The PPPSpace: Innovative Concepts for Permanent Capturing, Persistent Storing, and Parallel Processing and Distributing Events

Tom Gross, Christoph Beckmann, Maximilian Schirmer
Faculty of Media
Bauhaus-University Weimar
99423 Weimar, Germany
<firstname.lastname>(at)medien.uni-weimar.de

***Abstract*—Media spaces provide users with flexible support for easy interaction with technology and with each other, both at the same place and over distance. From a technological perspective the development of these environments is often inefficient, since most environments are developed specifically, without any synergies or reuse of previous concepts and implementations. In this paper we present the cooperative media space PPPSpace that is based on powerful technical parallel and distributed software engineering concepts and at the same time easy to use for end-users.**

***Keywords*—Distributed System; Event Notification Infrastructure; Media Space; End-User Configuration.**

I. INTRODUCTION

Media spaces are cooperative ubiquitous environments that provide convenient connections of rooms and their users. They typically support permanent audio and video connections between two physical rooms, where users can have easy background information on who is around in either room and can flexibly start conversations with present and remote colleagues. In general, media spaces aim to bridge distance by providing connections that are permanently available and do not require users to make any configurations of connections or call their remote colleagues. They can simply see and hear who is around at the other site. Accordingly, 'in a media space, people can create real-time visual and acoustic environments that span physically separate areas' [3, p. 30].

When users are working in such a distributed environment, they do not only require audio and video connections with remote rooms and other users, but also background information on the cooperative activities of all users. This background information is particularly important if the distributed users want to go beyond pure presence and communication and really cooperate with each other on a shared task or project. Such awareness information provides a basis for shared orientation in teams and for effective and efficient team work [12].

Sensor-based event notification infrastructures offer technologically advanced concepts for the realisation of awareness information distribution. They provide support for capturing data, processing data, and presenting data to local and remote users who are interested in the respective data and have adequate access rights to the data [9, 11].

Since the capturing and presenting of data entails challenges for users' privacy and disruption [14], end-users need to be able to configure their environment, including individual preferences for disclosing personal information, as well as preferences for the adequate presentation of information within the media space environment. Only if the users are provided with both a complex and sophisticated event notification infrastructure underneath, and an easy editor for configuring their environment with respect to the information they want to be captured about them, and with respect to the information they want to be presented in their environment.

Existing event notification infrastructures mostly do not enable end-users to configure their media space environment according to their needs and therefore leave this task to experts like programmers or administrators.

In this paper, we present the *PPPSpace* (*Permanent, Persistent*, and *Parallel Space*) as a technologically powerful sharing environment for connecting rooms and users. The environment is based on permanent audio and video channels as well as event notifications for awareness information that result from various sensors. Notifications are presented on special *PPPSpace* nodes and mobile clients as well as on workstation computers. With a lightweight editor, the *PPPSpace* enables end-users to configure their environment. So, end-users can discover, manage, manipulate, and configure components of the media space environment, and construct and explore their personal scenarios.

In the next section we introduce the concepts of *PPPSpace* for permanent capturing, persistent storing, and parallel processing and distributing of data in a media space environment. We then report on the implementation of *PPPSpace*, and we outline a performance evaluation of the environment. We sketch related work. Finally, we draw conclusions, and glance at future work.

II. THE PPPSPACE CONCEPT

The *PPPSpace* concept provides mechanisms for the **P**ermanent capturing by sensors in the environment, for **P**ersistent storing of awareness data, and for **P**arallel processing and distributing of data to clients that registered for notifications on these sensors.

In *PPPSpace* data is captured with the help of clients that provide sensors for all available data sources. The data is processed and distributed with a sensor-based event notification infrastructure. The processing within the server is based on inference engines that allow to filter data or apply heuristics on data and that can be integrated in any stage of the processing chain between the gathering

and the presenting of data. Data is presented with actuators in the clients; actuators display data or provide changes to the environments that serve as subtle notifications.

End-users can make configurations based on chains of sensors, inference engines, and actuators. With an easy-to-use editor users can produce dynamic configurations of environments with sensors (for end-users called data sources), with inference engines (for end-users called filters), and with actuators (for end-users called actuators).

In this section, we subsequently describe the mentioned concepts and mechanisms in detail.

A. Permanent Capturing of Data

The permanent capturing concept provides mechanisms for sensors that constantly gather data from the environment and forward changes in the environment to the server. The captured data contributes to a rich collection of sensed data, which helps to infer higher-order data on activities in the environment.

A variety of sensors capture data from electronic sources (cf. Table 1). We have also developed adaptors for integrating external sensors that capture data from physical sources, but these are not used in the current scenario. Each sensor has four distinct collecting variants specifying levels of granularity of sensor data to be captured. They provide users a means to decide how many details on personal information should be captured and stored. The collecting variants are:

- Precise: complete and exact data
- Approximate: selected and exact data
- Vague: selected and rounded data
- Undisclosed: no data

The latter implies that from a specific sensor no data at all are captured, and the user has complete privacy. Please note that the granularity of the captured data is always dependent on the capabilities of the respective sensor. For the sensors used in our environment this implies that data on the users (first six entries in Table 1) are captured on the users' computers as excerpt of users' communication and work data as well as on the users' mobile phone as GPS data (seventh entry in Table 1); and data on the environment (latter two entries in Table 1) are captured as raw data streams.

