Cite as:
Peter Tandler: Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices. In: *Proceedings of UbiComp 2001: Ubiquitous Computing*. Heidelberg: Springer LNCS 2201, 2001, pp. 96-115.

Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices

Peter Tandler[1]

GMD – German National Research Center for Information Technology,
IPSI – Integrated Publication and Information Systems Institute,
AMBIENTE – Workspaces of the Future,
Dolivostr. 15, D-64293 Darmstadt, Germany
Peter.Tandler@darmstadt.gmd.de

Abstract. In ubiquitous computing environments, multiple users work with a wide range of different devices. In many cases, users interact and collaborate using multiple heterogeneous devices at the same time. The configuration of the devices should be able to change frequently due to a highly dynamic, flexible and mobile nature of new work practices. This produces new requirements for the architecture of an appropriate software infrastructure. In this paper, an architecture designed to meet these requirements is proposed. To test its applicability, this architecture was used as the basis for the implementation of BEACH, the software infrastructure of i-LAND (the ubiquitous computing environment at GMD-IPSI). It provides the functionality for synchronous cooperation and interaction with roomware components, i.e. room elements with integrated information technology. In conclusion, our experiences with the current implementation are presented.

1 Introduction

Ubiquitous computing environments offer a wide range of devices coming in many different sizes and shapes [45]. In situations where tight collaboration is necessary, users must be able to work synchronously with information shared among all these devices. Due to the heterogeneous nature of ubiquitous computing devices, their software infrastructure must provide a user interface taking advantage of their different properties. At the same time, it must enable tight collaboration of users working with different devices or sharing the same device.

We developed a software architecture that offers both flexibility and extensibility for different devices that are part of such ubiquitous computing environments. We used this architecture to create a software system called "BEACH", the Basic Environment for Active Collaboration with Hypermedia. BEACH provides the software infrastructure for environments supporting synchronous collaboration with many different devices. It offers a user interface that fits also to the needs of devices that

[1] Peter Tandler, until his marriage in the summer of 2000, was known as Peter Seitz.

have no mouse or keyboard, and which require new forms of human-computer and team-computer interaction. To allow synchronous collaboration BEACH builds on shared documents accessible via multiple interaction devices concurrently.

In the following section, requirements for the software infrastructure of a ubiquitous computing environment to support synchronous collaboration are discussed, also pointing to related work. Based on these requirements, the proposed architecture has been designed, which is presented next. Some example settings taken from our implementation of the architecture are used to show how it can be used. The paper closes with an overview of our experiences and ideas for future work.

2 Requirements for the Software Infrastructure

The software infrastructure of a ubiquitous computing environment with multiple heterogeneous devices has additional requirements compared to collaborative software running on distributed standard PCs. This section explains these requirements organized in five categories:

A. interaction with devices using different forms of interaction (section 2.1)
B. collaboration of users supported by a wide range of devices (section 2.2)
C. integration of devices in the environment (section 2.3)
D. support for different tasks (section 2.4)
E. hardware configuration (section 2.5)

To give a concrete description of requirements, among others, the example of roomware components (i.e. room elements with integrated information technology [38]) is used. The roomware components mentioned here have been developed in the context of the i-LAND project [36]. Similar environments are described in [1, 9]. The meeting context is used when application scenarios are given.

2.1 Interaction with Devices

In a ubiquitous computing environment, a variety of different interaction devices is available. Compared to a "traditional" desktop PC equipped with screen, mouse, and keyboard, these devices come in many different forms and support different styles of interaction, with the aim of providing "natural interfaces" [2].

Requirement A-1: Different Forms of Interaction. It is important for the software infrastructure to be open for different styles of interaction and extensible for future developments [24, p. 15 f]. As current operating systems and platforms only offer direct support for "traditional" interaction techniques and devices, a software infrastructure for a ubiquitous computing environment must allow the *integration of other device drivers* [3, p. 82].

Different interaction styles like pen, speech, or gestures, require the introduction of *new interaction models*. E.g. pen input cannot be dispatched to one single point at a display but might affect a wide area on the screen, while speech or gestures require

different processing levels. Here, higher levels influence the recognition steps made on lower levels, always being aware of possible ambiguity [17].

For each of these types of input, *abstractions* must be defined that can be easily mapped to the invocation of functionality [3, p. 81]. For mouse, keyboard, and pen input, events are a useful abstraction, but for other types of input, other concepts might be adequate [24, p. 24]. Other interaction techniques like hardware buttons (often found in PDAs) offer a very similar functionality compared to software button widgets. Therefore the same interaction model should be usable, but it must be possible that some kind of "button-pressed" events can be triggered directly by the device driver and flexibly mapped to software functions.

Requirement A-2: Adapted Visualization. Due to the different form-factors of interaction devices in a ubiquitous computing environment, displays appear in a broad range of different sizes and with different orientations. For differently sized devices different scaling factors, a different representation, or a different selection of objects must be used. This could take into account, whether the user needs an overview of the whole document, or just a part of the document is being edited. The more the devices differ, the harder it becomes to use a similar interface for all devices [3, p. 81; 24, p. 16], as other interface metaphors and concepts then become appropriate.

Another problem arises at an interactive table [36]: the orientation of the output does not necessarily have a common top-bottom/left-right for all users working at an InteracTable, as different users can look at the surface from different positions. At a traditional paper-based table, the users would simply rotate a sheet of paper to show it to someone else. At an interactive table, the same should be possible. In addition, the user should be able to keep a view of this object oriented towards her so that they can both look at the object with the preferred orientation.

Requirement A-3: Non-Visual Output. In analogy to the different input techniques (req. A-1), other output devices besides visual displays can be found in a ubiquitous computing environment. This can range from audio-based output [20] to ambient displays [13, 25].

While for different visualizations a common interface can be provided at a rather low level describing the visual appearance, another approach has to be taken here. One idea is the separation of the models for the *abstract* and the *physical* UI [44]. Depending on the used output device it might be possible to generate the physical user interface automatically from generic elements [24, p. 13]. In general, it is important, that all different interaction models use a common interface to the underlying functionality.

