Supporting Mobility and Collaboration in Ubiquitous Computing

Jakob E. Bardram
Department of Computer Science
University of Aarhus, Denmark
bardram@daimi.au.dk

Abstract: For unknown reasons, research into ubiquitous computing seems to pay no attention to collaboration among users in such an environment. This paper presents the design philosophy of activity-*based computing* (abc) and a technical implementation of it in a ubiquitous computing infrastructure, the *ABC framework*, which takes collaboration in ubiquitous environments as its starting point. The idea of activity-based computing and the aim of the framework is to: (i) support the *human activity* by managing its collection of work tasks on a computer, (ii) to support *mobility* by porting activities across heterogeneous computing environments, (iii) to support *asynchronous collaboration* by allowing several people to participate in an activity, and (iv) to support *synchronous, real-time collaboration* by enabling 'desktop conferring' by sharing the activity across several clients. During a period of two years, our framework have been co-designed and evaluated in cooperation with a range of clinicians in a hospital.

Introduction

Taking the ideas of Mark Wieser (1991) as a starting point there is a growing number of research projects that explore ubiquitous and pervasive computing. For example, the MIT project Oxygen aims to "communicate with people using natural resources, such as speech and vision", to support users on the move, and to monitor and control the environment (Dertouzos, 1999). The Aura project at Carnegie Mellon University (Garlan et al., 2002) encompass a range of projects and technologies that enables “mobile users to make the most of ubiquitous computing environments, while shielding those user from managing heterogeneity and dynamic variability of capabilities and resources” (Sousa and Garlan, 2002). The ‘Office of the Future’ project at Georgia Tech designs and builds an interface for individual office workers that uses projected visual displays to extend their virtual workspace (MacIntyre et al., 2001).

Common to most of the research into creating technical platforms for ubiquitous computing is the lack of support for cooperation. Within CSCW there is a growing body of research showing that cooperation is inherent to most human activities and work. For example, the studies of mobility and cooperation (Heath and Luff, 1998; Bellotti and Bly, 1996) show how mobility, cooperation, social awareness, and direct face-to-face interaction are closely interlinked. From a CSCW perspective it seems odd that ubiquitous computing projects, like Aura and Oxygen, focus solely on the mobility and resource handling of individuals moving between different computational environments. And that task awareness in the Office of the Future is mostly concerned with monitoring the status of individual tasks, like a printout of a document, or whether there is a reply to an email. None of these proposed infrastructures considers or implement support for direct cooperation among multiple users in a ubiquitous environment.

In this paper we present the concept of Activity-Based Computing (ABC) as a way of thinking about supporting human activities in a ubiquitous computing environment. We furthermore present our ABC framework, which is an implementation of the activity-based computing concepts. The ABC framework is a platform for the *development* and *deployment* of ubiquitous computing applications. In this paper we will focus especially on its support for *cooperation* among the people using the platform.

The key idea behind the ABC framework is a platform-independent support for user activities (Christensen and Bardram, 2002). Specifically, we describe an architectural framework for ubiquitous computing applications with the following key features. First, the ABC framework enables mobile users to make the most of ubiquitous computing environments, while shielding those users from managing heterogeneity and dynamic variability of capabilities and resources. Second, user activities become first class entities that are represented explicitly. Third, activities are inherently collaborative, treating single-users activities as collaborative activities that just happen to have just one participant. We argue that our concept of Activity-Based Computing and our current implementation in the ABC framework addresses some of the important, but overlooked, issues involved in creating cooperative support in ubiquitous computing environments. Furthermore, the concept of organizing computer support in terms of activities has turned out to offer simple, light-weight mechanism for solving some of the traditional hard problems in collaborative systems. For example, collaborative session management, including session creation, user entrance and departing, and floor-control, finds itself a

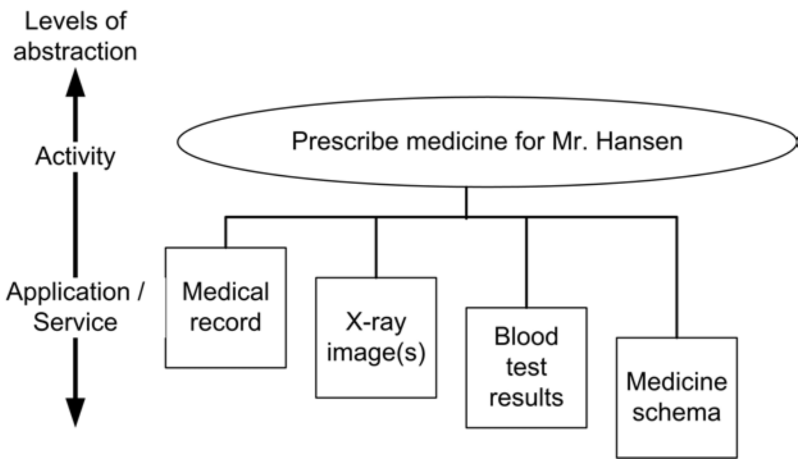

Figure 1: A single activity involves many services

finds itself a simple but robust solution by using the activity concept as a foundation for session management.

This paper presents our activity-based design philosophy, as embodied in the ABC framework. We begin by presenting the concepts of Activity-Based Computing by relating them to a scenario from a hospital. Then we go on to present the architecture of the ABC runtime infrastructure, which is the underlying mechanism for activity-based computing. We then discuss how this infrastructure enables asynchronous collaboration, mobility, and real-time 'desktop conferencing' – which we term 'real-time activity sharing'. Finally, we discuss related CSCW research before we end the paper with a discussion and a conclusion.

Activity-Based Computing

The background for developing the activity-based computing concepts is studies of healthcare practices (Bardram, 1997; Bardram, 1998; Bossen, 2002). There is a range of challenging properties of medical work, which makes it fundamentally different from typical office work: extreme mobility, ad-hoc collaboration, interruptions, high degree of communication, etc. This makes healthcare an interesting application area for the design of pervasive computing technology.

Healthcare has a long tradition of using computer-based systems, and a clinician is today faced with many different systems and even faced with a wide range of functionality within each one of them. Thus, carrying out a single activity typically involves a lot of different systems and a lot of specific functionality and data presentation within each system.