Sensors [1,*]	
Name	**Description**
Mail	Data on unread email messages in email application
Calendar	Data on current day's events in calendar application
Shared work-space	Data on recently changed documents in the shared workspace
Music	Data on songs that are currently played on respective user's workstation computer
Notes	Data on notes created on a workstation computer
Social network	Data on recently published personal states in an social networking Web application
GPS	Data on respective user's current position
Camera	Camera stream for each location
Audio	Audio stream for each location

Table 1. Overview of the PPPSpace sensors.

B. Persistent Storing of Event Data and Environment Component Data

The persistent storing concept provides mechanisms for saving and retrieving event data, and for descriptions of components of the environment. These mechanisms are base of the persistent, server-based runtime environment.

The environment is capable of storing all data gathered by sensors persistently as event data. This enables components of the environment to retrieve data from the central server-based runtime environment independently of their actual location. Sensor data is typically hierarchical and relational—that is, sensor data can have a semantic relationship with each other and can also be nested (e.g., multiple sensor data can be the ingredients of a composed data set). Therefore, all data is stored hierarchically based on the sensor the information originated from, and the time of occurrence.

The environment also stores the descriptions of its components persistently. For instance, properties for each sensor (e.g., collecting variants, descriptions, data types) can be saved, retrieved, but also changed over time.

C. Parallel Processing and Distributing of Event Data

The parallel processing and distributing concept provides mechanisms for the simultaneous handling of incoming data in different components of the environment. Therefore, the server provides parallel subsystems for event data distribution to internal inference engines for the abstraction of high-level information, and attached clients that act as actuators.

Inferencing information requires a data distribution process where the server includes all inference engines into the notification distribution process, providing them with the necessary event data. Several inference engines can be used in series, while each inference engine relies on the processed data of another inference engine in the series as input. The event data processed in inference engines can originate from the respective user's or other users' sensor data, and considers the individual disclosure preferences.

Table 2 provides an overview of the inference engines that can be used in the *PPPSpace* concept.

Distributing event data in parallel to subsystems and to actuators allows the server to handle a huge amount of incoming data. Actuators are registered, managed, and notified according to a publish/subscribe concept with scalable adaptive publish to interested subscribers.

Inference Engines [0,*]	
Name	**Description**
String Filter	Returns event data if string matches
Threshold Filter	Returns event data if threshold is reached
XML Tag Filter	Filters XML documents for the occurrence of a tag; can be configured to either return complete event data, or matching sub-tree, or the sub-tree's contents
Logical Connec-tions	Logical connections: AND, OR, NOT

Table 2. Overview of the PPPSpace inference engines.

In this concept, actuators are subscribers to data that is published by the server. Subscribers are notified whenever events they are interested in occur. For each notification, the server dispatches a notification process that is handled in the background, parallel to the notification processes of all actuators and other attached clients.

Table 3 provides an overview of the actuators in the *PPPSpace* concept.

Actuators [1,*]	
Name	**Description**
Global PPPSpace Presenter	Presentation of notification on all nodes of a PPPSpace environment
Local PPPSpace Presenter	Presentation of notification on a certain nodes of a PPPSpace environment
Desktop Notification	Presentation of notification on a workstation computer (either as information presentation; or as start of application)
Mobile Notification	Presentation of notification on a mobile phone

Table 3. Overview of the PPPSpace actuators.

D. Dynamic Configuring and Sharing of Configurations

The dynamic configuring and sharing concept provides mechanisms for the end-user configuration of the environment components and a sharing mechanism for configurations of the users.

With the help of an editor, end-users configure their personal environment. The editor offers graphical user interface elements that represent abstractions of the software and hardware used in the environment. End-users instantiate user interface elements representing environment components in the editor and configure the data flow between these components. During the configuration process, the editor's user interface elements reflect the current state of all environment components and allow the users to inspect recent event data as well as shared configurations of other users. This way the users do not need any technical skills of the underlying event and notification infrastructure—they simply manipulate objects on the graphical user interface.

When multiple users make use of the same components in a similar way, the editor highlights this synergetic use and allows users to browse and inspect the corresponding configurations of other users. Multiple synergetic use of the same components leads to more complex and highly adapted configurations that represent comprehensive user scenarios.

The graphical user interface and all user interface elements of the editor are dynamically generated according to the settings for each environment component. This includes the required user interface elements (i.e., text fields, sliders, option menus) and their contents (i.e., thresholds, descriptions, sensor values). This sophisticated approach allows the presentation of collecting variants representing disclosure levels to end-users. The user interface elements and their contents are permanently synchronised in a way that any configuration change in the editor is reflected in the server and vice versa.

III. IMPLEMENTATION

In this section, we provide details on the software architecture of the *PPPSpace* environment consisting of the *PPPSServer*, the *PPPSEditor*, the *PPPSPresenter*, as well as the *PPPSClient* and the *PPPSMobileClient*.

