CollaborationBus: An Editor for the Easy Configuration of Ubiquitous Computing Environments

Tom Gross, Nicolai Marquardt
Faculty of Media
Bauhaus-University Weimar
Bauhausstr. 11
99423 Weimar, Germany
<firstname.lastname>(at)medien.uni-weimar.de

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 introduce the CollaborationBus application: a graphical editor that provides abstractions from base technology and thereby allows experts as well as non-experts to configure 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. 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 the information flows through their pipelines. In this paper we present the concept and implementation of the CollaborationBus application.

1 Introduction

Creating, maintaining, and adapting sensor-based infrastructures in general requires a thorough knowledge of base technology for sensing data, for processing the captured data, and for adapting the system's behaviour accordingly [3, 10, 23, 24, 26]. In this paper we argue that besides experts also non-experts—that is, users who have some basic knowledge of application programming, but do not necessarily have expert knowledge in sensor hardware, wired and wireless networking, and data processing algorithms—should be able to create, maintain, and adapt configurations of Ubiquitous Computing environments.

There are some research projects providing easy-to-use configuration interfaces for non-expert users to create sensor-based Ubiquitous Computing environments, yet they generally target at configuration for the private home [9, 15, 18, 25]. Furthermore, most systems lack integrated facilities for the collaborative exchange of users' configurations. Only some systems—typically complex configuration tools [3, 17]—provide enhanced visualisations of the data flow and sensor-network data [4] to support users while creating or configuring environments.

In this paper we introduce *CollaborationBus*: a graphical editor that provides adequate abstractions from base technology and thereby allows non-expert users to easily configure Ubiquitous Computing environments. CollaborationBus is based on three main concepts—pipeline compositions, integrated sharing mechanisms, and real-time visualisations—which per se and in combination go beyond existing approaches:

- The *pipeline compositions* enable users to easily specify the information flows through Ubiquitous Computing environments 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.
- The *integrated sharing mechanisms* allow users to share their own pipeline compositions with other users. In an analogous manner they can add others' compositions to their own repository, and build new compositions based on these compositions.
- The *real-time visualisations* display relations between incoming and outgoing events, and let the user interactively keep track of and adjust the information flow through their pipelines. They help the users understand the information flow, which can become quite complex sets of sensors, filters, and actuators.

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

2 Requirements

In this section we develop scenarios of configurations for Ubiquitous Computing environments that users might want to develop and maintain, and we derive requirements for the CollaborationBus editor.

2.1 Application Scenarios

Basically, users should be able to configure Ubiquitous Computing environments in their private homes as well as in their workplaces, as exemplified below.

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 sensor attached to the telephone detects whether the phone receiver is picked up and serves as 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 wants to specify the desired information flow: if the pipeline detects that the phone is used, the volume of the computer (e.g., an AppleScript application is started to mute the volume of the Mac) and the sound system (e.g., a sensor board sends out infrared commands) should be muted, and the user's calendar application should be started so that the user can input new appointments during the phone call (e.g., another AppleScript application starts the iCal application). When the phone call ends, the original state should be re-established 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 computer science 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 that we developed in our research group (three of which were described above), and from a detailed study of related work (some examples of related work are presented 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 beginners—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 essential 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 and outcomes of more experienced users.

Subsequently we present the concept and implementation of CollaborationBus addressing these functional requirements.

3 Concept

In this section we describe the key concepts of the CollaborationBus editor.

3.1 Generic Approach

CollaborationBus works across multiple applications domains, temporal patterns, and complexity patterns.

Application Domains. Sensor- and actuator-based environments in the private home differ from those in the work domain in the use of software and hardware sensors. CollaborationBus supports the creation and maintenance of environments in the workplace and at home.

Temporal Patterns. In any application domain various patterns with regard to capturing data and starting actuators can be identified: recurrent, permanent (e.g., ongoing data collection); recurrent, occasionally (e.g., depending on day-time, day of the week); and one-time (e.g., call-back if the required person is reachable). CollaborationBus supports any of these temporal patterns.