2.2 Collaboration of Users Supported by Devices

One characteristic property of a ubiquitous computing environment is the presence of many collaborating and communicating users and devices.

Requirement B-1: Multiple-Device Interaction. In a ubiquitous computing environment, a user usually has access to more than one computer. Within a meeting, a user might leave an interactive chair and walk up to a large public presentation surface to give a presentation. Here, the software must be able to detect and quickly change the assignment of the current user at these devices to give the user access to private information, for instance to the prepared material for the presentation.

The continuation of this scenario brings up a different case of multiple-device interaction: the user giving the presentation might have access to another device in parallel to the public display. To view her private annotations in addition to her slides, she uses e.g. an electronic lectern. Here, she uses several devices simultaneously with the same information displayed on both devices — but within a *different context* which influences the resulting view (different size, different level of detail, private annotations). This relates to the adapted visualization (req. A-2) where the context is defined by the *used devices* in contrast to the *usage of the device*.

Many examples where a PDA-like device (personal digital assistant) is used concurrently with a digital whiteboard, a table, or PC are given in literature [9, 22, 27, 28]. The PDA is used to have access to additional information or functionality without wasting space on the main display. On public displays, the PDA can be used for private information [11] or for functionality only relevant for its user. In these cases, both devices show different information and offer a different functionality.

Requirement B-2: Collaboration with Different Devices. The situation becomes even more complicated in the case of multiple users working together. Here, standard methods of shared editing cannot be used. E.g. WYSIWIS ("What-You-See-Is-What-I-See" [34]) would require that all collaborating users have coupled workspaces of exactly the same size in pixels, which is not possible if the devices cover a display size from very small to very large.

Fig. 1. The CommChair allows interacting remotely with documents displayed on the DynaWall. In addition, a private workspace can be accessed

Instead, the software must allow even tightly coupled components to use different view properties, but ensure that the users get a representation that fits to the current working mode. A user in a "CommChair" working on a shared workspace together

with a user at a "DynaWall" (fig. 1) will need both an overview representation of the whole workspace content and a second, zoomed view to work with. If the CommChair is located directly in front of the DynaWall so that the user can see the overview there, the overview representation displayed at the CommChair can be shrunk much more, as it is only needed for navigation.

Requirement B-3: Multiple-User Devices. Some devices like interactive tables or walls offer another challenge for the software: several people can use one device together and interact simultaneously with a single device. This is often called Single Display Groupware (SDG) [7, 22, 35].

In addition to SDG, multiple users at one display can collaborate with multiple users at other displays. This leads to a n:m relation between collaborating users and devices.

Software running on this device must therefore be able to receive events from several input-streams, to recognize input from different users, and to track several concurrent event sequences. Examples for event sequences are the drawing of a stroke or the dragging of a window [12].

Beside these technical issues, the user interface should be designed to allow the interaction of multiple users without interference.

2.3 Integration of Devices in the Environment

Another important area of requirements arises from the integration of devices and software within the working environment or working context.

Requirement C-1: Context Awareness. Software being aware of its context can act depending on the state of the surrounding environment. Common examples of context are the current location of devices and specific users [2, p. 35ff], but also the kind of device a software application is actually running. Besides the physical environment, other contextual information like the current task or project could influence the behavior of the software, as far as it is available to the software.

The software infrastructure must therefore maintain a representation of the current context. To be able to update this representation, an interface to sensors collecting context information (distributed all over the environment) is needed. If context changes are detected, mechanisms must exist to inform the application. Similar to what was described for different input devices, the data collected by the sensors will normally need preprocessing in order to generate information on a layer of abstraction useful for the application [29].

Requirement C-2: Physical Interaction. Since the configuration of physical objects in a meeting room strongly depends on the current work mode of a team, changes made to "real" objects can be used to trigger actions of the software. It is especially useful to reflect adaptations made by users to the setting of devices, due to changes of the current collaboration mode.

There are cases where a state change of the software is essential to maintain the consistency of the "real" and the "virtual" parts of the world ("augmented reality"). For example, "ConnecTables" are small interactive tables that can be assembled quickly to yield a larger homogeneous interactive area if desired [40, 43]. This is useful to support flexible splitting into and re-joining of subgroups. Here, the software must be capable of *dynamic changes* to the size and format of the currently available visual interaction area, and it is necessary to reflect these possibilities in the conceptual design of the user interface.

2.4 Support for Different Tasks

An interview study that we carried out found that creative teams have several recurring tasks [39]. Consequently, the software should offer dedicated help for a selected set of such tasks, which should be extensible to meet future needs [18].

Requirement D-1: Generic Functionality. Many functions are common to a wide range of application scenarios. This functionality should be reusable.

Requirement D-2: Tailorable Functionality. Important examples for typical group tasks that should be supported are creative sessions, presentations, meeting moderation, and project or task management. To be able to provide tailored support, a module concept should be available that is capable of extending the generic functionality. Of course, this should be possible without the need to change existing code and without interference to other modules.

Requirement D-3: Capture and Access Information. Another recurring task is that of gaining access to previously generated information. Within a context-aware ubiquitous computing environment, a lot of information can be captured automatically in addition to what is generated manually [1, 2]. For both capturing and accessing this information, dedicated support is needed.

2.5 Hardware Configuration

The restrictions of currently available hardware also place some requirements on the software.

Requirement E-1: Multiple-Computer Devices. What a user perceives as one *device* might actually consist of many individual hardware components. Some roomware components (e.g. the DynaWall, see fig. 1) are composed from several segments, each run by a separate PC. Due to the limitations of the available hardware (currently, each SMART Board [33] can only recognize a single pen position at any one time) this configuration allows each segment to receive pen input by one user each, thereby supporting many users simultaneously. To give the user the impression of a homogeneous interaction area, the segments must therefore be coupled via

software. This enables multiple users to collaborate on the same visual interaction area in spite of the limitations of hardware or physical space.