The *PPPSpace* environment provides a sophisticated base technology for an event and notification infrastructure with technical scalability and with abstraction from the technical complexity for the end-users.

The mechanisms for persistent storing of data are implemented in the *PPPSServer* system that provides gateways for the exchange of data between the server and the other systems of the *PPPSpace* environment using XML Remote Procedure Call (XML-RPC) [24]. The *PPPSClients* and *PPPSMobileClients* permanently capture data from the environment through sensors and make use of the aforementioned gateways to submit the gathered data to the *PPPSServer*, where the data is processed in parallel and evaluated using inference engines. The *PPPSPresenters* are the primary nodes of the *PPPSpace* environment and provide the hardware for presenting audio and video streams as well as event data in actuators. The *PPPSEditor* allows users to configure the *PPPSpace* environment based on sensors, inference engines, and actuators.

Figure 1 shows the component diagram of the systems described in the subsequent sections. All systems are implemented in Java (Java 2 Platform, Standard Edition 1.5.0_16) [21] on Mac OS X.

Subsequently we provide details on the implementation of the central concepts of the *PPPSpace* environment.

A. Permanent Capturing

The permanent capturing of data is implemented in the *PPPSClient* and *PPPSMobileClient* systems. Both transmit the gathered data from sensors to the *PPPSServer* for persistent storing, distributing to subscribers, and further processing.

Within the *PPPSClient* system, the subsystem *PPPSCSensors* provides handlers for retrieving data from the respective user's computer (i.e., locally available sensors) and Web resources (i.e., shared workspaces and social networks). This *PPPSCSensorHandler* allows access to sensor implementations written in AppleScript [1] and Java [21]. Each *PPPSC-SensorHandler* instantiates *SensorTask* for retrieving data in parallel and according to end-user configurations. The *PPPSCGUI* provides users with an overview of active sensors.

The capturing behaviour of the *PPPSCSensors* permanently adapts to the users' changes of the configurations in the *PPPSEditor*. When end-users conduct changes to their configurations in the *PPPSEditor*, an event is transmitted on the *ConfigurationChanged* sensor to the *PPPSServer*. The *PPPSServer* notifies all *PPPSClient* clients that are subscribed for events from this sensor, using the described XML-RPC publish/subscribe mechanism. The *PPPSClient* clients retrieve recent configurations when the notification is delivered, and updates each *SensorTask*. For instance, if a user changes the collecting variant of a specific sensor in the *PPPSEditor*, the actual behaviour of this sensor is adapted accordingly.