This is illustrated in Figure 1. If you ask the doctor what he is doing, he would answer "I'm prescribing medicine for Mr. Hansen". If you instead view it from the computational level, the doctor is actually handling several distinct services or

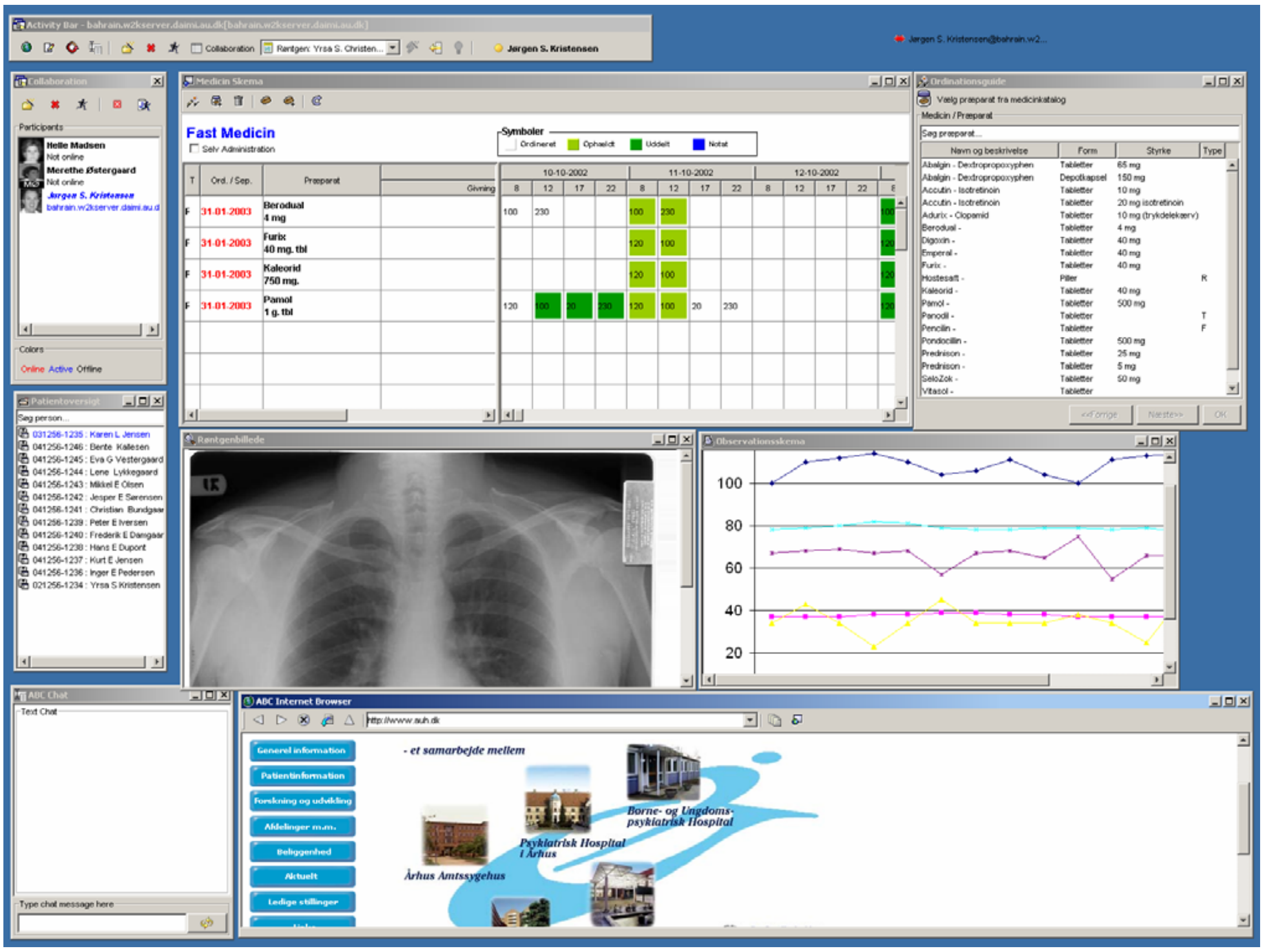

Figure 2. A typical setup of a medical activity used in a team conference situation. The Activity Bar and the Collaboration window is the user-interface part of the ABC framework.

applications: reviewing the medical history, looking over the medicine schema, studying the results of blood tests, viewing X-ray images, etc. Thus, we can identify at least two levels of abstraction, namely the higher level of human activities (the 'activity level') and the lower level of computational services/applications manipulated (the 'application level'). Our key argument is that contemporary computing systems do not support the activity level, only the application level. Our aim is therefore to explore how to support the activity level directly in the computing system; explore what the concept of "activity" is in this context, and to evaluate how activities may help clinicians in their daily work.

The ABC Framework @ Work

To illustrate how the ABC Framework achieves its goal of supporting user mobility and collaboration, we now describe how it realizes a typical scenario from our field-studies of work in large metropolitan hospitals in Denmark, namely the morning ward round.

The morning ward round starts with a team conference between the group of physicians and the nurses at the ward. In the ward's conference room, eight people (3 physicians and 5 nurses) sit around the interactive conference table, which has a large display build into it. There are several interactive whiteboards in the room. It is Monday morning and the nurse team-leader starts by making a report on the incoming patients that weekend. As she approaches one of the wall displays she is authenticated and her last *active activity* is displayed. This was an 'Office Activity' she had been working on Friday afternoon on the PC in the office, and it is containing *applications* like a word processor, a browser, etc. She *activates* the 'Ward Round Activity' via the *activity bar*. This brings up various views of the electronic patient record (EPR) normally used in briefing situations. For example, the textual record, medicine schema, blood-sample answers, X-ray images and descriptions, and a list of patients admitted to the ward (see figure 2). All persons in the room are *participants* in this activity, and some have a copy of the activity displayed on the interactive table in front of them. One nurse – Ms. Jensen – makes comments on care related issues and how a patient is progressing in his treatment. All her actions on the table are reflected on the wall. For example, when she calls forth a nursing care record, the record is shown on the wall display as well and her gestures made by her pen on the table are reflected via a *telepointer*. At a certain point when discussing the medication of a patient Dr. Kristensen decides to prescribe some morphine. As a nurse, the team-leader cannot make prescriptions in the EPR. Dr. Kristensen walks up to the wall display and by approaching he is authenticated, thereby enabling the prescription icon in the EPR.