Requirement E-2: Dynamic Configuration. During a meeting it often happens that several independent problems are identified that have to be solved in parallel. In such situations, a team usually divides into a set of subgroups, each trying to solve one of these problems. After a defined amount of time, the team forms a plenary again and all solutions are presented. This scenario shows that different kinds of collaboration modes must be supported within a ubiquitous computing environment, each demanding a different configuration of available devices. The dynamics of a meeting must therefore be reflected in the design of the software, which should be flexible enough to give a team the necessary freedom to work efficiently.

3 Architecture of the Software Infrastructure

With respect to the requirements described in the previous section, an architecture has been developed that offers the flexibility necessary for supporting heterogeneous devices and ensures the extensibility for future devices. This section first gives an overview of the different layers and models before discussing the layers and their duties in more detail.

3.1 Architecture Overview

In order to provide reusable components on the one hand and hooks for adaptations for different devices on the other, the architecture is organized in four horizontal layers defining different levels of abstractions. Orthogonal to these layers, the architecture is structured by five models separating basic concerns within each layer (fig. 2).

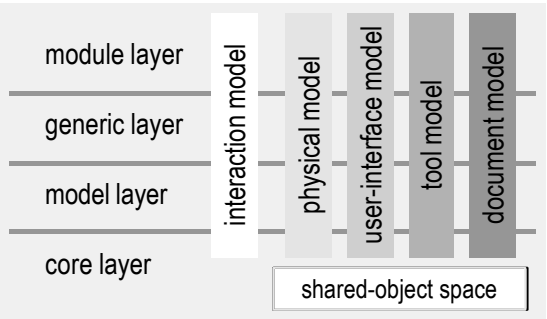

Fig. 2. The software architecture is horizontally organized in four layers defining different levels of abstractions and vertically by five models separating basic concerns. Crucial for synchronous collaboration is a shared-object space provided by the core layer enabling distributed access to objects

Levels of Abstraction. As mentioned, the proposed architecture is organized in four layers defining different levels of abstractions. The *module layer* (section 3.4) is most specific containing modules that provide tailored functionality for distinct tasks (req. D-2). It can be used to extend the functionality defined by the *generic layer* (section 3.3), which contains generic components that provide the basic functionality necessary in most teamwork and meeting situations (req. D-1).

While the two top layers consist of components perceivable by the user, the two lower layers are introduced to structure the application and ensure extensibility. The *model layer* (section 3.2) specifies the basic structure for the two top layers by defining interfaces for documents, tools, user interface, the physical environment, and interaction styles as the common abstractions for all devices (req. A-1, A-3, B-2, C-2). Thus, it can be seen as the implementation of the five basic concerns that are separated by the architecture (see below).

The *core layer* (section 3.5) offers a specialized infrastructure making the implementation of the higher layers easier (req. A-2, B-3, D-2). Most important, it provides *shared objects* that are crucial to allow distributed access from multiple computers (req. E-1, B-1).

Basic Models. To ensure a clear separation of concerns, models for document, tools, user interface, interaction, and physical environment are distinguished (fig. 2). While the interfaces and abstract classes used to implement these basic models are defined by the model layer, parts of the core layer can also be structured according to these concepts.

The *document model* defines the base classes and functionality of all objects that can be part of a document (req. D-1).

The *tool model* describes the elements that are directly attached to the user interface, providing additional functionality to the documents. In addition, the tool controls the possible work modes like the degree of coupling.

The *user interface model* is needed to define an alternative user interface concept suitable for different devices (req. A-1). Furthermore, multiple-computer devices (req. E-1) require that the user interface elements are part of the shared-object space. This enables user interface elements to be distributed among several computers.

The *physical model* is the representation of relevant parts of the "real" world. The two top layers can define physical models for interaction devices (req. A-1) or other objects to be monitored (req. C-1). These can be adapted dynamically if sensors recognize changes (req. E-2).

To be able to support different styles of interaction (req. A-1, A-3, C-2), the *interaction model* specifies how different interaction styles can be defined. The term used here describes a part of the software architecture, and should not be confused with the "interaction model" describing the "look and feel" of a user interface at a conceptual level as defined in [4].

While document, tool, user interface, and physical model are implemented as shared objects to give several users or devices the possibility to access these objects simultaneously, interaction model objects are always local to each machine. This allows each client to adapt the interaction style according to its local context, especially its physical environment and interaction capabilities (see examples in section

4). To connect the interaction model objects to the other models, the architecture uses the constraint mechanism described below.

A more extensive discussion of these models is given in [42].

3.2 Model Layer: Basic Separation of Concerns

The aim of the model layer is to provide an implementation of the basic models to be used as the basis for the implementation of the higher layers. This is important to ensure extensibility and interoperability of both generic components and modules.

For example, the interaction model defined by this layer could specify to use the *model-view-controller* (MVC) concept [14] to separate the handling of input and output. In this case, the model layer would contain not only the base classes for views and controller, but also the code necessary to create, update etc. the views and to dispatch events to specific controllers. For visual-based interaction, BEACH uses an adapted version of the model-view-controller concept (see section 3.5).

Similarly, the document model separates the *domain model* (sometimes also called "business domain objects" or just "model objects") and the *application model* [15, 41]. Domain models represent entities of the domain world. Application models are used to describe all application aspects such as presentation and manipulation of domain objects. For, e.g., a "text" object, the domain model includes the string describing the text and text attributes like font or size. The application model adds the editing state for text, like cursor position or selection. The shared editing state gives the ability to provide awareness, e.g. to display cursors of other users [31]. In addition, it specifies the degree of coupling between different users, i.e., which parts of the editing state are shared by all users and which allow private values. The workspace application model allows for instance different rotation of the workspace for two users working at an interactive table (see req. A-2), while all other properties are tightly coupled.

3.3 Generic Layer: Generic Collaboration Support

One important goal of every software system is to provide generic components that are useful in many different situations and for different tasks (req. D-1). To illustrate the generic components layer, some examples taken from BEACH are presented, which are further elaborated in section 4.

The basis for *documents* created with BEACH is a hypermedia data model. The generic document elements include hand-written input (scribbles), texts, images, and links as basic objects constituting information and workspaces to structure information (the equivalent of a page).