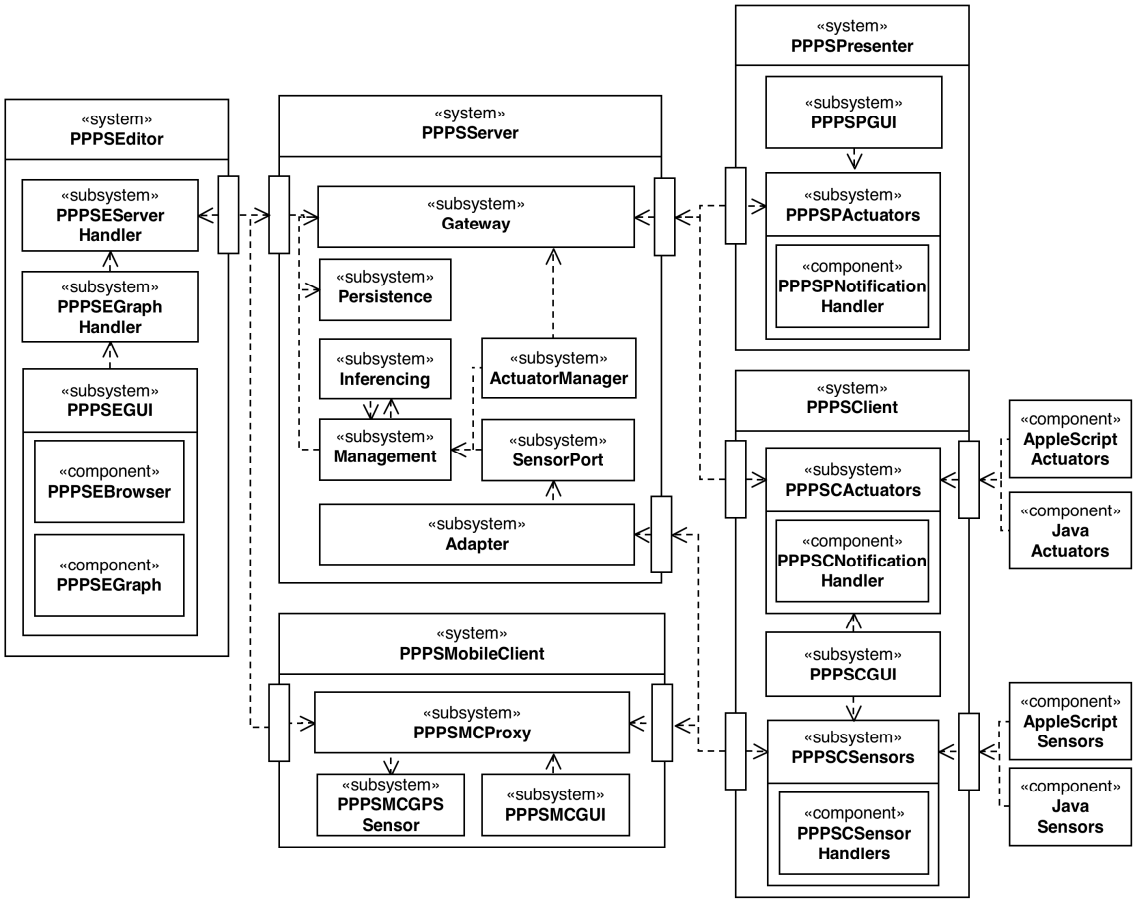

Figure 1. Component diagram of the architecture of the *PPPSpace* environment.

With the *PPPSMobileClient* system, the *PPPSMCGPSSensor* captures the data of the current location of the user with the big5 framework [13], and sends it to the *PPPSMCProxy*. The *PPPSMCProxy* is implemented using Java Server Pages (JSP) [23] and forwards that event data via XML-RPC to the PPPSServer.

B. Persistent Storing

The persistent storing of event data and user configurations is realised in the subsystems of the *PPPSServer*. The *PPPSServer* receives the gathered event data from the *PPPSClient* and *PPPSMobileClient* via its *Adapter* and *SensorPort* subsystems. The *Management* and *Persistence* subsystems of the *PPPSServer* provide advanced mechanisms to store data in a database.

Several concepts are implemented for optimised performance of the persistent storing. A generic database implementation in the *Persistence* subsystem supports parallel communication, and pooling of database connections and statements. It also provides access to several database management systems based on JDBC [22]. A caching mechanism in the *Persistence* and *Management* subsystems optimises frequent accesses to recent data, such as event data, sensor configurations, and actuator configurations. The caching mechanism is transparent to attached clients, which generally access getter and setter methods of the *Gateway* subsystem. Depending on the access interval, the *Management* subsystem delegates the query either to the cache (for short intervals) or to the database (for mid to long intervals).

The *PPPSEditor* provides a tool for managing user configurations. It is a specialised front-end for the *PPPSServer* and connects via the *PPPSEServerHandler* to the *PPPSServer* in order to store and retrieve user-generated configurations of *PPPSpace* environment sensors, inference engines, and actuators. Configurations consist of serialised XML data that is validated and deserialised by the *PPPSEGraphHandler*. From the XML description of a configuration, the *PPPSEGraphHandler* dynamically creates graphical representations of the environment components of a configuration. These representations are visualised in the editor's *PPPSEGUI* component *PPPSEGraph*, which is based on a `JGraph` user interface element [2].

C. Parallel Processing and Distributing

The parallel processing of event data in the *PPPSpace* environment is implemented in the *PPPSServer* and the subscribed *PPPSClients*, *PPPSMobile-Clients*, and *PPPSPresenters*. The *Management* sub-system processes incoming data; this includes the simultaneous processing in the *Inferencing* subsystem as well as the *ActuatorManager* subsystem.

Inference engines, implemented as components of the *Inferencing* subsystem, provide results of their processing operations to sensors that can be used throughout the *PPPSpace* environment, including the *PPPSServer* itself.

The *ActuatorManager* is responsible for the registration and management of actuators and for matching incoming event data. When event data matches an existing subscription, the matching data is published to the actuator client using the publish/subscribe mechanism, implemented in the *PublisherXMLRPC* as part of the *Gateway* subsystem. For each actuator, a `ThreadPoolExecutor` task is created for parallel notification distribution. *PPPSClients* and *PPPSPresenters* implementing actuators register as subscribers using their IP address, a reserved listening port, an actuator ID, and a remote method. Distributed notifications include the event data, which in turn ensure the parallel processing on numerous clients in the *PPPSpace* environments. The *PPPSMobileClients* act as actuator as well, but require a different mechanism for receiving and presenting event data as follows. The *PPPSMCProxy* actively requests event data from the subsystem *Gateway* of the *PPPSServer*. The user triggers this request on demand, in order to reduce network traffic and power consumption on the mobile device. The *PPPSMCGUI* then presents the event data in a simple list view in reverse chronological order showing the latest events on top of the list.

D. Dynamic Configuring and Sharing

The dynamic configuring and sharing of user configurations is implemented in the *PPPSServer* and *PPPSEditor* systems. Using the *Gateway* subsystem of the *PPPSServer*, the *PPPSEditor* retrieves data on available environment components.

For dynamic configuring each location in the *PPPSpace*, the *PPPSEBrowser* provides an overview of available sensors, inference engines, and actuators. The *PPPSEBrowser* is populated with data using the *PPPSEServerHandler*, which retrieves and parses environment component descriptions in XML format. The *Inspector* for accessing environment components is based on a dynamic user interface generation concept. Its user interface elements such as `JLabel`, `JTextField`, `JPasswordField`, `JcomboBox`, and `JSlider` are created from the evaluated XML description. These XML descriptions are based on information about collecting variants and the required user input fields for each user interface element.