Complexity Patterns. Each setting can have a specific complexity pattern such as: one sensor, one actuator (e.g., one binary sensor controls one actuator); sensor, filters, actuator (e.g., only react to specific values of a temperature sensor); multiple parallel sensors, filters, and actuators (e.g., create summaries of various sensor sources, control a set of actuators); and complex network of components (e.g., determine the current work context or mood of a person). CollaborationBus supports all complexity patterns, except for complex networks that are too complex to be represented in a graphical editor and require programming by experts.

3.2 Pipeline Metaphor

CollaborationBus handles relations between sensor and actuators with a pipeline metaphor.

Pipelines are compositions that connect and forward event data and commands between at least one sensor and one actuator component, and optionally filter components for processing sensor values between them. They can be nested: parallel sub-pipelines as logical `OR` condition; sequences of sensor sources as logical `AND` condition; and negations as logical `NOT` condition.

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

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

Filters represent single conditions or transformations and generate data of particular formats (e.g., `Integer`, `Boolean`, `String`). Filter types are: universal (e.g., count number of event occurrence, create event summary reports); numerical (e.g., determine numeric threshold, interpolation, average); string (e.g., search for specified keywords); binary (e.g., create negation, conjunction); and transformations (e.g., convert numeric value to string message, binary value to numeric). The filter components include a variety of transformation methods (e.g., for generating a SMS a string message can be entered, and the values of the respective sensors can be attached).

With CollaborationBus users can rapidly connect local sensors and actuators or sensors and actuators from remote locations and build new configurations.

3.3 Diverse Users

CollaborationBus works for users with diverse levels of technical background. *Novice* users with some basic knowledge of application programming can start using CollaborationBus by loading and adapting pre-configured environment configurations that are part of the CollaborationBus distribution. Feasibility checks automatically deactivate inadequate operations (e.g., average calculations on non-numeric data). *More experienced users* can create their own environment 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 using 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.

3.4 Collaborative Sharing

Three types of sharing of compositions are possible: *Sensor and event sharing* allows users to either share events of their own sensors or processed event data of their filters. *Actuator sharing* allows users to 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. And, *pipeline sharing* allows users to share complete pipeline compositions with others.

With all sharing methods security and privacy are maintained through abstraction and access control. Users can choose to only share *abstract templates*—so a composition contains the configuration of all filter components, but the original sensors and actuators are not shared (other users cannot see the creator's event data captured by her software and hardware sensors). Other users can then insert their own sensors in the placeholders at the beginning of the pipeline composition, and their own actuators at the end. This lets them use the knowledge of the processing filter components of the composition, while at the same time the users who share pipeline compositions preserve privacy of their personal data.

4 Implementation

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

4.1 CollaborationBus Software Architecture

Figure 1 provides a schematic overview of 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 interpret, aggregate, and transform 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 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.

Personal and shared repository are serialised in XML format, for platform independency and easy exchange of pipeline composition descriptions. The CBServer can serialise and de-serialise XML descriptions, and validate and process descriptions. An example of a Personal Repository encoded in XML looks like the following:

```
<personalRepository>
  <pipelineCompositions>
   <pipelineComposition>
    <name>Smart Telephone
    </name>
    <description>...
    </description>
    <active>false</active>
    <edited>false</edited>
    <sharePrivileges>...
    </sharePrivileges>
    <tagTemplate>false</tagTemplate>
    <tagReuse>true</tagReuse>
    <tagShareable>true</tagShareable>
    <idCounter>4</idCounter>
    <components>
      [...]
    </components>
    <pipes>
     <pipeEntry>
      <sourceID>source_component_1
      </sourceID>
      <sinkID>filter_component_2
      </sinkID>
     </pipeEntry>
     [...]
    </pipes>
   </pipelineComposition>
     [...]
  </pipelineCompositions>
</personalRepository>
```

Each pipeline composition is represented by a separate XML subentry; with all specifications included to instantiate all necessary software objects to rebuild the complete pipeline composition. Inside of each of these pipeline compositions we have the entries of the necessary pipeline components. Each of these component entries starts with the name and description as well as all component specific definitions (e.g., threshold values, string keywords). An example of a single component description (keyword filter object) is the following:

