events are collected and sent as a daily email summary once a day. Additionally, if the number of messages containing the keywords reaches a specified occurrence threshold, the system additionally sends the users an immediate summary message to their mobile phones via an SMS gateway (a short message service sending a message to the mobile phone).

Informal Group Awareness. In a final scenario, the users of two remote labs want an information channel of the lab activities as RSS feed that can be integrated into tickertape displays or screensavers. They wish to receive information on the activities at the other site. They create a pipeline composition and add the following information sources as input sources: the current lab members logged in on the server and in the instant messaging system, the current CVS submissions of the developers, the average values of the movement and noise sensors and the current temperature of the two labs and the coffee lounge. As actuator component for the output they add an RSS feed generator and publish the RSS file to a server. Now, the lab members can access this RSS feed and add it to their favourite notification display (e.g., a Web browser, or a screensaver). This summary of group events and activities can help users to find out more about the whole development team, and can facilitate the informal and spontaneous communication between the colleagues.

2.2. Functional Requirements. The following functional requirements were derived from various application scenarios (we described three of them in the preceding sub-section), and from a detailed study of related work (we present some examples of related work in Section 6 below).

- *Provide adequate abstraction for various applications domains:* Configuration editors should allow users to integrate a variety of software and hardware sensors capturing information, and software and hardware actuators adapting the behaviour of the environment accordingly. The integration of existing and new sensors and actuators should be easy. Various configurations should be possible—ranging from configurations for home environments as well as for work environments.
- *Support diverse users with heterogeneous knowledge, ranging from novice to experts:* Configuration editors should facilitate the immediate utilisation. For this purpose, they should provide a pre-defined library of common configurations and configuration assistants that allow the users—especially novice users—to use the editor immediately and to incrementally explore its functionality. Additionally, configuration editors should offer guided compositions. Therefore, the user interface and the functionality provided should be restricted to significant and needed functions; functions that are not adequate or not needed should be disabled (e.g., if a sensor captures data in the form of text strings, calculations such as average should be disabled). Finally, configuration editors should provide details on demand. For this purpose, especially more experienced users should be able move from more abstract to more fine-grained layers, and to see and manipulate details.
- *Support the exchange of configurations among users:* Configuration editors should allow the sharing of configurations among users. The sharing of configurations is useful for workgroups and friends, because it allows users to build on the results of other users, and gives less experienced users the chance to benefit of the knowledge of more experienced users. Subsequently we present the concept and implementation of CollaborationBus addressing these functional requirements.

TABLE 3.1
Applications with sensors/actuators in home and work environments.

	Home actuators	**Office actuators**
Home sensors	*a) Smart Home Applications:* often connections between hardware sensors and actuators (e.g., to control electrical devices (power plugs) in dependence of observed sensor values, or to control multimedia home devices)	*b) Home Awareness Applications:* mixed use of software and hardware sensors and actuators (e.g., to observe the private home from the work office, or to display state of family members at home)
Office sensors	*c) Office Awareness Applications:* mixed use of software and hardware sensors (e.g., to summarise information of projects and to inform people at home about the working activity)	*d) Collaborative Work Applications:* often applications based on software sensors and actuators (e.g., to observe computer logins, instant messenger presence, and other activities)

3. Concept. In this section we describe CollaborationBus' key concepts for a generic approach, for pipelines, for a diverse user experience, and for collaborative sharing.

3.1. Generic Approach. The approach of CollaborationBus is generic—it works across multiple applications domains, temporal patterns, and complexity patterns.

3.1.1. Spanning Application Domains. Sensor- and actuator-based applications in the private home differ from those in the cooperative work domain. While we try to integrate a common, universal user interface and metaphors for users of both domains, these domains can vary in their use of hardware and software sensors as illustrated in Table 3.1.

Smart Home Applications (cf. *a* in Table 3.1) are mainly built with hardware sensors and actuators, where the developed sensor-based applications adapt the home environment automatically to the requirements of the private users. While computer applications provide appropriate functionality for the configuration and creation of these applications, the computer and its applications should disappear during the everyday execution of the sensor-based applications. In order to support the development of appropriate applications, the CollaborationBus editor supports a variety of hardware sensors and actuators, and the editor is only needed for composing the setting.

In contrast to these mainly hardware-based applications, most *Collaborative Work Applications* are based on both hardware and software sensors and actuators (cf. *d* in Table 3.1). Since computers are in general part of the workplace, software sensors and their events (e.g., appointments, emails, tasks, project activity) and software actuators (e.g., for sending emails, displaying messages on the computer screen) can be used to create sensor-based applications for awareness and information-flow of workgroups. At the same time, the integration of hardware—both sensors and actuators—and their physical user interfaces can facilitate the interaction with these applications. This results in tangible user interfaces for applications at the workplace (e.g., physical sliders so set the presence in an instant messaging systems; LCD displays for displaying important email messages; audio signals to inform about the current project's state). CollaborationBus supports the creation and configuration of all these free combinations of physical user interfaces with software events as a main feature and allows users to create their envisioned interfaces themselves.

In between these two domains are applications that bridge the gap between the private home and the business work (cf. *b* and *c* in Table 3.1). *Home Awareness Applications* (cf. *b* in Table 3.1) support connections to family members and friends at the workplace. For instance, ambient displays let the users perceive the information in multi-sensory ways. This includes that users can configure their sensor-based applications at home as well as at their office; thus a universal application interface is required.

Office Awareness Applications bridge the gap between the home and the work environment (cf. *c* in Table 3.1) by informing users about events from the office while they are at home. Users define their own information channels that connect home environment with their work environment (e.g., project report summaries that are generated and delivered to the private home, important email or instant messages that are forwarded to the private home). Here, the configuration editor requires in most cases a variety of software sensors in the work environment that are connected to physical actuators in the private home.

On a whole both environments—home and work—have become increasingly intertwined in the recent years (e.g., telework). Therefore, utilities need to allow the building of universal sensor-based applications spanning both ambiences and the integration of software sensors and actuators as well as hardware sensors are needed.

3.1.2. Spanning Temporal Patterns. In any application domain various patterns with regard to capturing ongoing data and starting actuators can be identified:

- Recurrent, permanent (e.g., applications with ongoing collection of data)
- Recurrent, occasionally (e.g., applications depending on day-time, during the holidays, at night)
- One-time (e.g., applications with call-back if the required person is reachable)

The software needs adequate methods to support any of these temporal patterns, and should provide a structured overview of the current configurations of a user. Another important aspect is to enable the easy re-use of created configurations in the past: a copy method and templates can speed up the creation process. Systems supporting all these temporal patterns are needed.

3.1.3. Spanning Complexity Patterns. Each setting can have a specific complexity pattern ranging from simple sensor-actuator tuples to networks of sensors and actuators:

- One sensor, one actuator (e.g., one binary sensor controls one actuator)
- Sensor, filters, actuator (e.g., only react to certain temperature values of a temperature sensor)
- Multiple parallel sensors, filters and actuators (e.g., create summaries of various sensor sources, control a set of actuators)
- Complex network of components (e.g., determine the current activity or even mood of a person)

The CollaborationBus editor supports any application domain, and any temporal pattern described above. It supports any complexity pattern, except for complex networks. Complex networks are typically not configured with a graphical editor, but rather developed with programming languages; therefore, here a graphical editor would not be used anyways.