Additionally to the dynamic configuring of environment components, the *PPPSEditor* also supports sharing—it notifies end-users when the same environment components are used in a similar way in different user configurations. Therefore, the *PPPSEServerHandler* subsystem of the *PPPSEditor* implements a parsing and matching algorithm for finding synergies in user configurations using an event-based XML pull parser, based on the Java StAX [6] standard.

IV. PPPSPACE DEPLOYMENT

The *PPPSpace* environment has been deployed in our workgroup and is currently being used. In this section we describe the technology setup and report on a performance evaluation.

A. PPPSpace Setup

The *PPPSpace* environment consists of hardware and software. The configuration that we are using is shown in Figure 2. The *PPPSServer* system is deployed on the *scml* server that is a Mac mini with PPC 1.2 GHz processor speed, 1GB of memory and Mac OS X 10.4.11 as operating system.

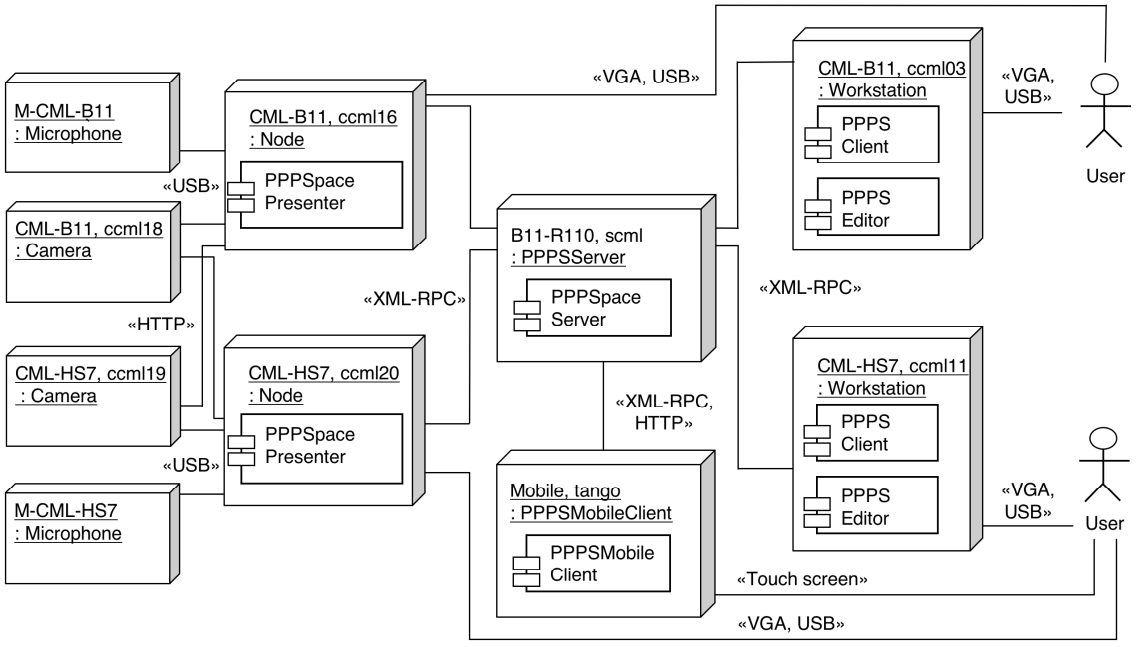

Figure 2. Deployment diagram of the PPPSpace environment as used in our workgroup.

The *PPPSClient* and the *PPPSEditor* systems are deployed on workstation computers (PowerMac G5, Dual 1.8 GHz processors, 1 GB of memory, running Mac OS X 10.4.11). The 14 workstation computers are connected to the server on a 100Mbit/s Ethernet campus area network (CAN) and are located in two offices of the workgroup's laboratory (i.e., CML-B11, CML-HS7). The *PPPSClient* and *PPPSEditor* are connected to the *PPPSServer* via XML-RPC for sending and receiving event data and user configurations. The software sensors (i.e., mail, calendar, shared workspace, music, notes, social network) are part of the *PPPSClient*. They permanently gather data and transmit these data to the *PPPSServer* according to the users' collecting variants. The *PPPSMobileClient* is deployed on an iPhone 3G extended with the big5 framework. The *PPPSPresenter* system nodes consist of:

- a dedicated Mac mini computer with the same system and hardware attributes as the *PPPSServer* (in our setting we use *ccml16* and *ccml20*)
- wide screen displays are wall-mounted to be prominently visible from all workplaces in the offices (ViewSonic 37" monitor with built-in stereo speakers)
- USB microphones to capture audio data (Logitech USB Desktop Microphones)
- network cameras to capture video data (AXIS 206)

The *PPPSPresenter* systems are connected to the *PPPSServer* using XML-RPC. The *PPPSPresenter* systems connect to the camera using HTTP streams. Both offices are connected using the 100 Mbit/s CAN.

B. PPPSpace Performance

In order to get an impression of the runtime behaviour of the *PPPSpace* environment we conducted two performance evaluations.