The *tools* currently realized by the generic layer of BEACH are toolbars and document browsers. Document browsers have a special role in defining the connection between the user interface and the document, i.e. specifying which part of the document is shown where, also offering possibilities of navigating in the document to a different workspace.

The main elements of the *user interface* of BEACH are segments and overlays [26]: the complete visual interaction area of a roomware component can be partitioned into "segments", which define the space available for a tool, e.g. a document browser. In addition, "overlays" reside in front of the segments in the background and can be positioned freely and are used in a similar way to the windows of most popular operating systems. They also contain a tool, but they would normally be used for toolbars and other smaller tools that have to be at hand for the user all the time.

One important part of the representation of the *physical environment* is the configuration of roomware components. A "station" refers to computers running a BEACH client. To be able to combine several stations to a composite roomware component (req. E-1, C-1) the current setting is available as shared objects, as shown in figure 3. A roomware component consists of one or more stations. Each station can have a display. The displays of all stations belonging to a roomware component are combined to a display area, which represents the complete interaction area of the roomware component, e.g. the complete area of a DynaWall (see fig. 1).

If displays are added to or removed from the display area, the views showing it will immediately adjust the size of the available area (req. E-2) due to the dependencies between the physical model and the views.

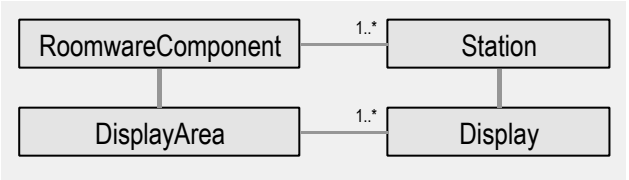

Fig. 3. The display area combines all displays of its roomware component's stations to one homogeneous interaction area

To provide an adequate *interaction* with roomware components, the traditional mouse-based interaction has been extended with support for gestures written with a pen (req. A-1), support for audio output is currently being implemented [20]. To generate the gesture events needed to handle pen input, each stroke that is drawn is sent to a gesture recognizer to check whether it is similar to one of the set of supported gesture shapes. In this case, a gesture event is generated and dispatched. In contrast to mouse events, which refer to a specific point, a gesture event is associated with a stroke — which could cross the bounds of multiple view objects. Therefore, a dispatcher for gesture events has been implemented that is capable of selecting the right view's controller.

3.4 Module Layer: Tailored Support for Tasks

The generic elements that are provided by BEACH are useful in many different situations. For some tasks it is of help if specific support is given (req. D-2). Therefore, the proposed architecture has a module layer, which allows modules to add further model elements and to extend the functionality of existing components. By providing hooks

already in the core layer to add new toolbars and services, modules can be plugged into BEACH without having to change existing code.

At present, only one BEACH module is available. It provides support for creative teams to collect ideas during brainstorming sessions. Ideas generated between sessions can be collected using a PDA and transferred to a public roomware component (e.g. a DynaWall) in the next meeting [16] (similar to [11]).

3.5 Core Layer: Specialized Infrastructure

The aim of the core layer is to provide functionality that will make the development of the higher levels more convenient. This includes

- synchronous access to distributed objects,
- automatic update and dependency detection,
- multi-user event handling,
- view transformations,
- device and sensor management, and
- module and services interface.

Here, we will focus on the first four items, as these are of main interest within the scope of this paper.

Shared-Object Space. In order to provide computer support for synchronous collaboration a platform for the distributed access to shared objects is needed (requirements B-1, B-2, C-1, and E-1). This section does not discuss the properties of different groupware frameworks and toolkits; it rather highlights the important features of the software infrastructure of a ubiquitous computing environment.

BEACH uses a replicated model, as some roomware components are connected via a wireless network with a rather low bandwidth of currently 10 Mbps shared by all connected clients (fig. 4). After an initial replication, only incremental changes to the shared objects have to be transmitted, thus reducing necessary communication. A server synchronizes all replicates of shared objects and ensures persistency. To minimize the coordination overhead, objects are grouped in “clusters” being the atomic elements for replication.

Transactions are used to guarantee consistency in spite of concurrent changes to objects. As a ubiquitous computing environment is highly interactive, it is important to ensure a fast response of the user interface. Avoiding delays waiting for the server’s commit, optimistic transactions offer a significant speedup whenever conflicting actions are unlikely or harmless.

Automatic Update and Dependency Detection. As changes to shared objects can be initiated by an arbitrary computer for a variety of reasons, it is very important that mechanisms are provided to trigger updates automatically when the state of shared objects changes (req. C-2, E-2).

Therefore, a declarative description of views (or other kinds of output objects) is used. The dependencies between views and attributes of shared objects are automati-

cally detected and re-computation is triggered whenever these attributes are changed [30]. When, e.g., the attribute 'color' of a workspace is set to 'blue' while a view for this workspace is open somewhere, this view will be repainted, regardless who changed this value on which device. This is very similar to the constraints used in systems like Amulet [23], but works also for a distributed setting.

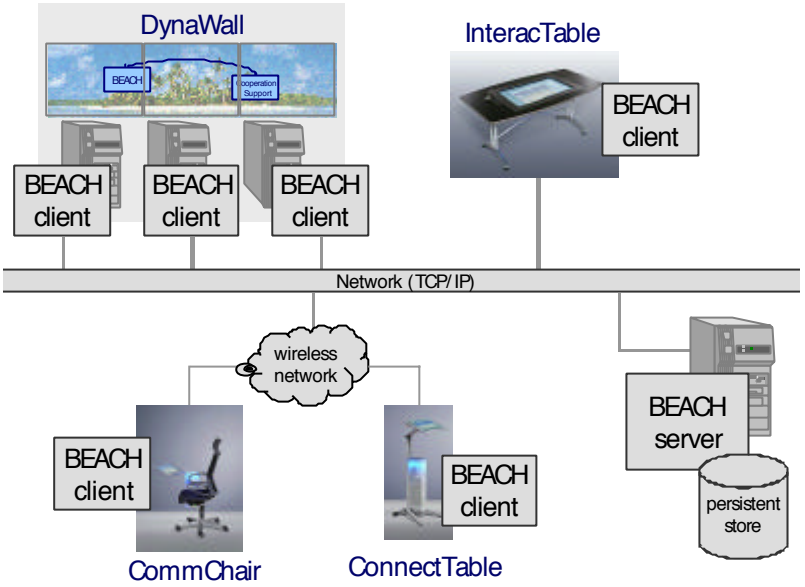

Fig. 4. BEACH clients running on different roomware component are synchronised by a server

Multi-user Event Handling. For multiple-user devices (req. B-3), it is necessary to provide an interface to hardware that is capable of handling multiple users at the same time using the same device. Multiple device drivers can send events, tagged with an identification of the originator, to BEACH. As an alternative to the standard mouse button-up or -down events, these can also be events like pen-up, pen-moved, or pen-down (req. A-1).