3.2. Pipelines. In CollaborationBus all relations between sensors and actuators are handled with a pipeline metaphor.

Pipelines are compositions that include several components: at least one sensor and one actuator component, and additionally further filter components for processing sensor values (e.g., to delimit the forwarded values, or to convert data formats). All components inside of a composition are connected via pipelines that forward events between them. Pipelines can be nested in various ways: several parallel sub-pipelines can be added (this represents the OR condition); sequences of sensor sources can be created (AND condition); or negations can be specified (NOT condition).

Sensors are the sources of any initial event in a pipeline. They can either be hardware sensors (e.g., sensors for temperature, movement, light intensity) or software sensors (e.g., sensors for unread emails, mouse activity, shared workspace events, open applications).

Actuators are at the sink-side of the pipeline composition. Hardware actuators affect the real environment of the users (e.g., activate light sources or devices), while software actuators only influence the computer system (e.g., display screen messages, start applications).

Filters for processing the captured data are between sensors at the one side and actuators at the other. The filter components can process all incoming events of a sensor source. Each filter component represents a single condition or transformation based on the incoming event value. Filters typically generate data of particular formats (e.g., integer values, Boolean values, strings). There are universal filter types that can be applied to any type of sensor data and specific filter types that can only be applied to particular types of sensor data. The respective filter types can do the following processing:

- Universal (e.g., count the event occurrence, create event summary reports)
- Numerical processing (e.g., numeric threshold, interpolation, average)
- String processing (e.g., search for specified keywords)
- Binary processing (e.g., negation, conjunction)
- Transformations (e.g., numeric value to string message, binary value to numeric)

Filters can be assembled in many different combinations. This includes, for example, an adaptive behaviour to changed conditions of the sensor sources (e.g., modified upper or lower limit of a temperature sensor, or a changed scale of values) by transmitting these changed conditions to all pipeline components. Each component can decide if a modification of its settings is necessary, and eventually display a confirmation dialog. The

components also include a variety of transformation methods (e.g., for generating a short message to the mobile phone (SMS) a string message can be entered, and the values of the respective sensors can be attached).

With CollaborationBus users can easily connect local sensors and actuators or sensors and actuators from remote locations and build new configurations in a few seconds by visual programming though point-and-click. Each pipeline composition includes all these components—sensors, actuators, and filters—and defines a complete information flow through them. Experts can program new pipeline components by deriving new classes from the *PipelineComponent* class (cf. next section for details). All compositions of a user are stored in a personal repository. This repository includes all data to dynamically instantiate the included pipeline compositions.

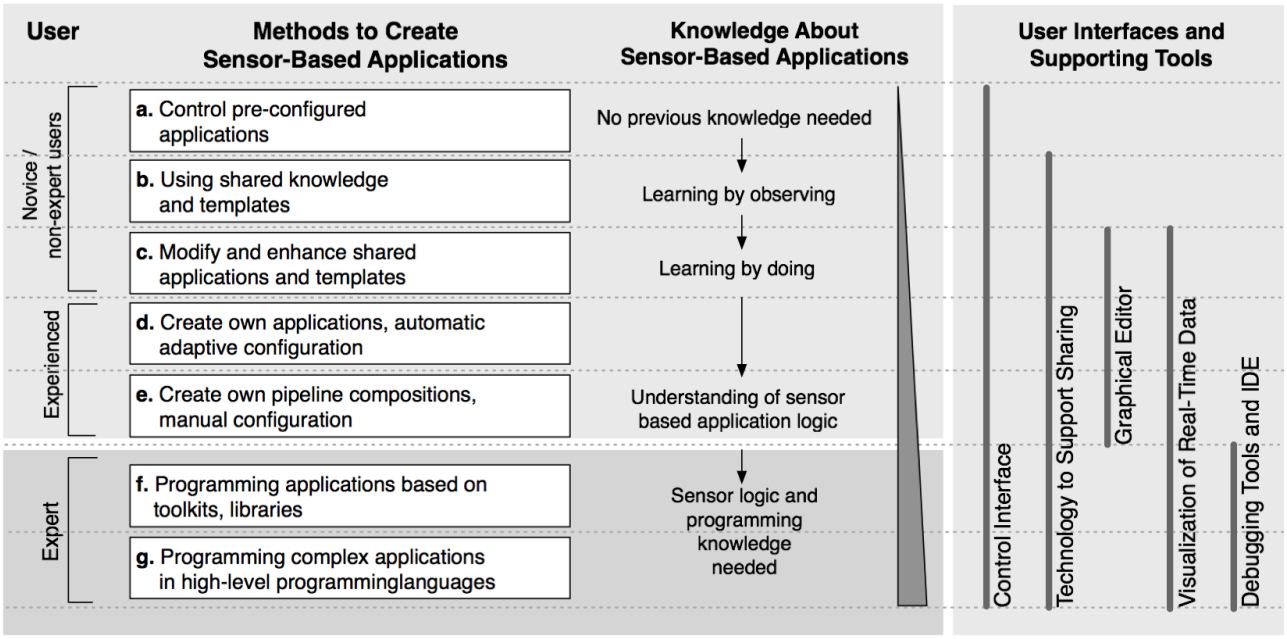

Fig. 3.1. *User experience levels, and adequate tools to support users.*

3.3. Experience for Diverse Users. The CollaborationBus editor can be used by users with diverse levels of technical background. Users' knowledge can range from no experience at all to very thorough technical knowledge. Figure 3.1 shows various user types ranging from novice users with no experience to more experienced users and to experts. It shows the methods that are available and can be used in dependence of the existing knowledge. It also shows the user interfaces and support tools that are offered for the respective user types (the user interfaces and supporting tools are described below).

Novice users with no prior technical knowledge can start using CollaborationBus by loading and adapting pre-configured application configurations that are part of the CollaborationBus distribution. As they progress, they can use the integrated sharing tool to load other users' configurations and to use them as templates for their own configurations. They can, furthermore, modify and enhance the application configurations and templates.

More experienced users can create their own application configurations, and execute them in order to learn more about intra-pipeline event forwarding.

Expert users can create the envisioned system-behaviour by developing the required software in a high-level programming language. Typically, for these activities they use toolkits, platforms, libraries, and development and debugging environments to facilitate and speed up the development process.

Taking these diverse user types into consideration is a core concept of CollaborationBus and its user interface (the latter is described below).

3.4. Collaborative Sharing. Users can build their own personal pipeline compositions from scratch, or build on shared compositions from colleagues and friends. Three types of sharing are possible:

- *Sensor and event sharing:* users either share the events of their own sensors, or the processed events of their sensors.
- *Actuator sharing:* users share the control of a personal actuator with other users, so that other users can send commands to the actuator and control the system behaviour.
- *Pipeline sharing:* users share complete more or less complex pipeline compositions with others.

The first sharing method lets users create their own configuration in dependence of remote located sensors of other users. The second sharing method lets users control the actuators of other users (leading to new challenges of potentially concurrent access to actuators). And the third sharing method lets users exchange and re-instantiate complete pipeline compositions, requiring a unified description format and exchange protocol for pipeline compositions. In the latter case the recipients of the compositions can change this released pipeline composition to fit to their requirements. Because each user creates a new instance of this pipeline composition, the changes of other users are not affecting the original composition.