We had two distinct setups. In the first setup, a single event was sent to the *PPPSServer* on a sensor. We measured the latency for notifying all actuators registered for this sensor. We sampled latency values with different numbers of actuators. We started with 1 actuator and increased by 1 up to 25 actuators. In the second setup, a fixed number of 25 actuators were registered for a single sensor. We sent a varying number of sequential events in the range of 1 and 20 events, and measured the latency required for notifying all 25 actuators. For the evaluations, we used two MacBook Pro computers (2.2 GHz, 3GB memory, running Mac OS X 10.5.7, and Java 1.5.0_19) that were directly connected using a 1000 Mbit/s Ethernet local area network. One computer hosted the *PPPSServer*. The other computer implemented the actuators, sent the events, and wrote the calculated latencies to a file. For each evaluation we have conducted five cycles measuring latencies.

The results of the first setup, as illustrated in Figure 3, show an increasing latency with an increasing number of actuators. Overall, the maximum latency for notifying 25 actuators is below 65 ms. Across all five iterations, the difference between minimum and maximum latencies remained mostly low (<10 ms), with a peak difference of 17 ms in the medium range of 15 to 20 actuators.

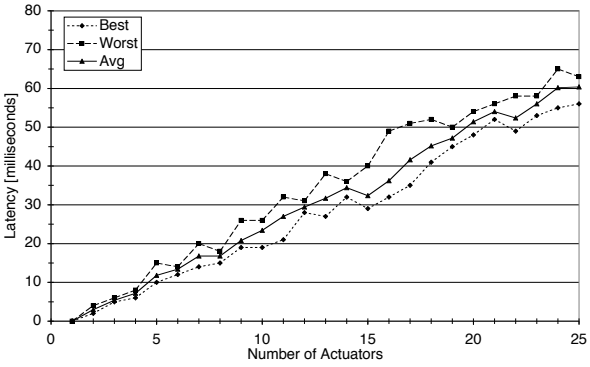

Figure 3. Performance chart of the PPPSpace event notification distribution for an increasing number of actuators.

The results of the second setup, as illustrated in Figure 4, show that the latency also increases with growing number of events. However, the average latency for notifying 25 actuators with 20 events is below 870 ms. The two outlier values (9 and 15 events) cannot be explained by the behaviour of the *PPPSpace* environment (they are probably results of network transmission latencies).

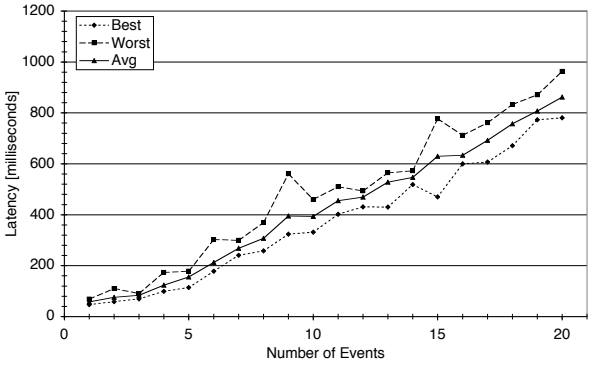

Figure 4. Performance chart of the PPPSpace event notification distribution for an increasing number of events.

As both evaluations show, the *PPPSpace* scales with an increasing number of clients implementing actuators as well as increasing number of events distributed to these clients. The small difference between best- and worst-case latencies indicates that the *PPPSpace* notification distribution mechanism is reliable. The multi-threaded mechanism works well for a large number of outgoing notifications that result from a limited amount of incoming event data. However, a large number of incoming event data results in increasing latencies for notification distribution. This is due to the sequential processing (one `for` loop in the Java program) of incoming event data, which presents a bottleneck for the parallel processing of event data (i.e., notification distribution and inference engines). It was clear for us that the sequential processing here is a bottleneck, but this systematic performance evaluation provides interesting details about it.

V. RELATED WORK

The *PPPSpace* combines concepts from media spaces, event notification infrastructures, and editors.

A. Media Spaces

Several media spaces provide connections between rooms and their users. Some allow selective information disclosure by users and are introduced below.

RAVE [7] provides audio and video channels connecting distributed users. In RAVE, up to 20 simultaneous connections between people in their offices are possible. Users can permit access to their audio and video channels. RAVE is limited to audio and video, and does not provide awareness information.

Notification Collage [8] is a media space with audio and video channels and awareness information. Its users can select sensors and manipulate simple properties (e.g., refresh rates, image sizes). It has limitations compared to the *PPPSpace*: it can only distribute data through a central server via broadcast to all presentation screens; and it can only present raw and unfiltered data.

Magic Window [16] is a media space that also provides audio and video channels for the users. Additionally, presence and availability information on the users is integrated into the video channel images. Users configure disclosure settings as fidelity levels that trigger blur image filter. So, Magic Window has advantages for information providers over Notification Collage, but has the same limitation for information recipients—it can only present unfiltered data.

The *PPPSpace* provides audio and video channels, and awareness information. Beyond the media spaces presented, the *PPPSpace* focuses on the fine-grained configuration of information disclosure using a graphical editor. Furthermore, the *PPPSpace* is rooted on a sophisticated sensor-based event notification infrastructure for the distribution of information.