During the conference Dr. Kristensen gets an *invitation* to participate in another activity – this is indicated in the activity bar as a blinking icon. A while later when the team conference is over, Dr. Kristensen checks his list of *incoming activities*, and together with the other physicians he walks up to the wall and activates the 'radiology conference' activity. The display changes into showing a large number of X-ray images and a live video link to the radiologist. Dr Kristensen explains that Dr. Buch isn't present and they agree on *recording* the conference session. The radiologist goes through the patients, describing his analysis of the X-ray images and is making *gestures* and *annotation* on the images. At the end, Dr. Kristensen hits the stop recording button and invites Dr. Buch. The next time Dr. Buch logs in, she will be able to play back the conference as it took place, including the voice and gestures made by all participants.

This scenario illustrates several central points in our view on activity-based computing and hence the support provided from the ABC framework.

- *User Activities are First Class Entities*: In Activity-Based Computing, 'computational activities' (or just activities) become first class entities that are represented explicitly in the framework. This means that activities are managed and persistently saved over time, is distributed across computational devices that can handle them, and are directly accessible by users in the user-interface. A user has a range of activities which s/he alternate between. Activities are *stateful* by maintaining and storing state information about the unfolding of the activity in time. An activity has a life cycle where they can be initiated, paused, resumed, and finalized. The activity concept thus supports *interruptions* and *multi-tasking* in work by allowing an activity to be paused and later resumed. Combining this statefulness with the distributed nature of activities implies that an activity can be paused at one host and later resumed and its state restored at another host. Hence, an activity is portable and hence supports *mobility*.

- *The Activity is light-weight*: An activity does not model nor control real-world human activities. A computational activity can be created and modified according to the desire of the user and do not come out of models of work activities.
- *An Activity is a Collection of Computational Services*: As illustrated in figure 1, an activity is a collection of services. For the activity to become stateful, each service needs to be stateful, i.e. be able to hand over state information to the activity. Services and applications are *decoupled*. Thus, there is not necessarily a tight coupling between a service described in an activity and an application running on a host device that can handle this service as part of executing an activity. In the scenario above, X-ray images are shown on the wall-sized display but when resuming this activity on a PDA, proper applications for image rendering might not be available, or might be replaces by a low-fidelity version. Thus, this decoupling between services and applications supports the usage of different applications for e.g. image manipulation on different platforms.
- *Activities are Collaborative*: Activities are inherently shared and thereby *collaborative*. An activity can have several *participants* as illustrated in the scenario above. The list of participants work as an access control list to the activity – only users on the participant list can resume the activity. The sharing of activities also provides an awareness of the state of the activity. A user who resumes an activity will get the latest state of it – hence s/he will see changes made by other participants. Furthermore, if two or more participants activate an activity simultaneously, they would engage in synchronous real-time 'desktop' conferencing by seeing what others are doing.

This last collaborative aspect of the Activity-Based Computing concept is the main focus of this paper as we shall turn to now.

The ABC Framework

The main goal of the ABC framework is to provide a programming platform for the *development* and *deployment* of computer applications that can be used in our activity-based computing concept. This is achieved by having on the one hand a *runtime infrastructure*, which handles the computational complexities of managing distributed and collaborative activities by adapting to the available services or resources in a specific environment. And on the other hand by having a developer's framework, which helps the programmer to create ABC-aware applications that can be deployed in the runtime infrastructure.

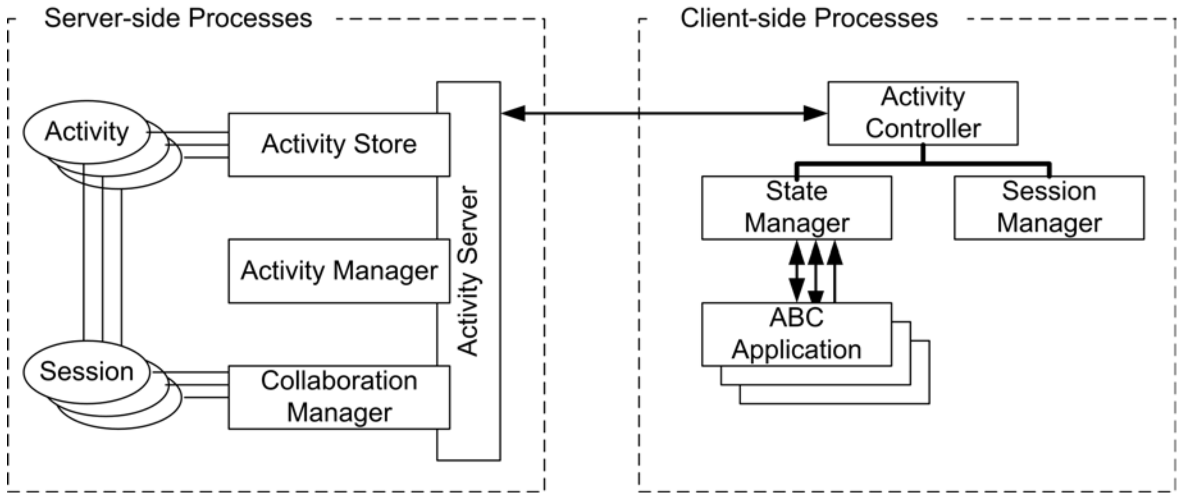

Figure 3. The ABC Runtime Infrastructure illustrating both server-side and client-side processes.

This section describes the actual runtime infrastructure that lies beneath the ABC framework. Its responsibilities as for activities is to manage, store, activate, and distribute activities, manage and distribute shared state information, ensuring synchronization methods on collaborative activities, and collaborative session management. Figure 3 illustrates the ABC runtime infrastructure. It consists of a range of server processes running on one or more servers and a range of client processes supporting the execution of the ABC applications.