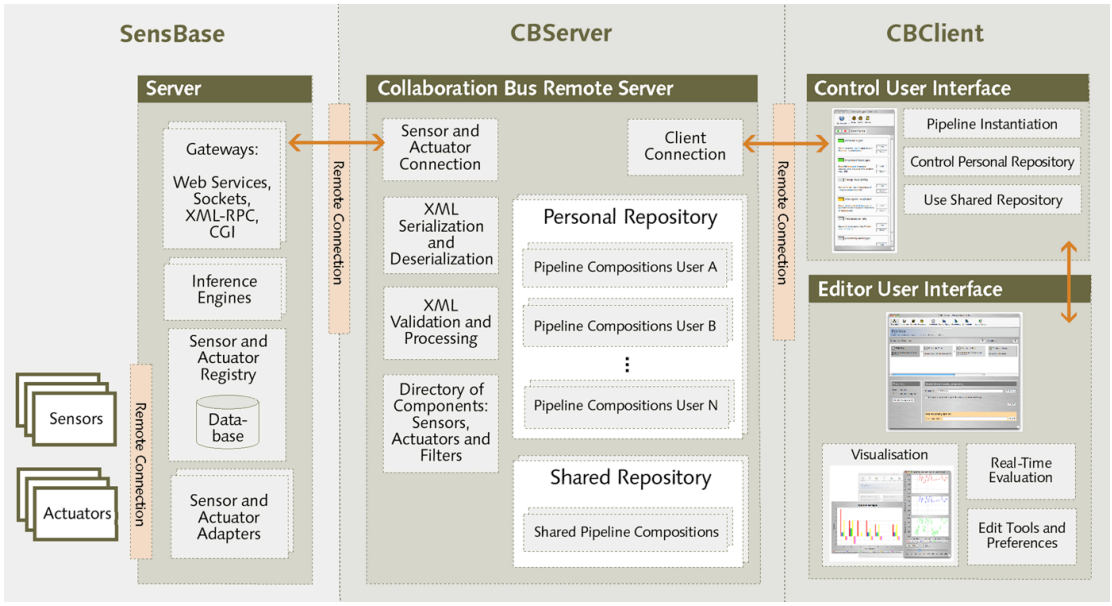

Figure 1. CollaborationBus schematic overview.

```
<filterKeyword>
  <name>Keyword Filter</name>
  <description>...</description>
  <keyWords>
    <string>Gross</string>
    <string>Marquardt</string>
      [...]
  </keyWords>
  <occurrence>1</occurrence>
  <forwardType>...</forwardType>
  <id>filter_component_8</id>
  <componentType>filter</componentType>
    [...]
  <isActive>false</isActive>
  <interrupted>false</interrupted>
  <isInitialized>true</isInitialized>
  <sinks>
    <filter>...</filter>
  </sinks>
</filterKeyword>
```

The published pipeline compositions for collaborative sharing are stored in the *Shared Repository*. They are saved in the XML format as well. Each of these shared entries starts with the identification of the publisher, the type of the shared entry (e.g., abstract template, complete pipeline), the category, and the description. The rest of the XML entry includes the pipeline composition itself, either as complete pipeline composition, or as an abstract template. An example of a Shared Repository is the following:

```
<sharedRepository>
  <sharedRepositoryEntry>
    <publisher>Nicolai Marquardt</publisher>
    <type>Complete pipeline</type>
    <category>Cooperative Media Lab
    </category>
    <description>Information channel
      of the CML lab.
    </description>
    <pipelineComposition>
      [...]
    </pipelineComposition>
  </sharedRepositoryEntry>
    [...]
</sharedRepository>
```

Furthermore, the CBServer manages a *directory* of all pipeline components (sensors, filter and actuators), and submits their specification to all clients. The dynamic directory can be extended with new components at any time, and this ensures the easy extendibility of CollaborationBus. New pipeline components can be developed by easily deriving them from one of the abstract base class (`Sensor`, `Filter`, `Actuator`). For them all common and important pipeline element methods are already implemented.