B. Event Notification Infrastructures

Several event notification infrastructures have been presented. Some examples supporting cooperative work scenarios are introduced below.

Khronika [17] provides sensors for capturing data from the environment, an infrastructure for distributing events to interested users, and a simple editor for users to specify interests. Users express interests in events by constraints, and at runtime daemons check if the constraints match. While Khronika was a great and early infrastructure, it lacks sophisticated and easy to use end-user configuration.

The Elvin infrastructure is a notification service with publish-subscribe functionality, where users can subscribe to data of interest [5]. It provides elegant technology on a network infrastructure level, and allows users to have content-based selection of notifications. In Elvin event notifications are a represented as a tuple of attribute-value pairs. Elvin also lacks an easy-to-use editor.

The NESSIE awareness information environment [19] captures event data, analyses the users' current context based on the event data, and provides notifications of the users according to the current context. It provides a range of sensors, a centralised server with filtering mechanisms, and actuators for presenting information. Compared to the *PPPSpace* it provides sophisticated possibilities of retrieving event data from the server, but it lacks convenient end-user specifications of interests.

The iCAAS [4] allows access to data in wireless sensor networks. Its server provides fast delivery of filtered sensor data. However, further processing of the data needs to be implemented on the clients for each application.

The sensor-based platform Sens-ation [10] provides a framework for developers of sensor-based infrastructures. It consists of sensors, actuators and plugin mechanisms for inference engines. The CoLocScribe concepts [9] for publish and subscribe have been developed for Sens-ation. The CoLocScribe use case made use of a media space concept based on video. The *PPPSpace* provides innovative concepts for permanent capturing, persistent storing, and parallel processing and distributing that go beyond Sens-ation and CoLocScribe. It presents event data on specific nodes, workstation computers, and mobile phones in parallel. It integrates audio and video; and it provides an elaborated and easy-to-use editor for configuring sensors, inference engines, and adapters.

The *PPPSpace* offers—compared to traditional event notification as well as Sens-ation and CoLocScribe—a smooth integration of audio and video streaming with event capturing, processing, and presenting as well as with end-user configuration in an easy-to-use editor.

C. Ubiquitous Computing Editors

Ubiquitous computing editors provide abstractions for the environment components and devices. Some are based on simple metaphors and interaction techniques to be used by end-users and are sketched below.

The Jigsaw Editor [15] supports users in configuring domestic ubiquitous environment. By dragging user interface elements (jigsaw pieces) from the list view onto a canvas, users create compositions that interconnect real-world sensors and devices. Differences among the jigsaw pieces (some offer an output port, others an input port, some both) reflect the connection properties of the underlying devices. Overall, the selection of environment components is limited to domestic appliances and devices and provides no mechanisms for filtering or further processing of gathered data.

The eGadgets Editor provides a front-end to the GAS framework [18]. By means of connecting the user interface elements' inputs and outputs, users generate a wide range of scenarios consisting of home appliances. The GAS framework models individual environment components following a plug-synapse model, where each environment component offers a set of abilities and requests services from other environment components as well. The eGadgets Editor does not provide synergy notifications.

The iCap editor [20] allows users to prototype applications and scenarios for context-aware environments. Following a pen-based interaction technique, the system's environment components (input and output devices) may be interconnected to form a conditional rule-based construct in a user-friendly way. iCap allows users to draw their own sketches, which are used to represent the underlying devices within the editor environment. The iCap system does not provide synergy notifications.

In the *PPPSpace* environment, the *PPPSEditor* provides a flexible graphical front-end that reflects the numerous generic configuration options. It provides flexible synergy notifications and sharing of user configurations.

VI. CONCLUSIONS

In this paper we motivated the need for user-configurable awareness information as well as audio and video data. We introduced the advanced concepts and implementation of the *PPPSpace* for permanent capturing, persistent storing, and parallel processing and distributing.

The *PPPSpace* can be easily extended by adding new environment components. The user interface of the *PPPSEditor* dynamically adapts to the required properties of environment components. Users can extend the repository with new sensors, inference engines, and actuators; and realise scenarios of arbitrary complexity. It is also easy to include new sensors and actuators on the mobile phone by implementing new functionality for querying the sensors of the iPhone that have not been used yet (e.g., accelerometer).

The performance evaluation has shown that the *PPPSpace* environment overall scales in the deployment of our laboratory. Concerning the processing of incoming events, there is room for improvements. Future work will include innovative concepts for multi-threaded incoming event processing, while maintaining consistency among incoming data.

The integration of machine learning algorithms in the future presents an opportunity to provide actuators in the environment that reflect prevalent situations to a greater extent, but also represents a challenge for the parallel notification distribution mechanism.

ACKNOWLEDGEMENTS

We thank all members of the Cooperative Media Lab; part of the work has been funded by the Federal Ministry of Transport, Building, and Urban Affairs and by the Project Management Juelich (FKZ 03WWTH018). Thanks to the anonymous reviewers for valuable feedback.

REFERENCES

[1] Apple Computer Inc. AppleScript: The Language of Automation. http://www.macosxautomation.com/applescript/, 2009. (Accessed 20/7/2009).