CollaborationBus supports security and privacy protection thought adequate levels of abstraction and control over access privileges of the own information sources are needed. In order to restrict the shared information, users can choose the sharing of abstract templates. In these shared pipeline compositions, only the skeleton of a pipeline is shared, and the original sensors and actuators of a user are not included in the sharing entry. Thus, the abstract template of a composition contains mainly the configuration of all filter components between the sensors and actuators. Using this abstract template, other users can insert their own sensors in the placeholders at the beginning of the pipeline composition, and their own actuators at the end. This let them use the knowledge of the processing filter components of the composition, while at the same the user who shares his pipeline composition does not share his own sensors and actuators.

These integrated collaborative sharing methods provide a powerful and easy-to-use method of knowledge exchange between different users of the system. As prior described in Figure 3.1, a novice and inexperienced user can use pre-configured pipeline compositions of another user (if this user shares the complete pipeline), or the user can load an abstract pipeline template and fill in his own sensors and actuators. At every time it is very easy for the users to share their new pipeline compositions again, and store them in the shared repository.

The following example illustrates a situation where these abstract templates are appropriate. A user has created an ambient notification display of important incoming email messages: all messages are scanned for adequate keywords or sender addresses, and if the scan was successful, a message will be displayed on an ambient external LC display. The user decides to share this configuration, while at the same time it stands to reason that the user do not want to share his personal email-sensor, or the exact configuration of the keyword filter. By using the abstract template, the user can share the basic concatenation of incoming sensors, filters, and the actuator display, without sharing his personal sensors.

On the other hand, a user who has created a SMS notification service for the average temperature of a series of temperature sensors may wish to share this complete configuration, and therefore shares the pipeline compositions with all the associated sensors.

4. Implementation. In this section we describe the implementation of CollaborationBus: software architecture and class diagram.

4.1. CollaborationBus Software Architecture. Figure 4.1 provides an overview of the software architecture of CollaborationBus. All sensor and actuator components are connected to the *SensBase* infrastructure, which provides adapters for the connection of sensors and actuators, a central registry of all connected components and a database for persistent storage of sensor event data. *SensBase* was implemented with the *Sens-ation* platform [13]. *SensBase* provides inference engines that can transform, interpret, and aggregate sensor values. A variety of gateways (e.g., Web Service, XML-RPC, Sockets) provide interfaces for the retrieval of sensor descriptions, event data, actuators, and so forth.

The *CBServer* uses these gateways to register for the sensor values needed for the users' pipeline compositions. Each time when changes occur at one of the connected sensors, the *SensBase* server forwards a change event to the *CBServer*. These events are forwarded to the adequate components inside of each pipeline composition. The compositions are inside of the Personal Repository of each user and include the complete description of all assembled components (in serialised XML format, for platform independency and easy exchange of pipeline composition descriptions). The *CBServer* can serialise and de-serialise these XML descriptions, and validate and process these descriptions. If a XML description of a pipeline composition is de-serialized, the *CBServer* creates instances of proxy objects for each of the pipeline components (sensors, filter, actuators).

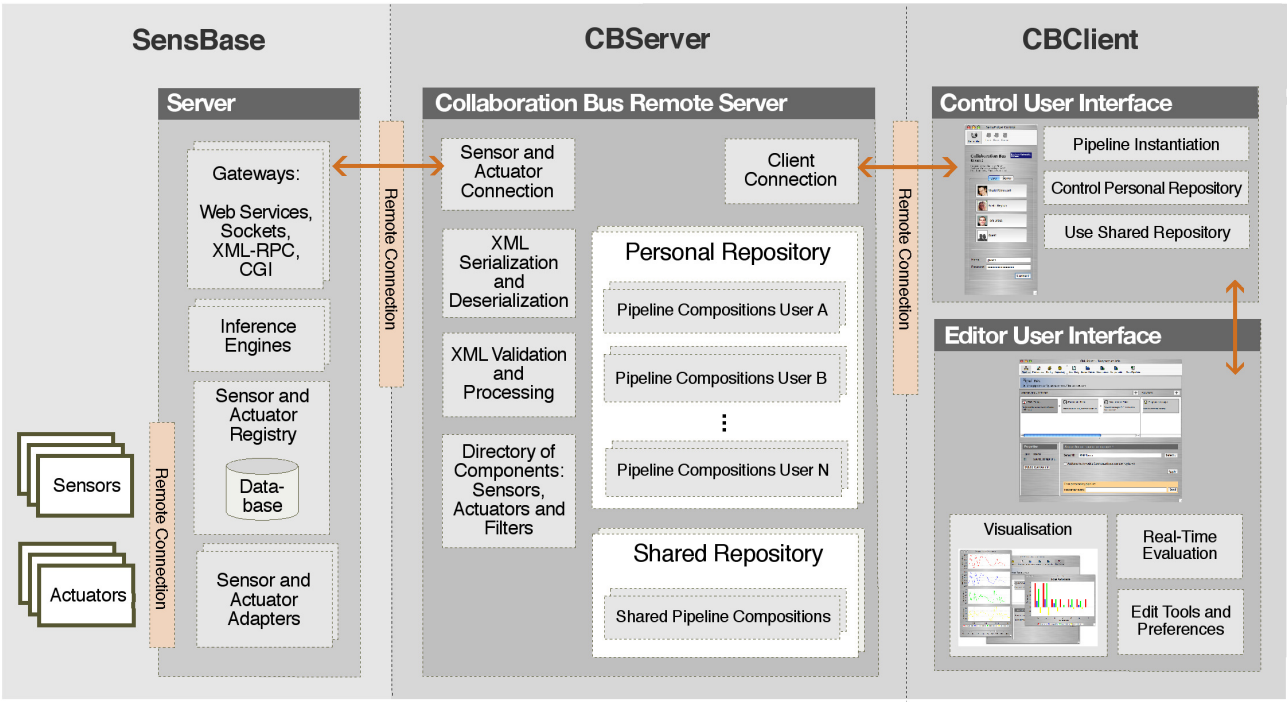

Fig. 4.1. CollaborationBus *software architecture.*

Inside of these components we have multiple threads running to ensure rapid processing of data as well as rapid forwarding of events to the subsequent component. While the sensor and actuator components inside of these compositions act as a proxy for the existing (and connected) devices, the filter objects represent the processing and transformation part inside of these compositions. A pipeline composition can include multiple processing pipelines simultaneously, and the users can run and stop as many of these compositions as they like (by using the Control User Interface).

In the Shared Repository the published pipeline compositions are stored. They are saved in the XML format as well, and XML processing is used to operate based on these descriptions (e.g., to modify existing entries, or to create an abstract pipeline composition template). Furthermore, the *CBServer* manages a directory of all the various sensor and actuator types, as well as filter components, and submits them to the client application. The dynamic directory can be extended with new components at any time, and this ensures the easy extendibility of CollaborationBus. If users want to integrate different actuators or sensors, they need to implement a new adapter driver at the *SensBase* level; this is independent from the CollaborationBus architecture. However, if new filter components are needed for a different data processing, then a new class (by deriving from an abstract base class with the core functionality of each filter component) is needed to represent this processing step. While this can be done with minor effort by any software developer, it is not easy to add a new filter for non-programmers.