To support an adapted user interface for roomware components equipped with a pen, the events generated by the device drivers can first be assembled to higher level events. As it is very intuitive to draw strokes with a pen instead of just clicking on a document, pen events can be combined to strokes. For these strokes, gesture events can be generated depending on the shape drawn with the pen (like tap, line, circle [10]).

As different kinds of events need different strategies for dispatching, the event can choose an appropriate dispatching strategy. For example, key-pressed events are received by the controller having the keyboard focus, button-down or pen-down events are dispatched to all views at the mouse or pen position. Mouse-moved events are directly discarded — as they only have an effect after a button- (or pen-) down event.

To track several concurrent event sequences, the concept of "trackers" has been extended. A tracker is an object receiving events directly, without using the view hierarchy for dispatching. This is the same mechanism as is used by Myers' multi-user interactors [21]. BEACH is capable of handling several trackers at the same time by keeping a mapping of input device IDs to the different trackers, which will get all events from this device.

View Transformations. As views should be displayable in different orientations and sizes (req. A-2, B-2), depending on the current context, the core model replaces the standard "graphics context" (which handles the drawing) by an adapted version that supports transformations. A transformation is an object that responds to messages for transforming points and graphic primitives like images. These transformations are applied by wrapper objects which are inserted into the view hierarchy and which "wrap" the view to be transformed without needing to change it. A similar idea is followed by introducing the *portals* in Pad++ [5] or the *internal cameras* in Jazz [6].

4 Example Device Configurations

To illustrate how this architecture can be used for different devices and configurations this section gives three examples. The examples are taken from the experience with the implementation of BEACH, which was developed based on the proposed architecture. As BEACH is used as the software infrastructure for the roomware components we have built at GMD-IPSI, their characteristics determined the focus of BEACH. Currently, the roomware components support pen or finger as the main medium for input. Thus, BEACH emphasizes direct visual interaction. All roomware components have a permanent (in parts wireless, see fig. 4) network connection aiming to support synchronous collaboration, and no "slow" CPUs. This implies that BEACH is not appropriate for very small devices like PDAs and for devices not having a permanent connection to the network.

The examples show elements of the generic layer only, as this layer defines the concrete classes used to implement generic support for roomware components. First, the DynaWall is an example for a multiple-computer device (req. E-1). Second, the collaboration between a large public and a smaller private device (req. B-2) is shown in the case of a DynaWall in conjunction with a CommChair. And third, the InteracTable is used to demonstrate a device that can be used by multiple users at the same time, but with different viewing preferences (req. A-2, B-3). These examples focus on the four shared models, leaving the interaction model aside. Another configuration showing the relationship of the shared models to the interaction model using the example of ConnecTables is presented in [43].

4.1 Combining Multiple Computers to One Interaction Device

As mentioned before, the DynaWall (fig. 1) consists of three computers (multiple-computer device, req. E-1), each with an attached SMART Board [33]. Therefore, the

physical model defines the roomware component "DynaWall" (DWRWC in fig. 5) to consist of three stations (DWStation1 to DWStation3) with their displays combined to one large display area. While the three SMART Boards in our lab are mounted to one wall (fig. 1), the software allows changing this setting dynamically (req. C-2, E-2), e.g., when the boards are mobile and equipped with sensors (similar to the ConnecTable [43]).

If the display area is not divided into several segments (DWSegment1), it can be used to display one large workspace (DWWorkspace1) within one document browser (DWDocBrowser1). As mentioned above, the application model (DWWorkspaceApp1) is used to define the editing state and functionality for the workspace.

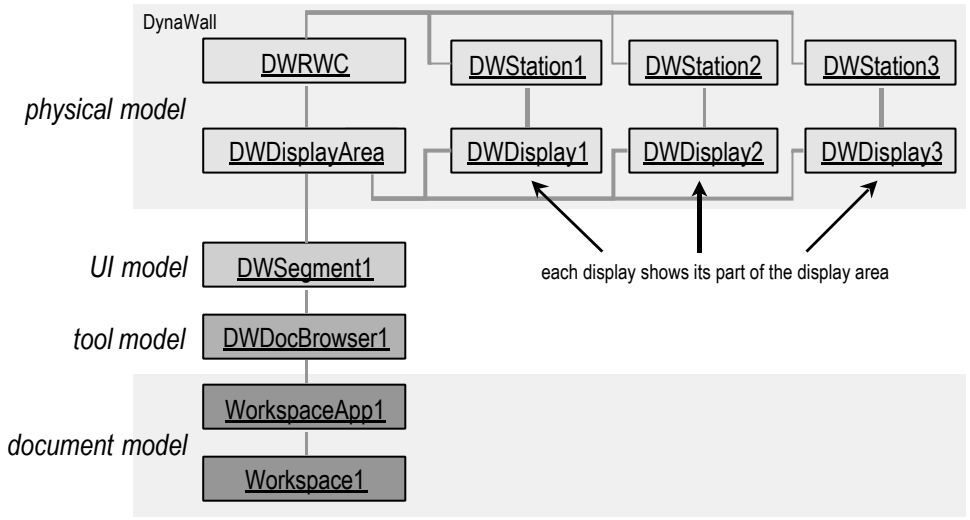

Fig. 5. The representation for a DynaWall consisting of three computers, which are combined to form a homogeneous display area. This allows the complete area to be used to show one large workspace

4.2 Tight Collaboration Using Two Different Devices

When a CommChair connects to the DynaWall specified in the previous example, it is interesting to see, on the one hand, how the tight collaboration between CommChair and DynaWall is implemented (req. B-2). On the other hand, it shows how the CommChair's display area is separated into two segments for public and private workspaces.

