Support for Activity-Based Computing in a Personal Computing Operating System

Jakob E. Bardram, Jonathan Bunde-Pedersen, and Mads Soegaard
Centre for Pervasive Healthcare, University of Aarhus
Aabogade 34, 8200 Aarhus N, Denmark
{bardram,jbp,madss}@daimi.au.dk

ABSTRACT
Research has shown that computers are notoriously bad at supporting the management of parallel activities and interruptions, and that mobility increases the severity and scope of these problems. This paper presents *activity-based computing* (ABC) which supplements the prevalent data- and application-oriented computing paradigm with technologies for handling multiple, parallel and mobile work activities. We present the design and implementation of ABC support embedded in the Windows XP operating system. This includes replacing the Windows Taskbar with an Activity Bar, support for handling Windows applications, a zoomable user interface, and support for moving activities across different computers. We report an evaluation of this Windows XP ABC system which is based on a multi-method approach, where perceived ease-of-use and usefulness was evaluated together with rich interview material. This evaluation showed that users found the ABC XP extension easy to use and likely to be useful in their own work.

Author Keywords
Activity-Based Computing, ABC, Ubiquitous Computing, Task Management, User Evaluation

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces—*graphical user interfaces, windowing systems*

INTRODUCTION

In 1983 Bannon et al. argued that "current human-computer interfaces provide little support for the kinds of problems users encounter when attempting to accomplish several different tasks in a single session" [3, p. 54]. Today, 20 years later, these words seem valid still. Contemporary studies have shown that there is a significant mental and manual overhead associated with the handling of parallel work and interruptions [12, 20, 23, 15, 1]. Studies of non-office work, like clinical work in hospitals, similarly show how desktop computers do a poor job in supporting mobile users' parallel work activities, distributed in time and space [5, 6].

Generally speaking, contemporary computer technology is designed according to an application- and document-centered model. This model enables users to work with specific, targeted applications that support the manipulation of particular kinds of information and performing specific tasks, like writing a letter or making a budget. This model is deeply embedded in the hardware, operating systems and user interface software, as well as the development frameworks available today. It has proven well-suited for office work at a desktop, but the personal and task-oriented approach provides little support for the aggregation of resources and tools required in carrying out higher-level activities. It is left to the user to aggregate such resources and tools in meaningful bundles according to the activity at hand, and manual reconfiguration of this aggregation is often required when multi-tasking between parallel activities.

In our research, we have seen how these problems are highly exacerbated when moving out of the office and into a mobile and collaborative working environment like a hospital. Mobile and nomadic work amplify the reconfiguration overhead when users move from one work context to another, potentially using different computers and different types of devices. Thus, mobility introduces yet another obstacle for suspending and resuming activities, since a user's activities are 'tied' to his or her personal computer.

To meet these challenges, we are pursuing the concept of activity-based computing (ABC). In activity-based computing, the basic computational unit is no longer the file (e.g. a document) or the application (e.g. MS Word) but the *activity of a user*. The end-user is directly supported by computational activities which can be initiated, suspended, stored, and resumed on any computing device in the infrastructure at any point in time, handed over to other persons, or shared among several persons. Furthermore, the execution of activities is adapted to the usage context of the users, i.e. making activities context-aware. This paper reports our approach to activity-based computing with special focus on the user experience. It presents the design, implementation, and evaluation of a user interface software technology for activity-based computing. The implementation is done as an extension of Windows XP, and illustrates how the principles of activity-based computing can be incorporated into an existing operating system (OS).

CHI 2006, April 22-27, 2006, Montréal, Quebec, Canada.

RELATED WORK

The original Rooms system [2] was the first *virtual desktop management system* which allowed users to organize application windows in different 'rooms' associated with different tasks. A more recent version of this principle has been presented in the GroupBar system by Microsoft [25]. In addition to virtual desktop management systems, a number of research projects have proposed solutions for task management. These include support for moving groups of windows to the desktop periphery in the Scalable Fabric system [23], extending the user's desktop with additional screen space [7] or peripheral displays [19], employing 3D in the TaskGallery window management [24], using time as the main organizing principle in task management [22], or to allow for hierarchical window organization and elastic stretch and resize of windows in the Elastic Windows system [17].