The *CBClient* implements the GUIs described above. For creating, controlling and editing pipeline compositions it is necessary to support all the XML operations of the server, and the methods for instantiating pipeline compositions as well (for the editor and testing tools).

4.2. CollaborationBus Class Diagram. The class structure of the repositories and pipeline compositions is illustrated in an UML class diagram in Figure 4.2. The *PersonalRepository* class provides methods to add, remove, modify, and get *PipelineComposition* objects. The *SharedRepository* contains a collection of *SharedRepositoryEntries*, which wraps one *PipelineComposition* and specify the sharing attributes of this *PipelineComposition* (e.g., abstract or complete template).

The *PipelineComposition* object is a composite object for a series of *PipelineComponents*. It encapsulates methods for controlling pipeline compositions (e.g., start and stop), and for adding and removing pipeline components. *PipelineComponent* is the abstract base class for the *Sensor*, *Filter*, and *Actuator* base classes.

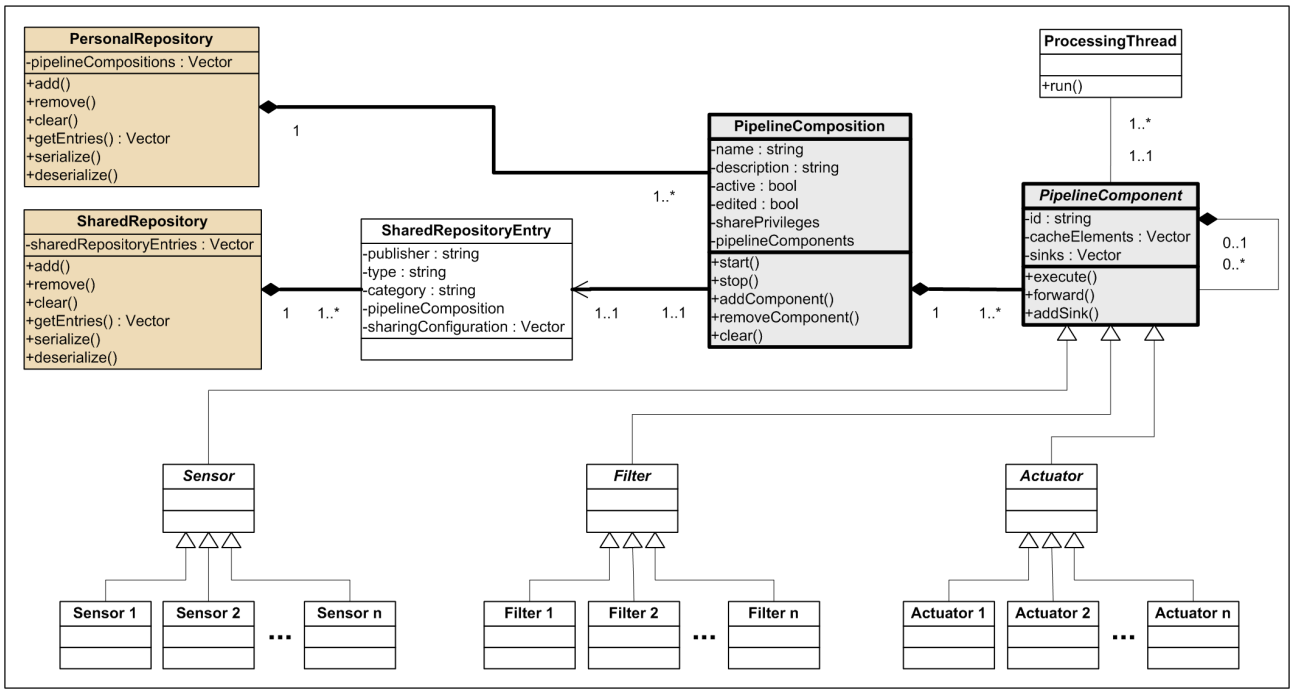

Fig. 4.2. CollaborationBus *repository and pipeline UML class diagram.*

It provides common methods for each pipeline component (like processing, forwarding and caching of events). Inside the *PipelineComponents* multiple threads (*ProcessingThread*) are running to ensure rapid processing of data as well as rapid forwarding of events to the subsequent component. *Sensor*, *Filter*, and *Actuator* are abstract base classes for the concrete pipeline components allowing to respectively: retrieve sensor values from any number of sensors from the SensBase infrastructure and push them into the pipeline process (e.g., sensor values from the Embedded Sensor Board or Phidgets hardware devices [11]); process incoming values (e.g., keywords, average, or threshold filter); and control the actuator elements (e.g., generate an RSS feed, show a message on a text display, or drive other applications via AppleScript). CollaborationBus is implemented in Java with Swing libraries for the GUIs. Several libraries are used for XML [30] processing (e.g., for the serialisation of pipeline compositions [27], for parsing sensor descriptions, for creating XPath expressions [29]); and for remote connections (e.g., XML-RPC [28], and SOAP [1]).

5. User Interface. The CollaborationBus editor provides four major graphical user interface (GUI) components: the Login and Control GUI; the Editor GUI; the Shared Repository GUI; and the Real-Time Visualisation GUI.

5.1. Login and Control GUI. The Control GUI is the central access point for all users to their personal repository of configurations. In order to get to their Control GUI, users have to login first. Figure 5.1 shows the Login and the Control GUIs.

After login, users can see the Control GUI with the listing of their pipeline compositions, including an indicator of the current state of each pipeline composition (rectangle to the right of the pipeline name): *Off* (grey), *Running* (green), or *In Edit Mode* (orange).

All functions for modifying the repository and its compositions are available from within this interface: Add, Remove, Rename, and Clone pipeline compositions (via the Commands button). Users can Start and Stop the threaded execution of each composition (via the Start/Stop button). And, they can use the Share method to upload the selected composition directly to the shared repository (via the Commands button).

5.2. Editor GUI. While the basic functions for the personal repository are available in the Control GUI, the underlying filter composition of each of the pipelines is only available in the Editor GUI that can be opened for each of the pipeline compositions. Figure 5.2 shows the Editor GUI. In the top area the user can choose several buttons for loading the Pipelines (via the Pipelines button), change the Preferences (via the Preferences