The display area, which consists of only one display in the case of a CommChair (CCRWC in fig. 6), is split into two segments (CCSegment1 and CCSegment2). While one segment is used to show a document browser (CCDocBrowser2) for the private workspace (Workspace2), the other connects to the document browser shown at the DynaWall (DWDocBrowser1). This enables very tight collaboration, as using a shared browser results in coupled navigation. As always the same application model is used,

all editing state is also shared between the DynaWall and the CommChair, which allows providing awareness information [31].

As the size of the segments at the DynaWall and the CommChair differs, the interaction model (not shown in figure 6) has to provide an appropriate mechanism to display the public workspace at the CommChair. Well know techniques are scrolling and/or zooming.

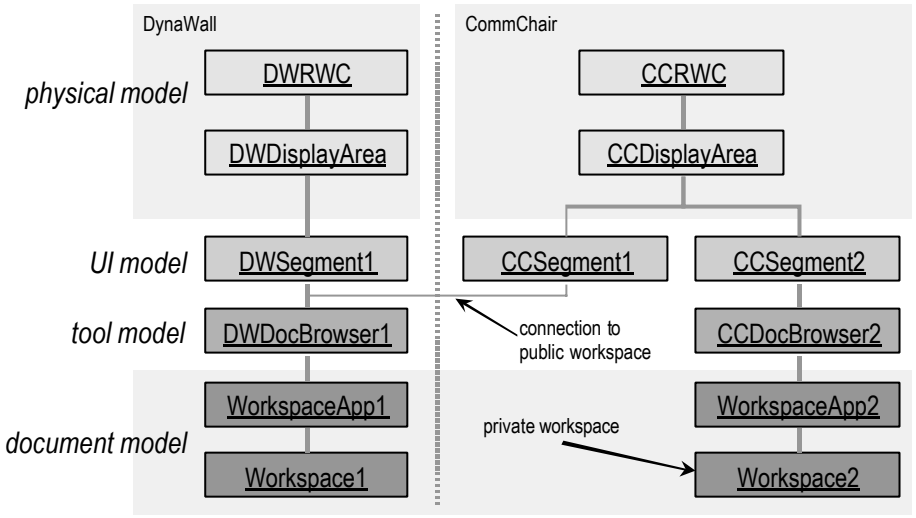

Fig. 6. The CommChair splits its display area into two segments. One connects to the document browser shown at the DynaWall, the other is used to show a private workspace

4.3 Multiple User Collaborating at One Device

While the CommChair in combination with a DynaWall enables collaboration of users with different devices, the InteracTable aims at supporting several users at the same device (req. B-3). As mentioned, it is necessary that horizontal displays require different orientation for users with different orientation towards the display (req. A-2).

To realize this, we used an approach where each user can open an overlay that can be freely moved around the display area, similar to a window (e.g. ITOverlay1 and ITOverlay2 in fig. 7). Each overlay gets an own document browser and workspace application model, but connected to the same workspace. In this case, the interaction model will open two views showing the same workspace. This allows each user to rotate her/his workspace to the preferred direction, as the rotation is specified by the workspace application model.

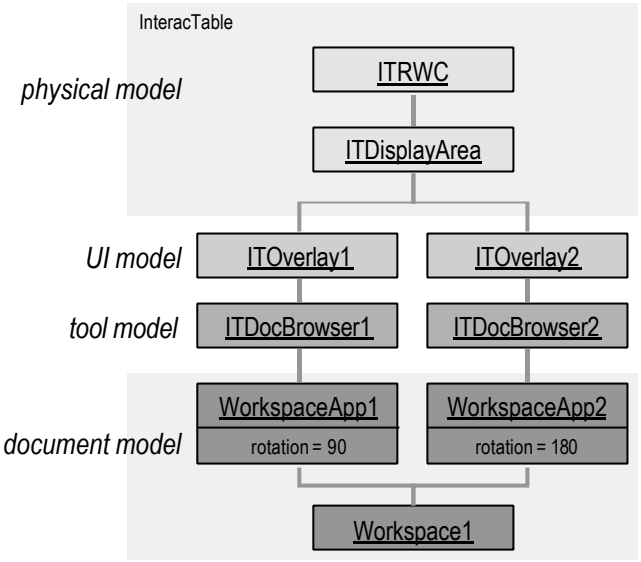

Fig. 7. Two user working at an InteracTable with the same workspace. By using separate browsers with separate application models, each user can look at the workspace with the preferred orientation

5 Current State and Experiences

The implementation of BEACH based on the proposed architecture shows that it helps to provide the functionality needed as the software infrastructure of the currently existing roomware components. BEACH is implemented using VisualWorks Smalltalk [8]. As shared-object space the COAST framework [30, 32] (developed at GMD-IPSI) has been chosen.[2] The former version of this framework was used in the DOLPHIN system [37].

Our experiences with the current version of BEACH are quite promising. We used the prototype at the international computer fair CeBIT at Hannover, Germany, in March 1999 to give interactive presentations using a DynaWall and a CommChair and for exploration by visitors. Since May 2000 there has been an installation of a roomware environment with a specialized version of BEACH at the German Occupational Safety and Health Exhibition in Dortmund (DASA). It is open to the public and visitors can experience "future work situations" for themselves. For our own work in the AMBIENTE team, we use BEACH both for internal and external presentations and for discussions and design meetings.

[2] COAST is now available as open source from http://www.opencoast.org.

5.1 Conclusions

Concerning the requirements identified in this paper, it is interesting to see how they influenced the design of the software architecture.

To be able to support different forms of interaction (req. A-1), high-level events that can be dispatched with an adequate strategy offer much flexibility. However, it does not free the developer from the task of defining what actions should be triggered by which controllers on which events. The extendable event dispatching mechanism is a good basis for supporting multiple persons using the same devices concurrently (req. B-3).