The server part of the ABC infrastructure is build in a scalable manner and the different processes making up the Activity Server can thus be deployed on different machines. The ABC infrastructure consists of the following key processes.

- *Activity Store* – handles the persistence of activities by providing an interface to create, delete, and get activities and templates for new activities by reference or query. The store keeps track of the usage history for a user, enabling stepping forward and backward in the list of activities.
- *Activity Manager* – manages the runtime behaviour of an Activity by enabling activities to be created, initialized, paused, resumed and finalized by clients.
- *Collaboration Manager* – handles the real-time requirements for synchronous collaboration among active participants within an activity. To do this it manages a *Session* object for each ongoing collaborative activity currently activated by one or more users at different host machines, including the same user on several hosts. Basically, a Session notifies its active participants if the Session or its associated Activity changes. Typical changes are entrance, movement and departing of users in a session and changes to the state of an activity. Parties interested in listening to changes to a Ses-

sion can add a *Session Listener* to the Session. A central listener on Session objects is the client side *Session Manager* described below.

- *Activity Controller* – is the link between the client and the server. A client's Activity Controller registers at one or more Activity Managers and maintain a link to the *Activity Bar*, the user-interface to the ABC infrastructure (see figure 2), which via the controller gets a list of activities for a user. The Activity Controller can also be remotely controlled by the Activity Manager, which can force the client to change user, for example. The Activity Controller is also notified about relevant events from the server processes. For example, if the current user is invited to participate in another activity, the Activity Controller is notified and an appropriate signal can be made to the user via the bar. When an Activity Controller activates an activity, the local *State Manager* is notified, which in turn uses the *Registry* to looks up appropriate *ABC application*, which can handle the services collected in the activity.

State Management and Stateful Applications

A key design invariant in the ABC framework is that applications are *stateful* and hence can hand over and restore its own state. The runtime infrastructure collects, manage, distribute and synchronize this state information across the movement of users between physical machines and in the participation in synchronous collaborative sessions (see next section). The collection of state information from all applications running in an environment is saved in the activity and hence, the activity can be assumed to always contain the shared state. The State Manager is key to upholding this invariant. The State Manager is a singleton process running on a client in the ABC infrastructure. It creates the link between the Activity Controller (and hence the Activity Manager(s) running as server processes) and the applications running on the client machine.

Figure 4 illustrates the interaction between the ABC processes when resuming and pausing activities. When an ABC-aware application starts up it register itself it the Registry, telling what kind of service it can handle. This is done using RMI and the ABC application can therefore run in a separate Java virtual machine, potentially on another physical machine. When an Activity is activated via the Activity Controller, the State Manager is notified and extracts from the activity's shared state the application-specific state information and hand it over to the applications. This also includes launching and tearing down applications. The application itself is now responsible for restoring its state. The application is basically given back the state information it has handed over previously. Similarly, when an activity is de-activated the State Manager asks for state information from all the running applications, which is then handed over to the Activity for storage and distribution.

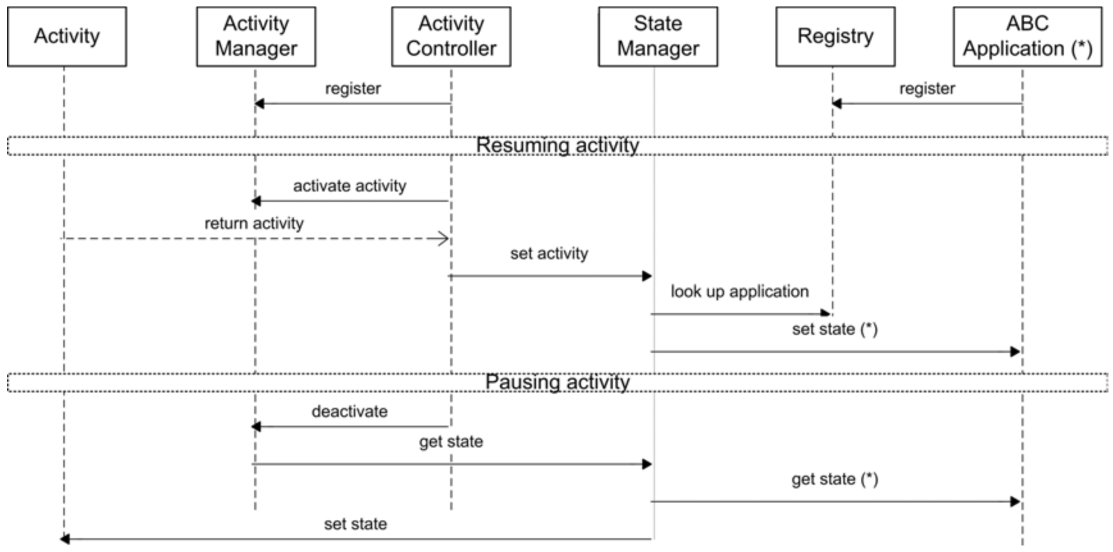

Figure 4. Interaction diagram showing the collaboration among ABC processes when restoring and saving state information. Multiple instances and calls are marked with (*).

There is no default application-specific state information saved by the ABC infrastructure. It is completely up to the application itself to hand over enough state information to the State Manager for it to restore its current state later, when asked to do so. The programming environment however provides several premade state classes that can handle the most common types of state information, like the position, size, and layout of a window frame. A similar approach is used in DistView (Prakash and Shim, 1994).

In the design and implementation of the shared state management of the ABC framework we are inspired by the work done in the Corona Server (Hall et al., 1996), especially the services for managing shared states in collaborative systems (Shim et al., 1997). In 'Corona' terms our current implementation of shared state is *persistent* (i.e. the collaborative session exist even though there are no active users), *memory-less* (i.e. an update to a shared state replaces the old one), supports *multi-cast* of state information (both sender-inclusive and sender-exclusive), and applies *optimistic state synchronization*, like the Corona server. The latter comes from observations of medical conferences, where concurrency conflicts are rare because people mediate interaction themselves (see also Greenberg and Marwood, 1994). Our implementation makes no distinction between *stateful* and *stateless* group collaboration, as the Corona server does. In the ABC framework each individual application can implement it own state management policy, ranging from no state management to complete state handling.