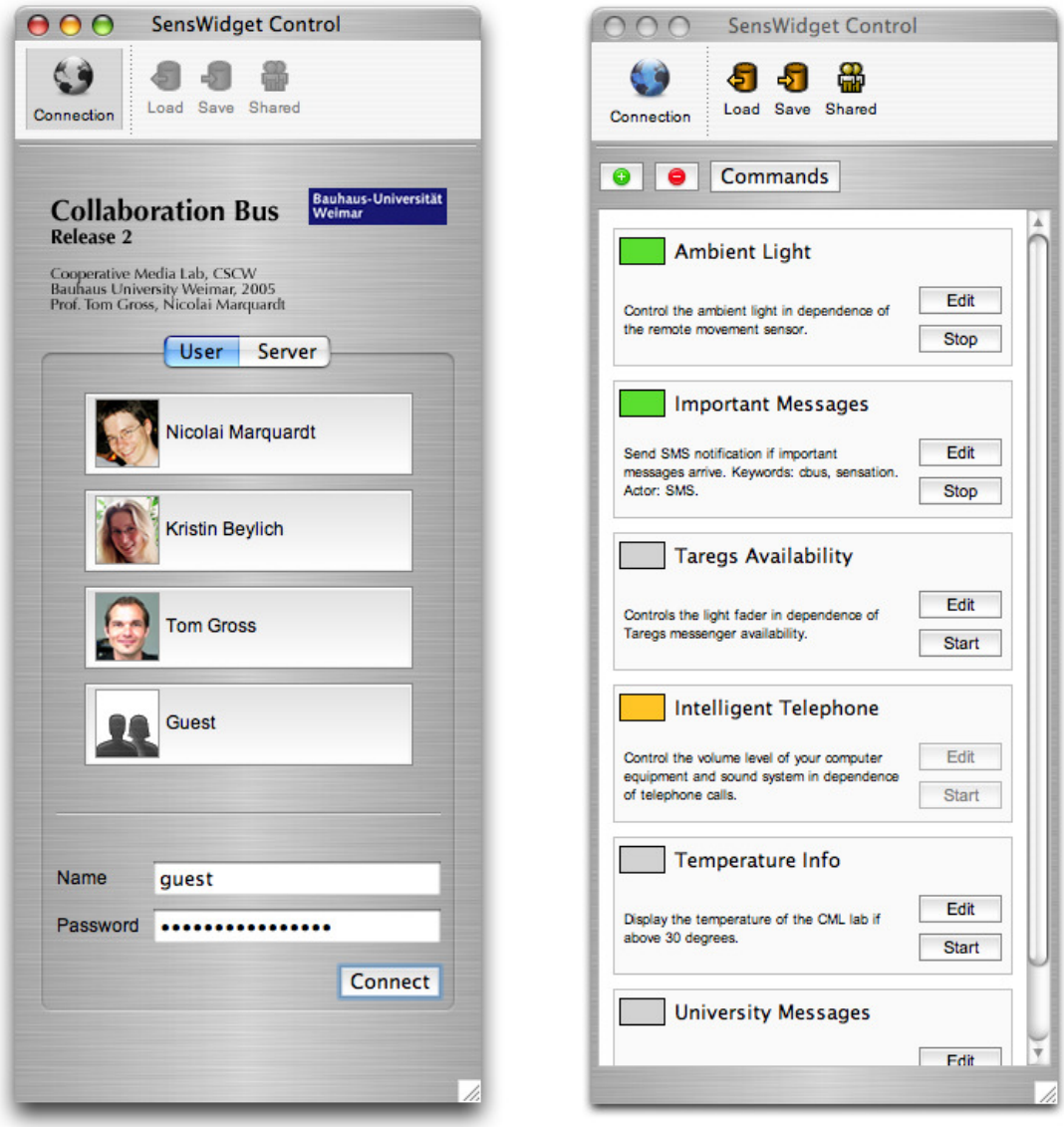

FIG. 5.1. *Login GUI and Control GUI.*

button), etc. In the middle area the respective pipeline with its sensors, conditions, and actuators is shown (each individual item is represented as a rectangular box). In the bottom area the properties of the currently selected pipeline part (rectangular box) are shown and can be altered.

In order to create a new pipeline composition, users can first discover the available sensor sources (e.g., movement sensor, temperature, sensor telephone sensor, instant messenger status sensor) of the infrastructure in a graphical sensor browser (the browser can be started by pressing the +-sign to the right of *Sensors and Conditions*), and add the sensors they need to the pipeline. Then they can specify rules and conditions (these can also be viewed by pressing the +-sign to the right of *Sensors and Conditions*) for the sensor values by adding sets of filters and operators. For each sensor types with the according sensor value type, specific filters and operators can be selected (e.g., an event value threshold, a counter for number of occurrences). Finally, the actuators can be specified by selecting them in the graphical actuator browser (the browser can be started by pressing the +-sign to the right of *Actuators*). Here, the editor provides the option to specify the mapping between the pipeline output and the actuator commands (e.g., if the pipeline output is a message, it can be displayed; if the pipeline output is a simple temperature value, the corresponding sound volume can be set).