The implementation of cooperative applications on top of the COAST framework was very successful. Many error-prone tasks, like the conflict detection and handling caused by concurrent actions or the updating of multiple distributed views are carried out automatically by the framework. In case of brief network failures, queued transactions are transmitted when the connection is restored. However, if multiple other users continue working, the chance of rollbacks due to inconsistencies increases over time.

The adaptation of visualization (req. A-2) is only possible within a certain range of devices with similar characteristics. This factor is called "plasticity" of the user interface in [44]. It is important to note that some of the adaptations might have to change dynamically (req. E-2). This is enabled by using COAST's dependency mechanism for generating the visualization.

The shared object space provided by COAST together with a consequent separation and of application and domain models is the key to allow multi-device interaction (req. B-1) and composite roomware components (req. E-1). Since all models are implemented as shared objects stored on a server, clients can easily be restarted after crashes and resume exactly where they stopped.

Combining the shared object space and the dependency mechanism allows the modeling of dynamic changes of collaboration using different devices (req. B-2). For the integration of contextual information provided by sensors (req. C-1), this is also a flexible platform, as sensors need not be attached to the local machine, but can be connected to an arbitrary computer. Since the sensors are part of the shared-object space, their state can be monitored by any client, thus allowing state changes to trigger actions.

The possibility to include modules with added functionality in BEACH offers the specialization needed for certain tasks (req. D-2).

5.2 Future Work

Currently, we are developing new BEACH modules, e.g. for navigation and access to created information, and for integration of audio output. Other issues are the integration of mobile devices that are not always connected to the server and devices with restrictions on CPU and memory like PDAs. One approach to realize the connection to PDAs is to provide an interface to shared-event-based system as described in [9]. While asynchronous work is possible as long as different groups work on non-overlapping parts of the shared-object space, this is not feasible in practice. For asyn-

chronous, unconnected collaboration, it is essential to deal with inconsistencies. One solution could be the automatic creation of different versions for all objects changed while not connected.

The latest information about BEACH can be found at http://www.darmstadt.gmd.de/ambiente/activities/beach.html.

6 Acknowledgements

The author would like to thank all the colleagues and students of the AMBIENTE division at GMD-IPSI for helpful discussions and for the support with the time consuming implementation work. He especially thanks Norbert Streitz, Jörg Finger, Richard Stenzel, Thorsten Prante, and Carsten Magerkurth who helped to improve this paper.

References

1. Abowd, G. D., et al. (1996). Teaching and learning as multimedia authoring: the classroom 2000 project. In *Proceedings of the ACM Conference on Multimedia (Multimedia '96)*, pp. 187–198.
2. Abowd, G. D., Mynatt, E. D. (2000). Charting Past, Present, and Future Research in Ubiquitous Computing. *ACM Transactions on Computer-Human Interaction*, Vol. 7, No. 1, March 2000, pp. 29–58.
3. Abowd, G.D. (1999). Software Engineering Issues for Ubiquitous Computing. In *Proceedings of the 21st international conference on Software engineering (ICSE'99)*. ACM Press, pp. 75–84.
4. Beaudouin-Lafon, M. (2000). Instrumental Interaction: An interaction model for designing post-WIMP user interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI'00), ACM Press, New York, NY, pp. 446–453.
5. Bederson, B. B., et al. (1996). Pad++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics. *Journal of Visual Languages and Computing,* 7, pp. 3–31.
6. Bederson, B. B., Meyer, J., Good, L. (2000). Jazz: An Extensible Zoomable user Interface Graphics Toolkit in Java. In *Proceedings of User Interface and Software Technology (UIST 2000)*, ACM Press.
7. Bier, E., Freeman, S. (1991). MMM: A user interface architecture for shared editors on a single screen. In *ACM SIGGRAPH Symposium on User Interface Software and Technology, Proceedings UIST'91*, pp. 79–86.
8. Cincom Homepage. (2001) http://www.cincom.com
9. Fox, A., Johanson, B., Hanrahan, P., Winograd, T. (2000). Integrating Information Appliances into an Interactive Workspace, *IEEE CG&A*, May/June 2000, pp. 54–65.
10. Geißler, J. (1995). Gedrics: The next generation of icons. In *Proceedings of the 5th International Conference on Human-Computer Interaction (INTERACT'95),* Lillehammer, Norway, pp. 73–78.
11. Greenberg, S., Boyle, M. and LaBerge, J. (1999). PDAs and Shared Public Displays: Making Personal Information Public, and Public Information Personal. *Personal Technologies*, Vol.3, No.1, pp. 54–64, March. Elsevier.