Compared to this work on new user interface metaphors for task management, our work is distinct in at least three ways: Firstly, activities in ABC are *persistent* and *stateful* which means that state is preserved across service restart or computer shut-down. This is a fundamental difference, because it illustrates that activity-based computing is not designed to support grouping of application windows in convenient ways, but to support long lived human activities which unfold and evolve over time. Secondly, activities are *distributed* across networked computers and thereby support users to move their work activities with them while roaming between devices. The Gaia architecture for Smart Spaces [16] also supports applications to move across different Windows XP computers, but there is no support for task or activity management. Finally, we have explored design for activity-centered window management that do not replace the entire PC desktop with a new metaphor, but rather adhere to the same conceptual and physical window management as the underlying OS. In line with the conceptual idea of activity-based computing, the ABC extension to Windows XP 'merely' adds another level of user interface mechanisms on the activity level in terms of the activity bar, the Ctrl+Tab switching, and the activity sharing support. Hence, activity-based computing, and its supporting technology, seeks to extend – and not replace – the prevalent application and document view.

Other systems have been designed to provide more direct support for managing of multiple concurrent activities associated with large amounts of digital material. In *task-centered communication systems*, recording the history of the task, and the virtual workspace associated with it, is done by creating contexts out of message threads. Email communication threads are stored and reused in TaskMaster [8] and activity threads comprised of all accessed data objects are stored and maintained in the Activity Explorer prototype [26, 21]. These systems accurately reflect history of a project (or task) and the various data objects used in it; contacts, email, documents, etc. A common feature of such systems is however, that they are tied to the application making the history and do not support arbitrary use of the computer. The Activity Explorer, for example, supports 6 predefined types of objects: Message, chat, file, folder, screen shots, and to-do items. Hence, users need to "live" [21, p. 381] in the application and important information (state) and work tasks that do not go through the application are not supported. Our approach differs while we seek to enable activity-based computing support on the operating system level and not on the application level.

In *inference-based activity systems*, the user's interaction with the computer is monitored and recorded in order to make inference about the activity of the user. The UMEA system [18] records information about users' activities when interacting with the computer via monitoring the computer file system, input devices, and running applications. This enables semi-automatic maintenance of the content of project-related pools of documents, URLs, etc. A similar approach is used by the TaskTracer [14] which tries to infer so-called 'task profiles'. The goal of these systems is to help users access records of past activities and quickly restore the historical task context. Our approach to activity-based computing does not support activity inference by monitoring user-interaction. For our purpose the benefit of having semi-automatic maintenance of activity content does not merit the cost of monitoring and inferring. We find that the small overhead of creating, naming an activity, and attaching services to it, does not warrant the use of inference techniques. Instead we are using a context-aware infrastructure to help recognize relevant activities based on the current usage context of the user [9].

ACTIVITY-BASED COMPUTING

Our approach to activity-based computing is rooted in a year-long engagement in the study of, and design for, hospital work with focus on the challenges of handling parallel activities, interruptions, mobility, sharing, collaboration, coordination, and easy access to large amount of physical and digital information [10, 5, 6]. In the analysis of the use of contemporary computer technology in hospital work we have seen a range of critical short-comings which pose fundamental challenges to the design of future computer technology.

Firstly, computers are *application- and data-centered* and provide little support for aggregating sets of related applications or services as well as negligible support for *interruptions* in work. Often users need to manually reconfigure their applications to match new or recurrent tasks. Secondly, computers are designed for *stationary* use at a desktop. This also includes so-called 'mobile devices', like laptops, which are difficult to use without sitting down at a desk. Thirdly, applications run *isolated on homogeneous devices*. Hence, it is very difficult to move a set of applications or services from one computer to another, and even more difficult to move it between different kinds of devices. Fourthly, the 'Personal Computer' with its operating system is made for *single-user tasks*. However, a core aspect of everyday activities – especially in a hospital – is their collaborative nature. A fundamental challenge is therefore to investigate how support for collaboration can be made part of the computing infrastructure. Finally, computers are inherently *insensitive* to the working context of its users. Hence, there is no way in which a computer can take contextual information into consideration in the interaction between human and computer.