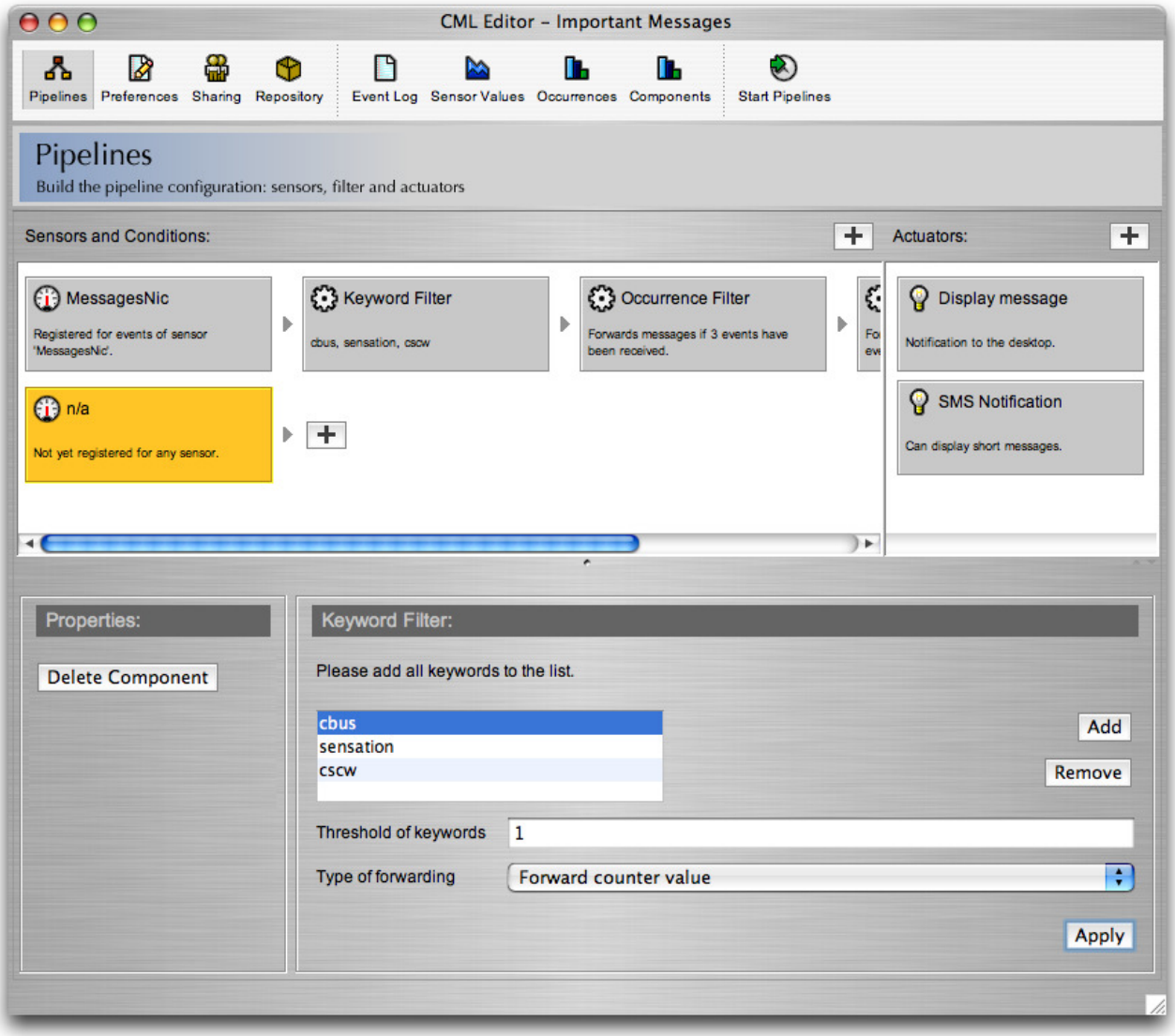

Fig. 5.2. *Editor GUI.*

5.3. Shared Repository GUI. The collaborative sharing mechanism described above is integrated in the Control GUI and in the Editor GUI. In order to make a pipeline composition available for others, users have two options. They can either select the Share method in the Control GUI (via the Shared button; cf. Figure 5.1). Here the default settings for sharing are used and no additional parameters are needed. Or they can choose the Sharing command in the Editor GUI (via the Sharing button; cf. the top area in Figure 5.2) to specify further settings for the shared composition. Further settings include description, category, and type of sharing (cf. three types of sharing above). Finally the users can upload the pipeline composition.

In order to use one of the shared pipeline compositions, the user can access the Shared Repository GUI from within the Control GUI. Figure 5.3 shows the Shared Repository GUI. By selecting one of the available compositions in the list at the left side, the information for this entry is displayed at the right side of the dialogue (description, owner, category, type of sharing, used sensor sources and actuators). Users can then download the respective composition.

5.4. Real-Time Visualisation GUI. In the assembly of pipeline compositions with a variety of components it can be difficult to keep track of the intra-pipeline communication between the components and the processing of the forwarded pipeline events. The Real-Time Visualisation GUI of the CollaborationBus

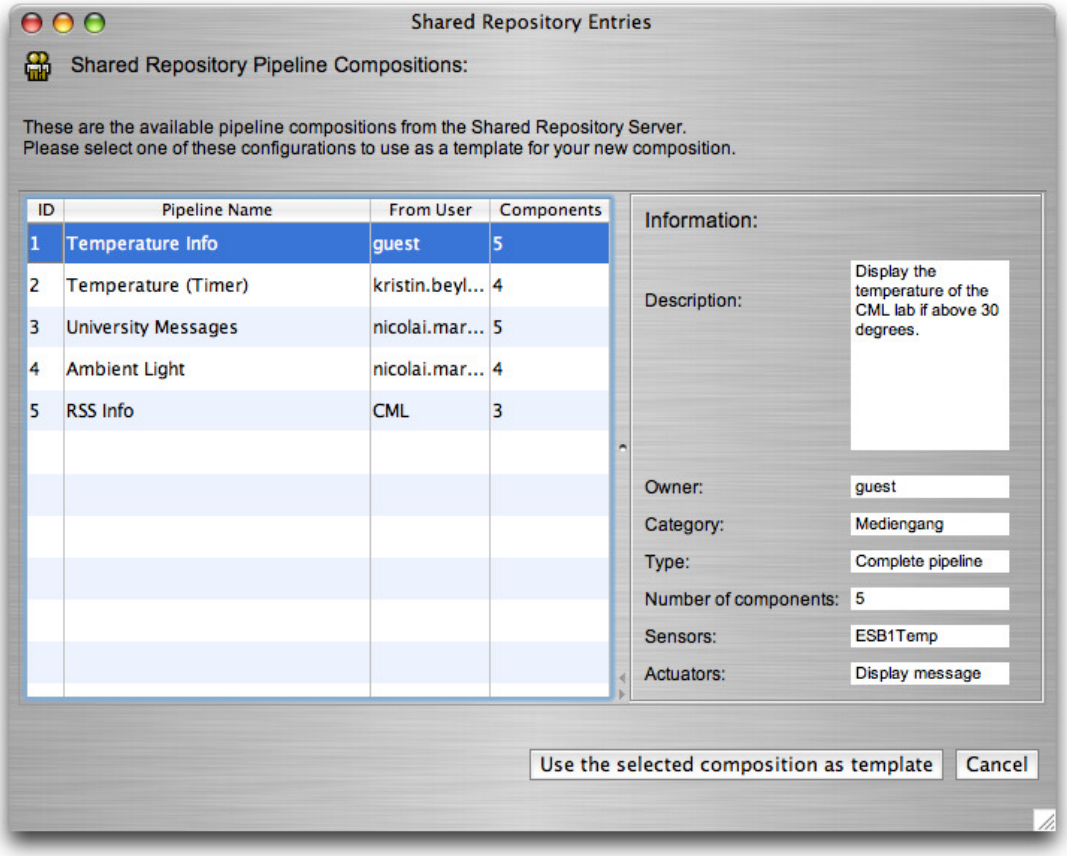

FIG. 5.3. *Shared Repository GUI.*

provides a variety of graph visualisations that can either display the forwarded values of each component of the pipeline (e.g., useful for interpolation and threshold filters) or the quantity of forwarded values (e.g., useful for gate filters, counters or timers).

Figure 5.4 shows the Real-Time Visualisation GUI with a time plot visualisation on the left (showing the absolute values of 4 temperature sensors), and an overview of the pipeline events on the right (showing the number of occurrences of events in a specific pipeline).

With these visualisations, the user obtains an inside view of the pipeline processing. The command Start Pipelines (via the Start Pipelines button) activates all components of the respective pipeline(s) and registers for the respective sensor events, starts the processing of threads, prepares the actuator modules, and generates and dynamically updates the visualisations. When any of the components of a pipeline is changed (e.g., a threshold, or an interpolation settings), the implication to the processing can be recognised immediately. Thus the adjustment and fine-tuning of component parameters becomes easier. In order to enable the testing of pipeline composition, we have, furthermore, integrated an input interface for simulated sensor events. It allows the users to manually insert sensor values to test and verify the pipeline composition without having to wait for real sensor values from the sensors. So, the processing of the data though the whole pipeline can be simulated.

6. Related Work. This chapter gives an overview of research related to the composition of sensor- and actuator-based applications. We introduce examples of programming tools for Ubiquitous Computing applications, software for controlling sensor networks, and collaborative sharing between users.

6.1. Programming Ubiquitous Computing Applications. Several research projects address the challenge to allow end-users to create and configure intelligent applications for in-home environments. With *iCAP*, Sohn and Dey introduce an application that allows end-users to rapidly prototype Ubiquitous Computing applications [25]. Similar to CollaborationBus, it uses rule-based conditions; especially the disjunction and

FIG. 5.4. *Real-Time Visualisation GUI (with time plot visualization, and overview of the pipeline events).*

junction of rules in their *sheets* is similar to our parallel and sequential pipelines (yet we think that workflow-adapted pipelines stimulate a better understanding of rule compositions than free arrangements). *iCAP* does not support sharing, or real-time visualisations.

Irene Mavrommati et al. have introduced an editing tool for creating device associations in an in-home environment [18]. Their editor connects various components called *e-Gadgets* to realise Ubiquitous Computing scenarios at home (similar to our connected processing components). Yet, it does not support workplace environments. The jigsaw editor of Jan Humble et al. [15] [22] demonstrates another application for getting control over the technological home environment. The metaphor of specifying the applications' behaviour by assembling pieces of a jigsaw puzzle sounds intuitive. Yet, we would like to give the users more control over their application than the encapsulated jigsaw pieces allow.