The *CBClient* implements the GUIs described below. 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 (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 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

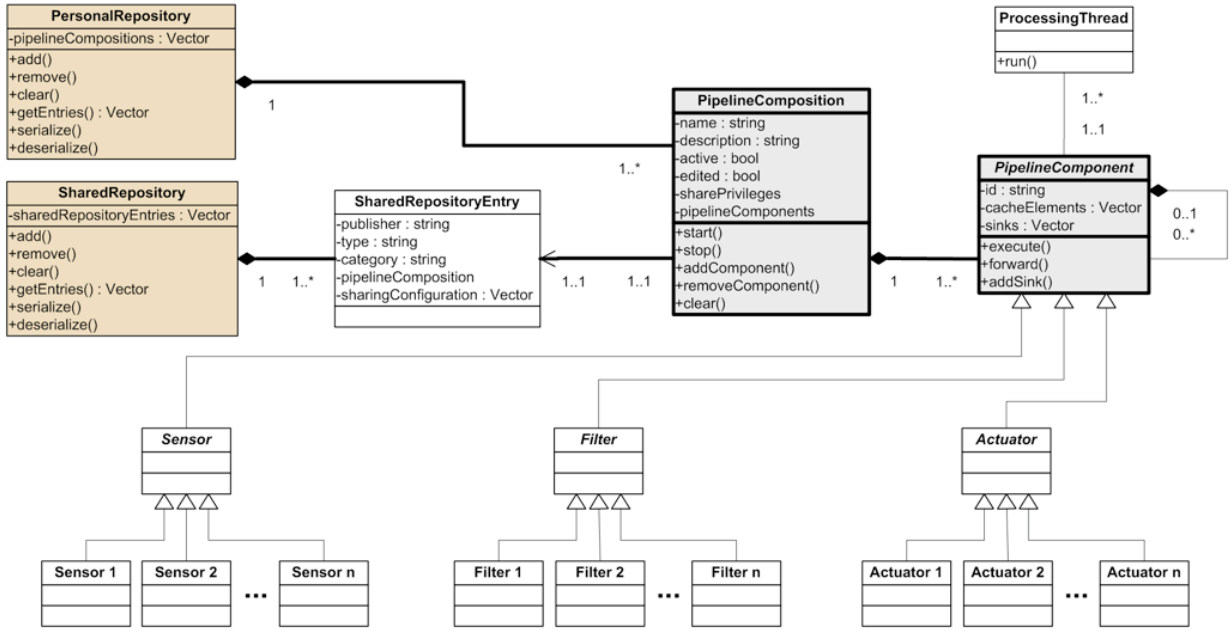

Figure 2. CollaborationBus repository and pipeline UML class diagram.

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. 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 ESB 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 by means of the Eclipse IDE version 3.1M6 on Mac OS X 10.4 in Java version 1.4.2 with Swing libraries for the GUI. Several libraries are used for XML [29] processing (e.g., for the serialisation and de-serialisation of pipeline compositions [31], for parsing sensor descriptions, for creating XPath expressions [28]); for remote connections (e.g., SOAP [27] and XML-RPC [30] connections); and for GUI enhancements.