12. Hourcade, J. P., & Bederson, B. B. (1999). Architecture and Implementation of a Java Package for Multiple Input Devices (MID). Tech. Report HCIL-99-08, CS-TR-4018, UMIACS-TR-99-26, Computer Science Department, University of Maryland, College Park, MD.
13. Ishii, H., Ullmer, B. (1997). Tangible bits: towards seamless interfaces between people. bits and atoms. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'97)*, ACM Press, New York, NY, pp. 234–241.
14. Krasner, G. E., Pope, S. T. (1988). A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk. *Journal of Object Oriented Programming*. 1(3), pp. 26–49.
15. Luo, P., Szekely, P., Neches, R. (1993). Management of interface design in humanoid. In *Proceedings of the ACM Conference on Human factors in computing systems (CHI'93)*, pp. 107–114.
16. Magerkurth, M., Prante, T. (2001). „Metaplan" für die Westentasche: Mobile Computerunterstützung für Kreativitätssitzungen. In *Proceedings of Mensch & Computer 2001*. Bad Honnef (Bonn), Germany, pp. 163–171.
17. Mankoff, J., Hudson, S., and Abowd G (2000). Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'00)*, ACM Press, New York, NY, pp. 368–375.
18. Moran, T., van Melle, W. (1998). Tailorable Domain Objects as Meeting Tools for an Electronic Whiteboard. In *Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work (CSCW'98),* ACM Press, pp. 295—304.
19. Müller-Tomfelde, C., Reischl, W. (1998). Communication Chairs: Examples of Mobile Roomware Components. *CHI '98 Summary. Suite on Late-Breaking Results: "The Real and the Virtual: Integrating Architectural and Information Spaces"*.
20. Müller-Tomfelde, C., Steiner, S. (2001). Audio Enhanced Collaboration at an Electronic White Board. To appear in *Proceedings of the 7th International Conference on Auditory Display (ICAD'01)*, Espoo, Finland.
21. Myers, B. A. (1999). An Implementation Architecture to Support Single-Display Groupware. Technical Report CMU-CS-99-139, CMU-HCII-99-101, Human Computer Interaction Institute, School of Computing Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891, May 1999, http://www.cs.cmu.edu/~bam.
22. Myers, B. A. et al. (1998). Collaboration using multiple PDAs connected to a PC. In *Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work (CSCW'98),* ACM Press, pp. 285–294.
23. Myers, B. A., et al. (1997). The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering,* 23(6), pp. 347–365.
24. Myers, B. A., Hudson S. E., and Pausch R. (2000). Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, Vol. 7, No. 1, March 2000, pp. 3–28.
25. Mynatt, E. D. et al. (1998). Designing Audio Aura. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'98)*, ACM Press, New York, NY, pp. 566–573.
26. Prante, T. (1999). A new pen-centric user interface to support creative teamwork in roomware environments (in German). Diploma thesis, GMD-IPSI, Darmstadt Technical University, Department of Computer Science.
27. Rekimoto, J. (1998). A Multiple Device Approach for Supporting Whiteboard-based Interactions. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'98)*, ACM Press, New York, NY, pp. 344–351.
28. Rekimoto, J. (1998). Multiple-Computer User Interfaces: A cooperative environment consisting of multiple digital devices. In Streitz, N., Konomi, S., Burkhardt, H. (Eds.), *Cooperative Buildings – Integrating Information, Organization and Architecture. First Interna-*

tional Workshop on Cooperative Buildings (CoBuild'98), Darmstadt, Germany, February 1998. Lecture Notes in Computer Science 1370. Springer: Heidelberg, pp. 33–40.
29. Salber, D., Dey, A. K., and Abowd, G. D. (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'99)*, ACM Press, New York, NY, pp. 434–441.
30. Schuckmann, C., Kirchner, L., Schümmer, J., Haake, J. M. (1996). Designing object-oriented synchronous groupware with COAST. In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work (CSCW'96),* ACM Press, pp. 30–38.
31. Schuckmann, C., Schümmer, J., Seitz, P. (1999). Modeling Collaboration using Shared Objects. In *Proceedings of International ACM SIGGROUP Conference on Supporting Group Work*, November 14-17, 1999, Phoenix, Arizona, USA, pp. 189–198.
32. Schümmer, J., Schümmer, T., Schuckmann, C. (2000). COAST – Ein Anwendungsframework für synchrone Groupware. Presented at net.objectdays 2000, Erfurt, Germany.
33. SMART Technologies Homepage. (2001). http://www.smarttech.com
34. Stefik, M., Bobrow, D. G., Foster, G., Lanning, S., Tatar, D. (1987). WYSIWIS revised: early experiences with multiuser interfaces. *ACM Transactions on Information Systems*. 2(5), pp. 147–167.
35. Stewart, J., Bederson, B. B., Druin, A. (1999). Single Display Groupware: A Model for Co-Present Collaboration. In *Proceedings of Human Factors in Computing Systems (CHI'99),* ACM Press, New York, NY, pp. 286–293.
36. Streitz, N. et al. (1999). i-LAND: An interactive landscape for creativity and innovation. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'99),* Pittsburgh, Pennsylvania, USA, May 15-20, 1999, pp. 120-127.
37. Streitz, N., Geißler, J., Haake, J., and Hol, J. (1994). DOLPHIN: Integrated meeting support across LiveBoards, local and desktop environments. In *Proceedings of CSCW '94,* ACM Press, pp. 345–358.
38. Streitz, N., Geißler, J., Holmer, T. (1998). Roomware for Cooperative Buildings: Integrated Design of Architectural Spaces and Information Spaces. In Streitz, N. et al. (Eds.), *Cooperative Buildings – Integrating Information, Organization and Architecture. First International Workshop on Cooperative Buildings (CoBuild'98)*, Darmstadt, Germany, February 1998. Lecture Notes in Computer Science 1370. Springer: Heidelberg, pp. 4–21.
39. Streitz, N., Rexroth, P., Holmer, T. (1998). Anforderungen an interaktive Kooperationslandschaften für kreatives Arbeiten und erste Realisierungen. In *Tagungsband der D-CSCW'98*. B.G.Teubner, Stuttgart, Leipzig, pp. 237–250.
40. Streitz, N., Tandler, P., Müller-Tomfelde, C., Konomi, S. (2001). Roomware: Towards the Next Generation of Human-Computer Interaction based on an Integrated Design of Real and Virtual Worlds. In J. A. Carroll (Ed.): *Human-Computer Interaction in the New Millenium,* Addison Wesley, pp. 551–576.
41. Szekely, P., Luo, P., Neches, R. (1992). Facilitating the exploration of interface design alternatives the HUMANOID model of interface design. In *Proceedings of the ACM Conference on Human factors in computing systems (CHI'92)*, pp. 507–515.
42. Tandler, P. (2001). Modeling groupware supporting synchronous collaboration with heterogeneous single- and multi-user devices. To appear in *Proceedings of CRIWG'01*, Sep. 6–8, Darmstadt, IEEE CS Press. http://www.darmstadt.gmd.de/ambiente/publications.html
43. Tandler, P. et al. (2001). ConnecTables: dynamic coupling of displays for the flexible creation of shared workspaces. To appear in *Proceedings of User Interface and Software Technology (UIST 2001)*, Nov. 11–14, Orlando, ACM Press. http://www.darmstadt.gmd.de/ambiente/publications.html
44. Thevenin, D., Coutaz, J. (1999). Plasticity of User Interfaces: Framework and Research Agenda. In *Proceedings of Human-Computer Interaction – INTERACT'99*, pp. 110–117.
45. Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM,* 36 (7), pp. 75–84.