Some systems are based on mobile devices to control configurations from every location at every time. This includes systems for PDAs [18], mobile phones [4], and TabletPCs [15]. These mobile systems often provide only limited access to complex configuration methods. We have not created a version for mobile devices yet, but a lightweight mobile version of CollaborationBus would certainly be highly complementary to the existing version.

Another approach for configuring Ubiquitous Computing environments is programming by demonstration. This method requires an extended period of observation of relevant sensor values. In a later definition and learning phase, the users specify relevant sensor events in the event timeline, so that algorithms from artificial intelligence can detect patterns in the observed sensor values and automatically execute desired actuators [8]. Programming by demonstration tools hide most specific details of the underlying mechanisms from the users. On the one hand this reduces the barrier for non-technical users to configure Ubiquitous Computing environments, but on the other hand restricts the influence and control methods for users.

The related work applications mentioned so far address the development of complete sensor-based applications in a rather abstract way. In the *eBlocks* project [6] [7] a user interface for building sensor-based applications and configuring Boolean condition tables is introduced. As the authors show in their evaluation, users still need support in building these Boolean tables (e.g., support by different colours or written text [6]). Therefore, we introduced pipelines to allow the easy combination of Boolean AND, OR, and NOT conditions, simply by adding components to a pipeline processing stream or by adding a new parallel pipeline.

The *Phidgets* toolkit created at the GroupLab by Greenberg and Fitchett [11] facilitates the development of physical user interfaces. It provides a range of sensor and actuator elements as building blocks for letting developers rapidly prototype sensor-based applications. The included developers' toolkit allows easy access to these hardware components from within the software. This approach was further extended to distributed architectures by Marquardt and Greenberg [17]. In summary, the use of Phidgets requires few hardware skills, but considerable programming knowledge and is therefore not suitable for end-users.

6.2. Sensor Network Composition Software. A variety of applications for the compositions of sensor-based networks is available [3] [21]. For instance, the VisualSense modelling and simulation framework as part of the PTOLEMY II project [2] [3] is a toolkit for the control over fine granular sensor network communication and processing. The GUI includes functionality for processing component assembly, and for graph visualisations to display the processed values of components.

Since the evaluation of the communication in sensor networks can be difficult for newly created applications, several special complex development environments have been presented (e.g., SensorSim [21], EmTOS [9], TinyDB [16], and J-Sim [24]). These tools provide adequate development environments for expert users (because they include programming languages, operator sets, mathematical processing libraries, visualisation tools, etc.). The integration of visualisations for the event flow inside of sensor-network arrangements is interesting for our purpose [5]. However, users with a non-technical background probably have difficulties in using these applications. Furthermore, these latter environment do not support the sharing of development configurations.

6.3. Collaborative Sharing. While in Computer-Supported Cooperative Work (CSCW) collaborative sharing of location information, files, workspaces, software and patterns is wide-spread [12], an approach to sharing sensor- and actuator-based applications among users is still missing. In [12] design issues of CSCW applications that use data sharing are examined. This includes proposals for access control, adding meta-information, version history, and methods for handling updates and concurrency difficulties. Further common classifications of sharing between users are described in [19] [20]. They have found common groups with similar sharing preferences, and patterns in the sharing behaviour of users. Integrating support for these clustered groups could facilitate the usage of sharing mechanisms.

Hilbert and Trevor describe the importance of personalisation as well as shared devices for Ubiquitous Computing applications [14]. With the modification of applications to the personal needs, the use of these applications becomes easier for users.

7. Conclusion. In this paper we have introduced the CollaborationBus editor allowing any users to create sensor-actuator relations.

7.1. Summary. Even novice users can easily specify complex Ubiquitous Computing environments with the CollaborationBus editor, without having to deal with complex configuration settings or programming details. The CollaborationBus editor provides novel abstractions by encapsulating and hiding the details of the underlying base technology (e.g., the sensor infrastructure, the sensor and actuator registration, the sensor event registration). At the same time, more experienced users can control the pipeline composition configuration in any technical detail they need and get details on demand.

Furthermore, users can share their pipeline compositions with colleagues and friends via a shared repository. Users can also decide how accurate they want to share (e.g., complete compositions, abstract template, only the processed event value). With a minimum effort, each user can browse the shared repository and download shared pipeline compositions and adapt the used shared repository template to fit to their needs (by specifying their own personal properties of the pipeline). This way the CollaborationBus features an incrementally growing library of ready-to-use pipeline compositions that form a diverse network of collaborative sensor-actuator-relations.

7.2. Evaluation. While the evaluation of the CollaborationBus GUI and functionality as well as the produced pipeline compositions is of vital interest to us, a formal user evaluation is still missing. Nevertheless, we have collected several user opinions at the public demonstration of CollaborationBus to many visitors at the Cooperative Media Lab Open House 2005 from 14 to 17 July 2005, where the visitors had the chance to try out the CollaborationBus software in detail (with a huge set of connected sensors and actuators).

Most of the visitors quickly started to create their own compositions, and to select desired sensors, actuators and filters. At the same time, they hesitated to change the configuration of the filter components, and were somehow not completely confident about whether they change the right parameters. A helpful support in this

case was the Real-Time Visualisation GUI; in particular, the activation of the graph views of all pipeline events. It supported users in understanding the effect of parameter changes.

The most popular function of the tool was the integrated sharing mechanism. Users enjoyed browsing the large set of ready-to-use pipeline compositions in the shared repository. Often they used one of the shared compositions as template, modified parameters in the compositions or built a new configuration on the basis of this composition and sometimes shared this composition again. They also liked the idea of sharing their own compositions with others.

A typical barrier of users when creating sensor-based applications with CollaborationBus was that they worried about privacy issues. Many of the visitors said that it is an important criterion influencing their decision to use such as systems was to exactly know all outgoing or shared personal data and to be able to quickly and easily change the settings.

7.3. Future Work. Currently all components of the CollaborationBus system presented in this paper have been implemented. In the future we would like to evaluate the created pipeline compositions of users (especially those in the shared repository), and identify common patterns in the created compositions. From that we would like to develop assistive functions that provide users suggestions for reasonable compositions. The configuration interface of the filter components in the Editor GUI can also be improved to become more intuitive for the user. A graphical mapping could allow users to drag and drop the desired input and output commands and the component configuration.

A final important aspect related to security and privacy is the introduction of a system-wide authorisation and authentication system in order to further secure the access to the sensor values and pipeline compositions. For this purpose the CollaborationBus repository storage and the sensor value access could be integrated in the security system of the Sens-ation platform.

8. Acknowledgement. We would like to thank all members of the Cooperative Media Lab—especially Tareg Egla, and Christoph Oemig—for inspiring discussions on CollaborationBus, and for providing the PRIMI and Sens-ation platforms. Thanks to the anonymous reviewers for valuable comments.

REFERENCES

[1] Apache Software Foundation, *WebServices - Axis Architecture Guide. http://ws.apache.org/axis/java/architecture-guide.html*, 2005. (Accessed 5/5/2010).