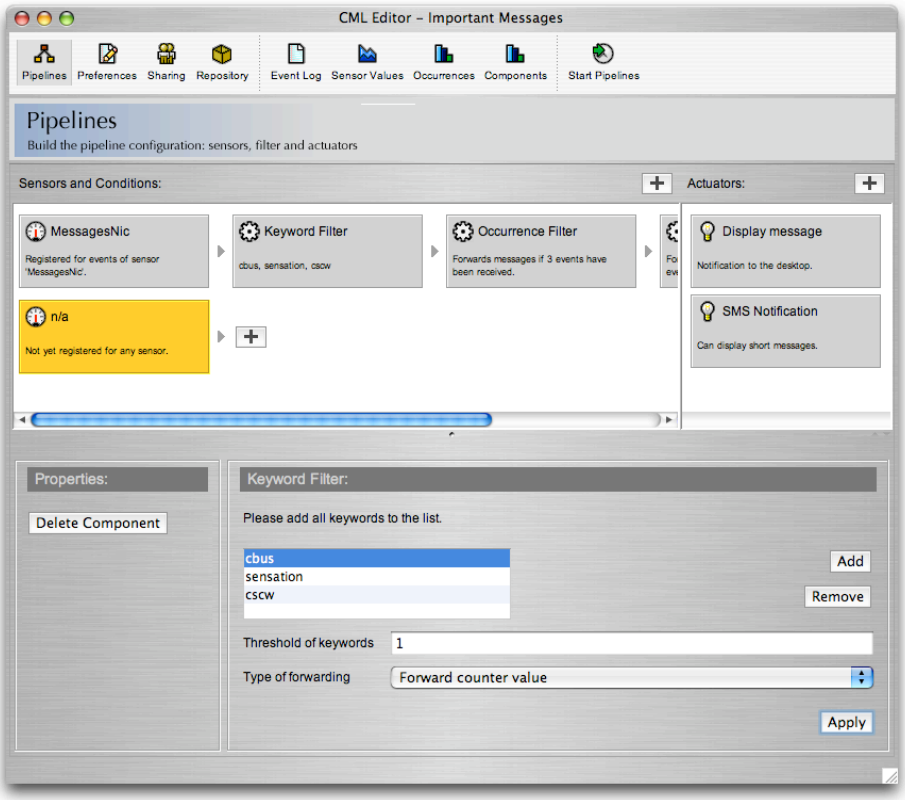

Figure 3. CollaborationBus Editor GUI.

5 User Interface

CollaborationBus provides four GUI components, which we briefly describe below.

Login and Control GUI. In order to get to their Control GUI, users have to login through the Login GUI. 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 (i.e., running, in edit mode, off). All functions for modifying the repository and its compositions are available from within this interface such as adding or removing pipelines; starting and stopping the threaded execution of compositions; sharing selected compositions.

Editor GUI. In the Editor GUI users can access the filter compositions underlying each pipeline. Figure 3 shows the Editor GUI. In the top area the user can choose several buttons for loading the *Pipelines*, change the *Preferences*, and so forth. In the middle area the respective pipeline with its sensors, filters, and actuators is shown (each item is represented as a rectangular box). In the bottom area the properties of the currently selected pipeline component (rectangular box) are shown and can be altered. In order to create a new pipeline composition, users can select available sensors in a *sensor browser*. For each sensor type with corresponding sensor value type, specific filters and operators can be selected. Finally, the actuators selected them in the *actuator browser*.

Shared Repository GUI. The collaborative sharing mechanism is integrated in the Control GUI and in the Editor GUI. Here users can either share and upload their compositions, or download compositions.

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 provides a variety of graph visualisations that can display the forwarded values of each component of the pipeline.

An input interface allows users to manually insert sensor values to test and verify the pipeline composition without having to wait for real sensor values from the sensors.

6 Related Work

There are some prototypes and systems with similarities to CollaborationBus regarding the pipelines and filters metaphor, the support of non-expert users, and the collaborative sharing. For the sake of completeness we also glance at environments for experts and at sensor network composition software.

Pipelines and filters. This concept is used in some systems. For instance, the Automator of Mac OS X [1] is a graphical editor for scripting the operating system and applications, and MAX/MSP [8] is a graphical environment for music, audio, and multimedia.

Environments for non-experts. iCAP [25] allows users to rapidly prototype Ubiquitous Computing environments. e-Gadgets [18] is an editing tool for creating device associations in a home environment. The jigsaw [15, 22] is an editor for getting control over the technological home environment through assembling pieces of a jigsaw puzzle. Some systems that are based on mobile devices to control configurations are systems for PDAs [18] and TabletPCs [15]. A CAPpella [9] supports programming by demonstration of context-aware environments. In eBlocks [6, 7] a user interface for building sensor-based environments and configuring condition tables are provided. The Phidgets toolkit [11] facilitates the development of physical user interfaces by providing a range of sensor and actuator elements.

Collaborative sharing. While in Computer-Supported Cooperative Work collaborative sharing of location information, files, workspaces, software and patterns is wide-spread [12], an approach to sharing sensor- and actuator-based environments among users is still missing. In [12] design issues of Computer-Supported Cooperative Work 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]. Hilbert and Trevor describe the importance of personalisation as well as shared devices for Ubiquitous Computing environments [14].