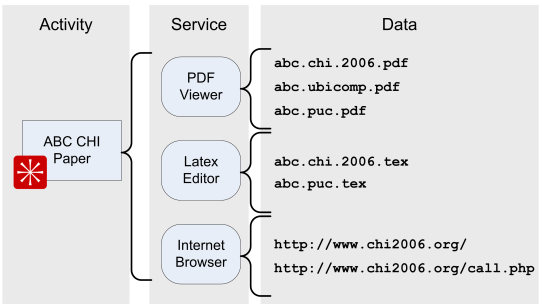

Figure 1. The 'ABC CHI Paper' activity used as an example throughout the paper.

However, many studies have shown that these challenges also exist in other kinds of work, including so-called 'information work' taking place in an office environment. Hence, even though our initial research was rooted in a hospital environment, we are currently working on a more general level and is proposing activity-based computing (ABC) as an approach to computing, which focuses on computational support for mobile, collaborative, and distributed activities which are adapted to their usage context. We are arguing that support for whole activities, rather than individual tasks, is important in pervasive environments. Figure 1 is a conceptual illustration of an *activity* which is a work-related aggregation of *services* and *data*. We have defined activity-based computing around the following essential principles:

Activity-Centered – A 'Computational Activity' collects in a coherent set a range of services and data needed to support a user carrying out some kind of (work) activity. This principle addresses the challenge of application-centered computing.

Activity Suspend and Resume – A user participates in several activities and he or she can alternate between these by suspending one activity and resuming another. Resuming an activity will bring forth all the services and data which are part of the user's activity. This principle addresses the lack of support for interruptions.

Activity Roaming – An activity is stored in an infrastructure (e.g. a server) and can be distributed across a network. Hence, an activity can be suspended on one workstation and resumed on another in a different place. This principle addresses the challenge of mobility.

Activity Adaptation – An activity adapts to the resources available on the device (i.e. computer) on which it is resumed. Such resources are e.g. the network bandwidth, CPU, or display on a given devices. This principle addresses the challenge of isolated and homogeneous devices.

Activity Sharing – An activity is shared among collaborating users. It has a list of participants who can access and manipulate the activity. Consequently, all participants of an activity can resume it and continue the work of another user. Furthermore, if two or more users resume the same activity at the same time on different devices, they will be notified and if their devices support it, they will engage in an on-line, real-time desktop conference. This principle addresses the challenge of collaboration.

Context-awareness – An activity is context-aware, i.e. it is able to adapt and adjust itself according to its usage context. Context-awareness can be used for adapting the user interface according to the user's current work situation – or it can be used in a more technical sense, where the execution of an activity, and its discovery of services, is adjusted to the resources available in its proximity. This principle addresses the challenge of context insensitivity.

The focus of this paper is to show how activity-based computing has been incorporated in the Windows XP operating system. Hence, the paper will focus specifically on the single user aspects, i.e. support for handling activities, activity suspend/resume, activity roaming, and activity adaptation. Collaborative activity sharing and context awareness has been discussed elsewhere [5, 9].

ABC FOR WINDOWS XP

This section presents the user interface and implementation of the activity-based computing extension to the Windows XP operating system. This ABC user interface is part of the client layer in an overall ABC architecture, where the underlying infrastructure layer is responsible for activity distribution and concurrency control in activity-based collaboration. This paper, however, exclusively focuses on the client layer and its implementation as part of Windows XP.

The ABC user interface for Windows XP is shown is figure 2. In Windows XP, each service is mapped to an application window. Thus, an activity can be made up of a range of windows, including child windows to an application, where the main window is not part of the activity. In figure 2 the activity labeled 'ABC CHI Paper' is resumed and contains windows from different applications like Adobe Reader, Firefox, and an open mail in a child window from Thunderbird. Let us consider the different parts of the ABC user interface for XP in more details, including some of the implementation details.