[2] P. Baldwin, S. Kohli, E. Lee, X. Liu, and Y. Zhao, *Modelling of sensor nets in ptolemy ii*, in Proceedings of the Third International Symposium on Information Processing in Sensor Networks - IPSN 2004 (Apr. 27-27, Berkeley, CA), New York, NY, USA, 2004, ACM Press, pp. 359–368.

[3] ———, *Visualsense - Visual Modeling for Wireless and Sensor Network Systems*, tech. report, Report Number: UCB ERL Memorandum UCB/ERL M04/8, Ptolemy Project, 2004.

[4] L. Barkhuus and A. Vallgarda, *Smart Home in Your Pocket*, in Interactive Poster: Presented at The Fifth International Conference on Ubiquitous Computing - UbiComp 2003 (Oct. 12-15, Seattle, WA)., 2003.

[5] C. Buschmann, D. Pfisterer, S. Fischer, S. Fekete, and A. Kroeller, *SpyGlass: A Wireless Sensor Network Visualiser*, SIGBED Review 2, 1, (2005), pp. 1–6.

[6] S. Cotterell and F. Vahid, *A Logic Block Enabling Logic Configuration by Non-experts in Sensor Networks*, in Extended Abstracts of the 23th ACM Conference on Human Factors in Computing Systems - CHI 2005 (Portland, Oregon, USA), New York, NY, USA, 2005, ACM Press, pp. 1925–1928.

[7] S. Cotterell, F. Vahid, W. Najjar, and H. Hsieh, *First Results with eBlocks: Embedded Systems Building Blocks*, in Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis - CODES+ISSS 2003 (Oct. 1-3, Newport Beach, CA), IEEE Computer Society, 2003, pp. 168–175.

[8] A. Dey, R. Hamid, C. Beckmann, and D. Hsu, *A CAPpella: Programming by Demonstration of Context-Aware Applications*, in Proceedings of the Conference on Human Factors in Computing Systems - CHI 2004 (Apr. 24-29, Vienna, Austria), ACM Press, 2004, pp. 33–40.

[9] H. Gellersen, G. Kortuem, A. Schmidt, and M. Beigl, *Physical Prototyping with Smart-Its*, IEEE Pervasive Computing, 3 (2004), pp. 74–82.

[10] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer, *A System for Simulation, Emulation, and Deployment of Heterogeneous Sensor Networks*, in In Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems - SenSys 2004 (Nov. 3-5, Baltimore, MD), ACM Press, 2004, pp. 201–213.

[11] S. Greenberg and C. Fitchett, *Phidgets: Easy Development of Physical Interfaces Through Physical Widgets*, in Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology - UIST 2001 (Orlando, Florida, USA), New York, NY, USA, 2001, ACM Press, pp. 209–218.

[12] I. Greif and S. Sarin, *Data sharing in group work.*, ACM Transactions on Offce Information Systems 5, 2, (1987), pp. 187–211.

[13] T. Gross, T. Egla, and N. Marquardt, *Sens-ation: A Service-Oriented Platform for Developing Sensor-Based Infrastructures*, International Journal of Internet Protocol Technology (IJIPT), 1 (2006), pp. 159–167.
[14] D. Hilbert and J. Trevor, *Personalizing shared ubiquitous devices*, ACM interactions 11, 3, (2004), pp. 34–43.
[15] J. Humble, A. Crabtree, T. Hemmings, K.-P. Åkesson, B. Koleva, T. Rodden, and P. Hansson, *"Playing with the Bits" - User-Configuration of Ubiquitous Domestic Environments*, in Proceedings of the Fifth International Conference on Ubiquitous Computing - UbiComp 2003 (Seattle, Washington, USA), Seattle, WA, USA, 2003, pp. 256–263.
[16] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, *TinyDB: An Acquisitional Query Processing System for Sensor Networks*, ACM Transactions on Database Systems 30, 1, (2005), pp. 122–173.
[17] N. Marquardt and S. Greenberg, *Distributed Physical Interfaces with Shared Phidgets*, in Proceedings of the 1st International Conference on Tangible and Embedded Interaction - TEI 2007 (Baton Rouge, LA, USA), New York, NY, USA, 2007, ACM Press, pp. 13–20.
[18] I. Mavrommati, A. Kameas, and P. Markopoulos, *An Editing Tool that Manages Device Associations In an in-home Environment*, Personal and Ubiquitous Computing, 8 (2004), pp. 255–263.
[19] J. Olson, J. Grudin, and E. Horvitz, *Towards Understanding Preferences for Sharing and Privacy*, tech. report, Report Number: MSR-TR-2004-138, Microsoft Research, 2004.
[20] ———, *A Study of Preference for Sharing and Privacy*, in Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2005 (Apr. 2-7, Portland, OR), ACM, 2005, pp. 1985–1988.
[21] S. Park, A. Savvides, and M. Srivastava, *SensorSim: A Simulation Framework for Sensor Networks*, in Proceedings of the 3rd ACM International Workshop on Modelling, Analysis, and Simulation of Wireless and Mobile Systems - MSWiM 2000 (Aug. 11, Boston, MA), ACM, 2000, pp. 104–111.
[22] T. Rodden, A. Crabtree, T. Hemmings, B. Koleva, J. Humble, K.-P. Akesson, and P. Hansson, *Between the Dazzle of a New Building and its Eventual Corpse: Assembling the Ubiquitous Home*, in Proceedings of the 5th ACM Conference on Designing Interactive Systems - DIS 2004 (Cambridge, Massachusetts, USA), New York, NY, USA, 2004, ACM Press, pp. 71–80.
[23] D. Salber, A. K. Dey, and G. D. Abowd, *The Context Toolkit: Aiding the Development of Context-Enabled Applications*, in Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 1999 (Pittsburgh, Pennsylvania, USA), New York, NY, USA, 1999, ACM Press, pp. 434–441.
[24] A. Sobeih, W.-P. Chen, J. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, and K. Zhang, *J-Sim: A Simulation Environment for Wireless Sensor Networks*, in Proceedings of the 38th Annual Symposium on Simulation (Apr. 2-8, San Diego, CA), IEEE Computer Society, 2005, pp. 175–187.
[25] T. Sohn and A. Dey, *iCAP: An Informal Tool for Interactive Prototyping of Context-Aware Applications*, in Extended Abstracts of the 21st ACM Conference on Human Factors in Computing Systems - CHI 2003 (Fort Lauderdale, Florida, USA), New York, NY, USA, 2003, ACM Press, pp. 974–975.
[26] S. Talja, *Information Sharing in Academic Communities*, New Review of Information Behaviour Research 3, (2002), pp. 143–159.
[27] J. Walnes, *XStream - Architecture Overview. http://xstream.codehaus.org/architecture.html*, 2010. (Accessed 5/5/2010).
[28] D. Winer, *XML-RPC Specification. http://www.xmlrpc.com/spec*, 1999. (Accessed 5/5/2010).
[29] World Wide Web Consortium (W3C), *XML Path Language (XPath). http://www.w3.org/tr/xpath*. W3C Recommendation, November 1999. (Accessed 5/5/2010).
[30] ———, *Extensible Markup Language (XML) 1.0. http://www.w3.org/xml/*. W3C Recommendation, August 2006. (Accessed 5/5/2010).

Edited by: Pasqua D'Ambra, Daniela di Serafino, Mario Rosario Guarracino, Francesca Perla
Received: June 2007
Accepted: November 2008