Environments for experts. The iQL programming model [5] is a non-procedural language for specifying the behaviour of components in pervasive environments. Papier-Mache [16] provides programming tools for programming tangible user interfaces.

Sensor network composition software. The VisualSense framework of PTOLEMY II [2, 3] is a toolkit for the modelling and simulation of fine granular sensor network communication and processing. Several special complex development environments for the evaluation of the communication in sensor networks have been presented (e.g., SensorSim [21], EmTOS [10], TinyDB [17], J-Sim [24], and event flow visualisation [4]). However, non-experts probably have difficulties in using these environments.

7 Conclusions

In this paper we have introduced the *CollaborationBus* editor that encapsulates and hides the details of the underlying technology. It allows experts as well as non-experts to easily specify Ubiquitous Computing environments. More experienced users can control the pipeline composition configuration in any technical detail they need and get details on demand. Users can share their pipeline compositions via a shared repository. 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.

In an *informal evaluation* we have collected several user opinions at the public demonstration of CollaborationBus to many visitors at the Cooperative Media Lab Open House over three days, where the visitors had the chance to try out the CollaborationBus editor in detail (with a huge set of connected sensors and actuators). After a short introduction of the system, several visitors started to create their own compositions, and to select desired sensors, filters, and actuators. 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; and 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. However, some users were worried about privacy issues when sharing their compositions.

In the *future* we would like to evaluate the created pipeline compositions of the 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 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.

Acknowledgments

We thank the members of the Cooperative Media Lab—especially Tareg Egla, and Christoph Oemig—for inspiring discussions on the concepts and implementation of CollaborationBus, and for providing the PRIMI and Sens-ation platform. Thanks for the anonymous reviewers for comments on earlier versions of this paper.