[2] Benson, D. Java Graph Visualisation and Layout. http://www.jgraph.com/, 2009. (Accessed 20/07/2009).

[3] Bly, S.A., Harrison, S.R. and Irvin, S. Media Spaces: Bringing People Together in a Video, Audio, and Computing Environment. Communications of the ACM 36, 1 (Jan. 1993). pp. 28-47.

[4] Cinque, M., Di Martino, C. and Testa, A. iCAAS: Interoperable and Configurable Architecture for Accessing Sensor Networks. In Proceedings of the 3rd International Workshop on Adaptive and DependAble Mobile Ubiquitous Systems - ADAMUS 2009 (July 13-17, London, UK). ACM, N.Y., 2009. pp. 19-24.

[5] Fitzpatrick, G., Kaplan, S., Mansfield, T., Arnold, D. and Segall, B. Supporting Public Availability and Accessibility with Elvin: Experiences and Reflections. Computer Supported Cooperative Work: The Journal of Collaborative Computing 11, 3-4 (2002). pp. 447-474.

[6] Fry, C. and Slominski, A. StAX - Home. http://stax.codehaus.org, 2009. (Accessed 20/7/2009).

[7] Gaver, W.W., Moran, T., MacLean, A., Lövstrand, L., Dourish, P., Carter, K.A. and Buxton, W. Realising a Video Environment: EUROPARC's RAVE System. In Proceedings of the Conference on Human Factors in Computing Systems - CHI'92 (May 3-7, Monterey, CA). ACM, N.Y., 1992. pp. 27-35.

[8] Greenberg, S. and Rounding, M. The Notification Collage: Posting Information to Public and Personal Displays. In Proceedings of the Conference on Human Factors in Computing Systems - CHI 2001 (Mar. 31-Apr. 6, Seattle, WA). ACM, N.Y., 2001. pp. 514-521.

[9] Gross, T. and Beckmann, C. Advanced Publish and Subscribe for Distributed Sensor-Based Infrastructures: The CoLocScribe Cooperative Media Space. In Proceedings of the Seventeenth Euromicro Conference on Parallel, Distributed, and Network-Based Processing - PDP 2009 (Feb. 18-20, Weimar, Germany). IEEE Computer Society Press, Los Alamitos, 2009. pp. 333-340.

[10] 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.

[11] Gross, T., Paul-Stueve, T. and Palakarska, T. SensBution: A Rule-Based Peer-to-Peer Approach for Sensor-Based Infrastructures. In Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications - SEAA 2007 (Aug. 27-31, Luebeck, Germany). IEEE Computer Society Press, Los Alamitos, 2007. pp. 333-340.

[12] Gross, T., Stary, C. and Totter, A. User-Centered Awareness in Computer-Supported Cooperative Work-Systems: Structured Embedding of Findings from Social Sciences. International Journal of Human-Computer Interaction 18, 3 (June 2005). pp. 323-360.

[13] Holtwick, D. Web App Browser and Framework - Big5 for iPhone. http://www.big5apps.com/, 2009. (Accessed 20/7/2009).

[14] Hudson, S.E. and Smith, I. Techniques for Addressing Fundamental Privacy and Disruption Tradeoffs in Awareness Support Systems. In Proceedings of the ACM 1996 Conference on Computer-Supported Cooperative Work - CSCW'96 (Nov. 16-20, Boston, MA). ACM, N.Y., 1996. pp. 248-257.

[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 (Oct. 12-15, Seattle, WA). Springer-Verlag, Heidelberg, 2003. pp. 256-264.

[16] Kim, H.H.J., Gutwin, C. and Subramanian, S. The Magic Window: Lessons From a Year in the Life of a Co-Present Media Space. In Proceedings of the 2007 International ACM Conference on Supporting Group Work - Group 2007 (Nov. 4-7, Sanibel Island, FL). ACM, N.Y., 2007. pp. 107-116.

[17] Loevstrand, L. Being Selectively Aware with the Khronika System. In Proceedings of the Second European Conference on Computer-Supported Cooperative Work - ECSCW'91 (Sept. 24-27, Amsterdam, NL). Kluwer Academic Publishers, Dortrecht, NL, 1991. pp. 265-278.

[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] Prinz, W. NESSIE: An Awareness Environment for Cooperative Settings. In Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work - ECSCW'99 (Sept. 12-16, Copenhagen, Denmark). Kluwer Academic Publishers, Dortrecht, NL, 1999. pp. 391-410.

[20] 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 (Apr. 5-10, Fort Lauderdale, Florida). ACM, N.Y., 2003. pp. 974-975.

[21] Sun Microsystems Inc. J2SE 5.0. http://java.sun.com/j2se/1.5.0/, 2009. (Accessed 20/7/2009).

[22] Sun Microsystems Inc. Java SE Technology - Database. http://java.sun.com/javase/technologies/database/, 2009. (Accessed 20/7/2009).

[23] Sun Microsystems Inc. JavaServer Pages Technology. http://java.sun.com/products/jsp/, 2009. (Accessed 20/7/2009).

[24] UserLand Software Inc. XML-RPC Home Page. http://www.xmlrpc.com/, 2009. (Accessed 20/7/2009).