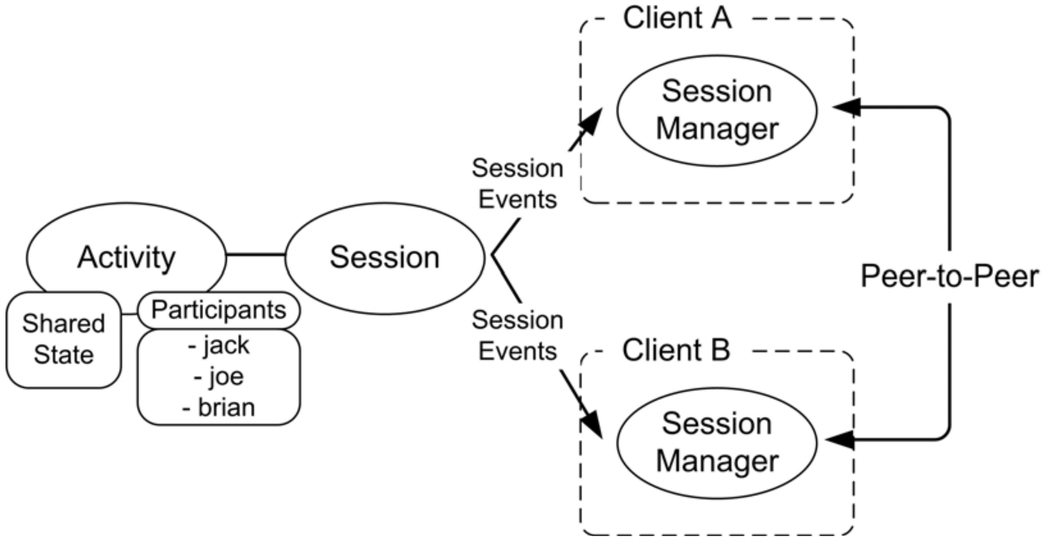

Figure 5. The interaction between the Activity, the Session, and the Session Manager at each client.

Collaboration Management in Real-Time Activity Sharing

An important part of the Activity-Based Computing concept is the support for collaboration – synchronously as well as asynchronously. We have already discussed how the Activity, the State Manager, and the stateful application concepts support asynchronous collaboration by allowing users to alternate in activating activities, they participate in. In this section we will described the synchronous, real-time collaboration aspects of the ABC framework, which we term *real-time activity sharing*. By real-time activity sharing we mean that when two or more participant of an activity has activated this activity simultaneously on different clients, then they are collaborating within this activity and should hence have a synchronized view on the activity. Within CSCW this is normally called desktop conferencing, collaborative applications, or application sharing. Central to real-time activity sharing is the concept of session management and how one can setup a range of collaboration 'widgets', like telepointers and audio/video connections.

Session Management via Sessions and Session Listeners

The runtime handling of a collaborative session with one or more active participants working together around a shared activity is handled by the Session class. This illustrated in figure 5. There is a one-to-one relationship between an *Activity* and a *Session*. Simply by activating an activity, the user joins the session. Hence, session management is implicit to the user avoiding the need for explicit session creation, naming, and browsing found in other application sharing systems (e.g. GroupKit; Roseman and Greenberg, 1995). This *activity-based session management policy* in the ABC framework is a light-weight session management policy (Edwards, 1994).

Processes can register as *Session Listeners* to a specific session, thereby receiving notifications about changes to this session. Session listeners receive events when a session changes, for example when a user joins or leaves a session, or when a session's associated activity changes. A central session listener is the *Session Manager* running on the different clients. The primary responsibility of the session manager is to maintain an up to date set of peer-to-peer communication channels to session managers running on the other clients participating in this real-time activity sharing. Another example of a session listener is the *Collaboration Frame* shown in figure 2. The frame is notified when participants enter or leave the session and it updates its user-interface accordingly.

In desktop conferencing situation there is a need for passing around the events of the users. For example, if a user scrolls down in a browser window, then all the other users should see the same. To accomplish this, the ABC framework utilizes the shared state mechanisms described above. The location of the scrollbar can be part of the shared state for the browser application and can hence be collected and shared among the participants via the State Manager and the Activity. However, this procedure is too coarse grained. Image a collaborative session with 5 clients, each running 10 applications, including the browser. If a user scrolls down, the State Manager would collect state information from all 10 applications and send it to the Activity, which subsequently would distribute this rather large chunk of data to the 5 participating clients. On each of these 5 clients, each of the 10 application would receive what appears to be new state information, and might start restoring this state. Therefore, a central part of the State Manager is to only distribute state objects that have changed.

As for handling late-comers to a collaborative session this is handled straightforward by the framework. When a client joins a session by activating its activity, the infrastructure simply transfers the current shared state of the activity to the new participant without disrupting the work of the existing members (see Shim et al., (1997) for a similar session synchronization strategy).

Because we are using the same state sharing mechanisms for 'desktop conferencing' an interesting issue arise in the ABC framework. Recalling that there is a decoupling between the activity's services and the concrete applications that might instantiate a service on a client, we now can have 'desktop conferencing' on two radically different types of 'desktops'. In scenario above, the X-ray images are shown using one image viewer on the wall-display, and another viewer on a PDA. Hence, we can have a real-time collaborative session with one user using a wall and another using a PDA. This clearly creates a new dimension to the WYSIWIS (What You See Is What I See) principle. This inconsistency is also the reason for terming our support for real-time collaboration for *activity sharing* rather than application sharing.

In this sense, we have left it in the hand of the application developers to decide how states are represented and handled, which moves a burden from the framework on the shoulders of the developers. However, as described above, the programmer's API contains several default state implementations that can help the developer to save and restore user-interface states easily. For example, default states for the UI look-and-feel of window frames, scrollbars, tables and lists already exists and others are constantly being created.