References

1. Apple Computer, I. Apple - Mac OS X - Automator. http://www.apple.com/macosx/features/automator, 2006. (Accessed 2/10/2006).
2. Baldwin, P., Kohli, S., Lee, E.A., Liu, X. and Zhao, Y. Modelling of Sensor Nets in Ptolemy II. In Proc. Third International Symposium on Information Processing in Sensor Networks - IPSN 2004. pp. 359-368.
3. Baldwin, P., Kohli, S., Lee, E.A., Liu, X. and Zhao, Y. Visualsense - Visual Modeling for Wireless and Sensor Network Systems. Report Number: UCB ERL Memorandum UCB/ERL M04/8, Ptolemy Project, Apr. 2004.
4. Buschmann, C., Pfisterer, D., Fischer, S., Fekete, S.P. and Kroeller, A. SpyGlass: A Wireless Sensor Network Visualiser. SIGBED Review 2, 1 (Jan. 2005). pp. 1-6.
5. Cohen, N.H., Lei, H., Castro, P., Davis, J.S. and Purakayastha, A. Composing Pervasive Data Using iQL. In Proc. Fourth Workshop on Mobile Computing Systems and Applications - WMCSA 2002. pp. 94-104.
6. Cotterell, S. and Vahid, F. A Logic Block Enabling Logic Configuration by Non-Experts in Sensor Networks. In Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2005. pp. 1925-1928.
7. Cotterell, S., Vahid, F., Najjar, W. and Hsieh, H. First Results with eBlocks: Embedded Systems Building Blocks. In Proc. 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis - CODES+ISSS 2003. pp. 168-175.
8. Cycling'74. Cycling'74 || maxmsp. http://www.cycling74.com/products/maxmsp, 2006. (Accessed 2/10/2006).
9. Dey, A.K., Hamid, R., Beckmann, C. and Hsu, D. A CAPpella: Programming by Demonstration of Context-Aware Applications. In Proc. Conference on Human Factors in Computing Systems - CHI 2004. pp. 33-40.
10. Girod, L., Stathopoulos, T., Ramanathan, N., Elson, J., Estrin, D., Osterweil, E. and Schoellhammer, T. A System for Simulation, Emulation, and Deployment of Heterogeneous Sensor Networks. In Proc. Second International Conference on Embedded Networked Sensor Systems - SenSys 2004. pp. 201-213.
11. Greenberg, S. and Fitchett, C. Phidgets: Easy Development of Physical Interfaces Through Physical Widgets. In Proc. ACM Symposium on User Interface Software and Technology - UIST 2001. pp. 209-218.
12. Greif, I. and Sarin, S. Data Sharing in Group Work. ACM Transactions on Office Information Systems 5, 2 (Apr. 1987). pp. 187-211.
13. 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.
14. Hilbert, D.M. and Trevor, J. Personalizing Shared Ubiquitous Devices. ACM interactions 11, 3 (May/June 2004). pp. 34-43.
15. 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 Fifth International Conference on Ubiquitous Computing - UbiComp 2003. pp. 256-264.
16. Klemmer, S.R., Li, J., Lin, J. and Landay, J. Papier-Mache: Toolkit Support for Tangible Input. In Proc. Conference on Human Factors in Computing Systems - CHI 2004. pp. 399-406.
17. Madden, S.R., Franklin, M.J., Hellerstein, J.M. and Hong, W. TinyDB: An Acquisitional Query Processing System for Sensor Networks. ACM Transactions on Database Systems 30, 1 (Mar. 2005). pp. 122-173.
18. Mavrommati, I., Kameas, A. and Markopoulos, P. An Editing Tool that Manages Device Associations in an In-Home Environment. Personal Ubiquitous Computing 8, 3-4 (July 2004). pp. 255-263.
19. Olson, J.S., Grudin, J. and Horvitz, E. Towards Understanding Preferences for Sharing and Privacy. Report Number: MSR-TR-2004-138, Microsoft Research, 2004.
20. Olson, J.S., Grudin, J. and Horvitz, E. Late Breaking Result: A Study of Preferences for Sharing and Privacy. In Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2005. pp. 1958-1988.
21. Park, S., Savvides, A. and Srivastava, M.B. SensorSim: A Simulation Framework for Sensor Networks. In Proc. 3rd ACM International Workshop on Modelling, Analysis, and Simulation of Wireless and Mobile Systems - MSWiM 2000. pp. 104-111.
22. Rodden, T., Crabtree, A., Hemmings, T., Koleva, B., Humble, J., kesson, K.-P. and Hansson, P. Between the Dazzle of a New Building and its Eventual Corpse: Assembling the Ubiquitous Home. In Proc. Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques - DIS 2004. pp. 71-80.
23. Salber, D., Dey, A.K. and Abowd, G.D. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In Proc. Conference on Human Factors in Computing Systems - CHI'99. pp. 434-441.
24. Sobeih, A., Chen, W.-P., Hou, J.C., Kung, L.-C., Li, N., Lim, H., Tyan, H.-Y. and Zhang, K. J-Sim: A Simulation Environment for Wireless Sensor Networks. In Proc. 38th Annual Symposium on Simulation. pp. 175-187.
25. Sohn, T. and Dey, A.K. Interactive Poster: iCAP: An Informal Tool for Interactive Prototyping Context-Aware Applications. In Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2003. pp. 974-975.
26. Talja, S. Information Sharing in Academic Communities. New Review of Information Behaviour Research 3 (2002). pp. 143-159.
27. The Apache Software Foundation. WebServices - Axis Architecture Guide. World Wide Web Consortium, http://ws.apache.org/axis/java/architecture-guide.html, 2005. (Accessed 2/10/2006).
28. W3C. XML Path Language (XPath). W3C Recommendation. World Wide Web Consortium, http://www.w3.org/TR/xpath, 1999. (Accessed 2/10/2006).
29. W3C. Extensible Markup Language (XML). World Wide Web Consortium, http://www.w3.org/XML/, 2006. (Accessed 2/10/2006).
30. Winer, D. XML-RPC Specification. http://www.xmlrpc.com/spec, 1999. (Accessed 2/10/2006).
31. XStream. XStream - Architecture Overview. World Wide Web Consortium, http://xstream.codehaus.org/architecture.html, 2006. (Accessed 2/10/2006).