Activity Bar

The main user interface component is the Activity Bar illustrated in figure 2. This bar replaces the Windows XP Taskbar since activities – and not applications – are the main focus in ABC. In order to facilitate an intuitive understanding of how the bar works, the activity bar is deliberately designed to resemble the Windows Taskbar. The 'Activities' button is used to list the current user's activities as shown in figure 4. The action buttons are used to: (i) Create a new activity; (ii) suspend the current activity; (iii) invite other participants; (iv) save the activity locally; (v) zoom out the activity; (vi) show the ABC control panel; and (vii) to show the radar view. Frequently used activities are shown in the middle part of the bar, and the status icons on the left reveal the collaborative status for the current user: (i) Other online participants; (ii) tele-pointers on/off; (iii) voice-link on/off; (iv) and server online status.

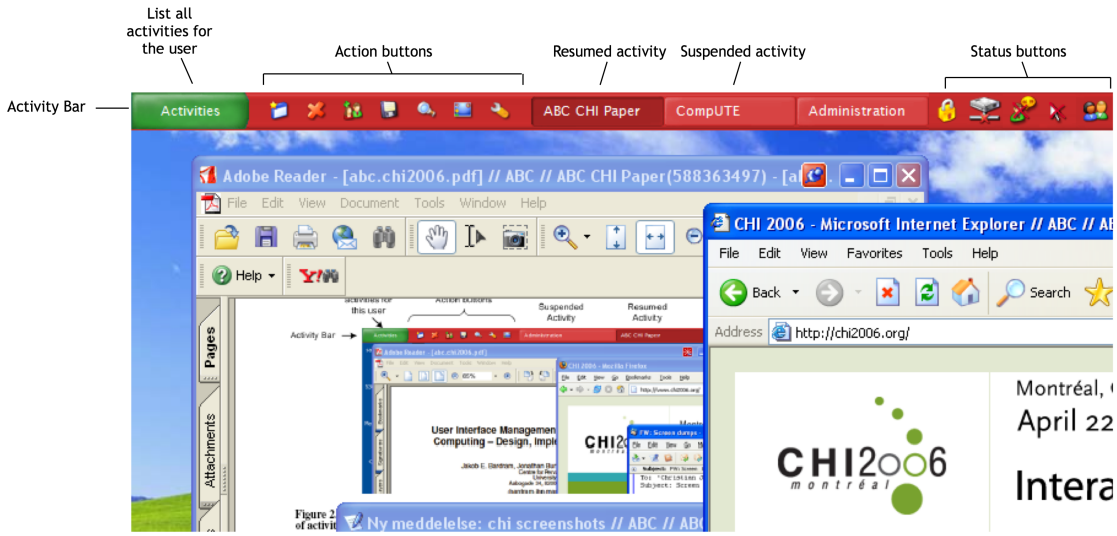

Figure 2. An overall view of the ABC user interface for Windows XP, including the Activity Bar, the current user's list of activities, and different application windows that are part of the resumed activity labeled 'ABC CHI Paper'.

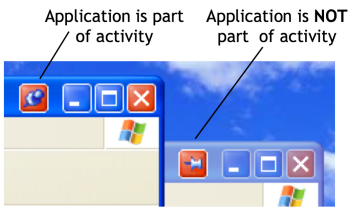

Figure 3. The 'activity icon' for pinning and unpinning an application to an activity.

Application windows are added and removed as services in an activity by using the 'activity icon' in the windows title bar as illustrated in figure 3.

Activity Suspend and Resume

In the ABC user interface, activities can be suspended and resumed in several ways. The typical way is to suspend the current activity and resume another by selecting a new activity in the activity bar or in the activity list illustrated in figure 4. The red cross action button in the activity bar suspends the current activity without resuming another one. This is useful in getting rid of an activity when creating a new one. For easy activity alternation, we have extended Windows XP with a 'Ctrl+Tab' switcher analogue to the Windows 'Alt+Tab' switcher. By pressing 'Ctrl+Tab' a user can quickly switch between his activities. The Windows 'Alt+Tab' switcher is still working and is used to switch between application windows within an activity.