Peer-to-Peer Collaboration

As a listener to the collaborative session, each client's session manager maintains a complete list of participating host machines in a session, including details of network addresses and their configuration of communication port usage. This list is used to establish, maintain, and destroy direct peer-to-peer communication link to the other participating clients in the session. The peer-to-peer links are used to create 'collaboration widgets' that provides mutual awareness and information between the participating users. Two examples of such peer-to-peer collaboration widgets developed on top of this peer-to-peer communication mechanism in the ACB framework are *telepointers* and *direct voice links*.

Tele-Pointers

Gesturing, like pointing at and making circles around objects is essential to small co-located group interaction (Tang, 1991). Shared desktop conferencing systems often use telepointers (also known as multiple cursors) because the sharing of gestures on the 2-dimentional user-interface with the mouse cursor often seems sufficient (Hayne, Pendergast & Greenberg 1994, Roseman & Greenberg 1995).

There are several technical challenges associated with the implementation of telepointers. One of them is to synchronize the telepointers across the involved clients, including establishing, maintaining, and tearing down telepointes between active participants in a collaborative session, and how to communicate cursor coordinates among them. In the ABC framework the implementation of telepointers use the peer-to-peer communication link of the session manager to do this because we can trust it to have an always updated list of addresses on participating hosts. The actual sending and receiving of telepointer events are done peer-to-peer using UDP sockets. This is the most efficient and there is no need for the ABC server processes to be involved in telepointing. It should be noted that the client sending telepointer events is itself a receiver of its own telepointer events. This can be seen in figure 2 where the user name and host name of the user currently using this client is attached to the mouse pointer. This drawing of ones own telepointer can off course be disabled, but the technical mechanism lying underneath is essential in making recording of and activity, including the gestures made.

Other technical challenges lie in actually capturing and drawing telepointers. In the ABC framework we have separated these concerns making it possible for the application developers to apply appropriate telepointer handlers for capturing and drawing. The developer can write his own telepointer handlers or he can use some of the default ones provided with the framework. There are currently two default implementations of how to handle telepointers. The first one is the most obvious one, where the absolute coordinates of a cursor on the screen is collected, transmitted, and drawn at the same coordinates at the receiving client machines. The second one allows the individual ABC applications to enable or disable telepointers and records and draws the telepointers relative to the application. This is useful if the location of the application is not the same on all clients. In this way the telepointers on top of e.g. an X-ray image still reveals the correct gestures disregarding that the X-ray image may be located differently on the different clients.

Voice-Link

Similar to the telepointers, a sound link between participants in a collaborative session is essential. All studies of voice-conferencing, telephone conferencing, and video-conferencing establish that a voice link is essential. In some groupware toolkits and framework voice-links are not always considered important because it can 'easily' be establish through other existing technologies, like telephones or standard tele-conferencing system to desktop computers (e.g. Roseman & Greenberg 1995). However, we have made studies of users using NetMeeting as a way to establish how we should create voice-links and whether is should be integral to the ABC framework. Our conclusion way clearly that it should. There are many problems of establishing a tele-conference using telephones and/or NetMeeting and the largest problem was to establish the session. Everybody should do something explicitly to enter a session, like dialling some number (this applies for both the telephone meeting technology as well as for NetMeeting), and it was often difficult to establish who was active in the session. Cockburn & Greenberg (1993) has similarly made the observation that one of the obstacles to groupware use is the difficulty of initiating a groupware session.

Now, in the ABC framework these challenges were already solved via the Session concept described above. Just like in the case with the telepointers, a voice-link can be established among the active participants in a collaborative session. In this way voice-links are established when a users activates an activity and is removed similarly when the user de-activated the activity. Again, through our evaluation of the framework we have concluded that the activity concepts can encompass a traditional tele-conferencing situation. A user can create a view-less activity without any visible ABC applications running and invite other to participate. New users will be notified via the 'lamp' icon in the activity bar.

The voice links are technically implemented using peer-to-peer communication between the participants' machines, similar to the telepointer implementation. The recording, transmission, and play-back of the sound is implemented via the Java Media framework (JMF).

Related Work

The work of the ABC framework clearly related to the work in ubiquitous environments, as discussed in the introduction. Our concept of activity-based computing resembles the task-driven computing concept in Aura. But the ABC framework has a greater focus on *local mobility* within a work setting and not remote mobility as discussed in Aura. Furthermore, the ABC framework is inherently designed to support collaboration – asynchronous as well as asynchronous – both of which are absent in the Aura project.

Looking at related work in the research field of CSCW there is a range of technical research that relates to the implementation of the ABC framework. First, the real-time collaboration features of the ABC framework relate to the work on *application sharing systems*. Such systems can in general adopt either a centralized or a replicated architecture Different architectures have different tradeoffs, advantages, and application domain (Begole et al., 1999; Li and Li, 2002). In a centralized system, only one host runs the shared application and the input/output events are distributed among all participating clients. This has been a natural choice of architecture in the groupware systems on the X Windows platform because X Windows defines a network-aware graphics protocol that separates an application's display (the X server) from its computation (the X client). This separation yields a natural approach to implementing application sharing by having one centralized X client with multiple X servers on different host machines. Examples of systems are SharedX (Garfinkel et al., 1994) and XTV (Abdel-Wahab and Freit, 1991). But also on the Windows platform, NetMeeting applies a centralized application sharing architecture. In a replicated architecture the same application and its execution environment are replicated at each participating host machine. The ABC framework implements a replicated architecture where the client side processes and the ABC applications are executing on the local host machines also during real-time activity sharing.

The main advantage of the centralized architecture is its simplicity. However, one of the disadvantages of a centralized architecture for application sharing is the need for a floor control mechanism because the application (running in only one copy) is unable to handle simultaneous input. Another is the lack of supporting heterogeneous computing environments because only one application is running in a specific execution environment (see however Li and Li (2002) for an exception). We have chosen the replicated architecture because replicated applications sharing has faster local responses, lower network bandwidth requirements, and has more potential in supporting concurrent work eliminating the need for floor control. In our design of the collaborative sharing of activities we used NetMeeting as a prototype in a large workshop held with clinicians from the hospital and one of the conclusions from this workshop was to avoid floor-control mechanisms. Furthermore, a replicated architecture is better suited to handle the contingencies in a ubiquitous and heterogeneous computing environment. For example, if the network connection disappears the consequence is that the ABC infrastructure with it state and collaboration management fall out. However, the distributed applications running on the client hosts continue to execute.

The traditional problems of maintaining consistency among replicas of the same application and of accommodating late-comers in a replicated architecture (Lauwers and Lantz, 1990; Li and Li, 2002) have proven to be relatively easy to handle within the ABC framework. This is caused by the invariant of an activity object, which ensures that it always has the shared state of the collaborative session, and late-comers just obtain this when activating an activity.

A related characteristic of synchronous collaborative applications is to what degree the application itself is made collaborative by having access to the source code (Lauwers and Lantz, 1990). *Collaboration aware* application are specifically designed for simultaneous use by multiple users, whereas *collaboration transparent* application are shared by collaboration-aware mechanisms that are outside and unknown (i.e. the word 'transparent') to the application and its developers. Researchers following the collaboration transparent strategy have a very strong argument when they say that the assumption of creating collaboration aware version of already well-accepted single-user application (e.g. Microsoft Office) is highly dubious. From a user point of view, who would abandon their favourite word processor to use a co-authorship application (Grudin, 1994) and from a technical point of view, the assumption of having access to proprietary source-code for modification is in practice impossible for any real-world kind of applications. The current implementation of the ABC framework supports the development of collaboration aware applications. This decision was made because the ABC framework is a vehicle for investigating various research questions within ubiquitous computing and CSCW. However, we plan to integrate existing applications running in different environments by making ABC application wrappers to e.g. programs running on a Windows platform. We are currently investigation how to do this by hooking into the Windows API and making a general ABC wrapper to be used for all Window based applications. However, this approach is basically in conflict with our cross-platform, cross-environment strategy of decoupling services from applications, because we want to make the state management work on other platforms than Windows.

Finally, as for the key idea in the ABC framework of collecting services ('what users are doing') in an activity, there are similarities with other more user-interface related research. Starting with Rooms (Henderson and Card, 1986), and continuing through recent system, such as the Task Gallery for Windows (Robertson et al., 2000) and the Kimura system (MacIntyre et al., 2001), research have been conducted to design ways of handle multi-tasking in window-based user-interfaces. These 'virtual desktop' approaches treat tasks as a cohesive collection of applications. When a user refers to a particular task, the system automatically brings up all the applications and documents associated with that task. This relieves the users from launching and arranging applications and documents individually. In our work, we extend this notion by modelling an activity as a collection of abstract services decoupled from application that can handle such services. This decoupling paired with the distributed nature of activities allows an activity to be handed over to and instantiated in different environments using different supporting applications running of different hosts, and by different users who participate in the activity.

Discussion and Future Work

The ABC framework has been emerging during a two year period. It has been developed in close cooperation with clinicians (physicians and nurses) having their daily work in hospitals. We have conducted 11 design and evaluation workshops where clinicians for a whole day were asked to co-design, use, evaluate, and test the framework. A common method in our design workshops was to let the clinicians try out different design alternatives. For example, when trying to design real-time activity sharing, we spend a whole day experimenting with NetMeeting, because it is fundamentally different to our approach in architecture and hence in use.

We have created a number of our own ABC applications, especially focusing on medical applications for an electronic patient record – e.g. list of patients, medicine schemas, textual records, X-ray image viewers, etc (see figure 2). When analyzing the number of code lines needed to make such applications ABC-aware, this is only a very small fraction. Most of the code had to do with handing over and receiving state information (a similar observation is made in DistView (Prakash and Shim, 1994:161)).

The 'activity' as a concept has proved rather robust in our design. It has unified a range of usability-wise and technical concepts. For example, from a user's perspective, the movement of activities from one host to another seemed naturally, as well as the concept of collaborative activity sharing. As for the more technical side of it, the activity concept, as implemented in the ABC framework, helps maintain share state, which is managed and distributed by the framework; the activity concept encompass a light-weight strategy for session management using the activity's list of participants; and it makes a nice solution to collaboration awareness by providing a platform for establishing a wide range of peer-to-peer type of widgets, like the telepointer and the voice link described above.

As for future work, there are a number of items on our list. One of them is to address collaboration transparency by finding ways of incorporating legacy application in the framework and make them activity-aware. We are currently investigating strategies in this direction. Another item high on our list is to open up the framework for applications written in other programming languages than Java by creating an XML based activity protocol. A third item is to investigate in greater details the technical and usability aspect of real-time activity sharing across heterogeneous ubiquitous computing environments.

Conclusion

In this paper we have described a framework that addresses two of the hard problems in developing software systems for collaborative work in a ubiquitous computing environment. First, it attacks the problem of allowing a user to preserve continuity in his or her work when moving between different computing environments. The key advantage of this framework over more traditional approaches (e.g. thin clients or portable equipment like laptops or PDAs) is that it allows the system to adapt the user's task to the computational resources in the environment, making it possible to use several of these resources concurrently. Second, the framework at its core attacks the problem of handling collaboration, especially session creation and management. As users resume and pause activities, they implicitly engage in a collaborative session.

The key ingredients of the framework are a runtime infrastructure, which handles activity storage, management, porting, and collaboration. Application programmer can use the framework API to develop activity-aware applications as well to tailor the behaviour of the infrastructure.

The framework has been implemented in prototype form in Java, permitting activity handling for services written within the framework's API. While this implementation is only a first step, already it demonstrates that activity migration, distribution, and collaboration can be supported by the ABC framework. Several laboratory-based evaluation of the end-users view on the framework has been conducted and design recommendations have been incorporated in the framework. However, more extensive evaluation of the framework will be possible once we have created wrappers for legacy systems, like the electronic patient record and Microsoft Office, and we have conducted larger-scale experiments within a hospital for example.

Acknowledgement

The concepts of activity-based computing have been developed in cooperation with Henrik B. Christensen and Claus Bossen, as well as a team of highly engaged physicians and nurses from the hospital. The Danish Centre for IT Research (CIT) funded this research.

References

Abdel-Wahab, H. and Feit, M. (1991): 'XTV: A framework for sharing X window client in remote synchronous collaboration', Proceedings of IEEE Tricomm'91, Chapel Hill, NC, April 1991, pp. 159-167.

Bardram, J. E. (1997): 'Plans as Situated Action: An Activity Theory Approach to Workflow Systems', *Proceedings of the Fifth European Conference on Computer Supported Cooperative Work*. Kluwer Academic Publishers, 1997, pp. 17-32.

Bardram, J. E. (1998): 'Designing for the Dynamics of Cooperative Work Activities', *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, Seattle, Washington, USA, 1998. ACM Press, pp. 89-98.

Begole, J., Rosson, M. B., and Shaffer, C. A. (1999): 'Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems. ACM Transaction on Computer-Human Interaction, vol. 6, no. 2, June 1999, pp. 95-132

Bellotti, V. and Bly, S. (1996): 'Walking Away from the Desktop Computer: Distributed Collaboration and Mobility in a Product Design Team', *Proceedings of CSCW 1996*. ACM Press, 1996, pp. 209 – 218.

Bossen, C. (2002): 'The parameters of Common Information Spaces: the Heterogeneity of cooperative Work at a Hospital Ward', *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work CSCW2002*, New Orleans, Louisiana, USA, Nov. 2002. ACM, pp.

Christensen, H. B. and Bardram, J. E. (2002): 'Supporting Human Activities— Exploring Activity-Centered Computing': *Proceeding of Ubiquitous Computing 2002 (UBICOMP 2002),* Springer LNCS 2498, Berlin, 2002.

Cockburn, A. and Greenberg, S. (1993): 'Making Contact: Getting the Group Communicating with Groupware' *Proceedings of the ACM COOCS'93 Conference on Organizational Computing Systems*, Nov 1993, ACM Press.

Dertouzos, M. L. (1999): 'The future of computing', *Scientific American*, July, 1999.

Edwards, K. (1994): 'Session Management for Collaborative Applications', *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work CSCW1994*, Chapel Hill, North Carolina, USA, Oct. 1994. ACM, pp. 323-330.

Garfinkel, D., Welti, B., and Yip, T. (1994): 'HP SharedX: A tool for real-time collaboration', Hewlett-Packard Journal, vol. 45, no. 4, April 1994, pp. 23-36.

Garlan, D., Siewiorek, D. P., Smailagic, A., and Steenkiste, P. (2002): 'Project Aura: Toward Distraction-Free Pervasive Computing', *IEEE Pervasive Computing*, April-June 2002, pp. 22-31.

Greenberg, S. and Marwood, D.(1994): 'Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface', *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work CSCW1994*, Chapel Hill, North Carolina, USA, Oct. 1994. ACM, pp. 207-217.

Grudin, J. (1994): 'Eight challenges for groupware developers', *Communications of the ACM*, vol. 37, no. 1, pp. 92-105.

Hall, R. W., Mathur, A., Jahanian, F., Prakash, A. and Rasmussen, C. (1996): 'Corona: A Communication Serice for Scalable, Reliable Group Communication Systems', *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work CSCW1996*, Cambridge, Massachusetts, USA, Nov 1996. ACM, pp. 140-149.

Hayne, S., Pendergast, M. and Greenberg, S. (1994): 'Implementing Gesturing with Cursors in Group Support Systems'*, Journal of Management Information Systems*, vol. 10, no. 3, pp. 43-61.

Henderson, J. D. A. and Card, S. K. (1986): 'Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in Window-based Graphical User Interfaces', *ACM Transactions on Graphics*, vol. 5, no. 3, July 1986, pp. 211-241.

Lauwers, J. C. and Lantx, K. A. (1990): 'Collaboration awareness in support of collaboration transparency. Requirement for the next generation of shared window systems', *Proceeding of AMC CHI'90 Conference on Human Factors in Computing Systems*, pp. 303-311.

Li, D. and Li, R. (2002): 'Transparent Sharing and Interoperation of Heterogeneous Single-User Applications', *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work CSCW2002*, New Orleans, Louisiana, USA, Nov. 2002. ACM, pp. 246-255.

Luff, P. and Heath, C. (1998): 'Mobility in Collaboration', *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work CSCW1998*. Seattle, Washington, ACM Press, 1998, pp. 305-314.

MacIntyre, B., Mynatt, E. D., Vodia, S., Hansen, K. M., Tullio, J. and Corso, G. M. (2001): 'Support for Multitasking and Background Awareness Using Interactive Peripheral Displays', *Proceeding of ACM User Interface Software and Technology 2001 (UIST01)*, Nov. 11-14, Orlando, Florida.

Prakash, A. and Shim, H. S. (1994): 'DistView: Support for Building Efficient Collaborative Applications using Replicated Objects', *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work CSCW1994*, Chapel Hill, North Carolina, USA, Oct. 2002. ACM, pp. 153-164.

Robertson, G. et al. (2000): 'The Task Gallery: A 3D Window Manager', Proceedings of CHI2000, pp. 494-501.

Roseman, M. and Greenberg, S. (1995): 'Building Real Time Groupware with GroupKit, A Groupware Toolkit', ACM Transactions on Computer Human Interaction, vol. 3, no. 1, March 1995, pp. 66-106.

Shim, H. S., Hall, R. W., Prakash, A., and Jahanian, F. (1997): 'Providing Flexible Services for Managing Shared State in Collaborative Systems', Proceeding of the 5th European Conference on Computer Supported Cooperative Work (ECSCW97), Kluwer Academic Publishers, Dordrecht, 1997, pp. 237-252.

Sousa, J. P. and Garlan, D. (2002): 'Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments', *Proceeding of the 3rd Working IEEE/IFIP Conference on Software Architecture*, Montreal, 2002.

Tang, J. C. (1991): 'Findings from Observational studies of Collaborative Work', *International Journal of Man Machine Studies*, vol. 34, no. 2, pp. 143-160.

Weiser, M. (1991): ‘The Computer for the Twenty-First Century’, *Scientific American*, vol. 265, no. 3, Sept. 1991, pp. 94-100.