Technically, all applications can be kept running when the activity is suspended. This is done for two reasons: (i) to enable *concurrent activities* by having services in a suspended activity to run in the background while working on another

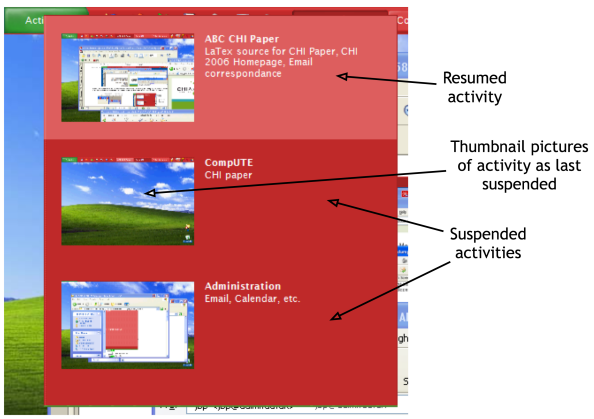

Figure 4. The list of activities for a user displayed when pressing the 'Activities' button.

activity; and (ii) to allow for much faster switching between activities on a local machine, saving the time it takes applications to start up every time an activity is resumed locally. However, applications which belong to suspended activities are removed from the Windows 'Alt+Tab' switcher. Hence, the user is not confused by applications not belonging to the currently resumed activity. When activities are resumed and suspended, activity state information is synchronized with the underlying ABC infrastructure which is responsible for activity roaming and sharing.

Activity UI Adaptation

The ABC user interface for Windows XP is also a zoomable window manager which enables the user to zoom in and out on activities as illustrated in figure 5. This zoom functionality is used to adapt the user interface to different display sizes and thus supports the principle of activity roaming in

the underlying ABC architecture. When an activity is resumed on various Windows XP devices with different screen resolutions, the ABC window manager for XP can zoom the activity to fit the current screen. In figure 5 the different application windows used in the 'ABC CHI Paper' activity is distributed in space and the activity is zoomed out. The shortcut 'Ctrl+1' is used to toggle between zoom in/out and enables the user to quickly get an overview of an activity and the zoom in on details by clicking on the window. Furthermore, in order to navigate within an activity, there is a radar view as illustrated in figure 6. This radar resides transparently on top of the application windows and a red square indicates the current viewport. By dragging the red square in the radar view, the current viewport on the Windows XP desktop is adjusted.

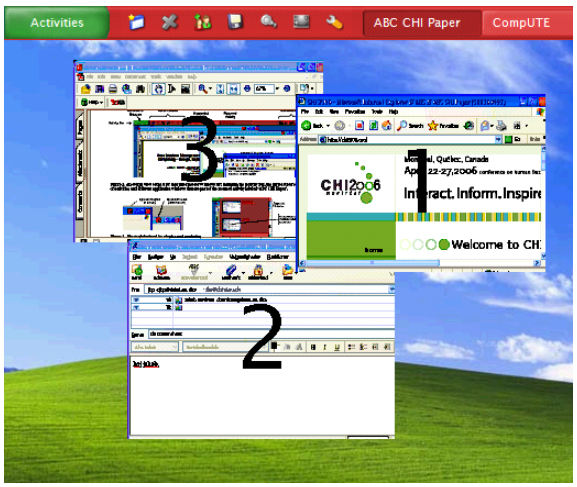

Figure 5. The activity 'ABC CHI Paper' zoomed out. The numbers rendered on top of applications are used to by a voice interface, allowing the user to zoom in on e.g. service no. 1.

The zoom functionality is primarily designed to enable the ABC user interface to support activity adaptation to different devices. Zooming however, also proved useful in single-device activity handling as it helps users manage activities with many windows in a spatial 2D metaphor that conserve the arrangement of windows . In contrast to other zoomable user interfaces, we maintain the window size and position, and do not rearrange windows (like Exposé in Mac OS) or introduce new layout metaphors using e.g. 3D [24].

Activity Roaming

Activity roaming is done via the underlying ABC infrastructure which distributes, manages, and stores activities and their state information. A local activity cache is used if the client is not online (the online status is revealed by the small icon on the right-hand side of the activity bar in figure 2). When an activity is resumed, the description and the state of the activity is fetched from this distributed infrastructure. On each device, the ABC XP extension runs a small registry which maps services to local applications. For example, the service named `internet_browser` may be mapped to Mozilla Firefox on one device and to MS Internet Explorer on another. When an activity is resumed on a device, the applications corresponding to the services in the activity are started and their state is restored, including references to relevant data. Similarly, when the user suspends an activity, state information from each application is collected and stored in the activity description, which is then handed over to the ABC infrastructure.

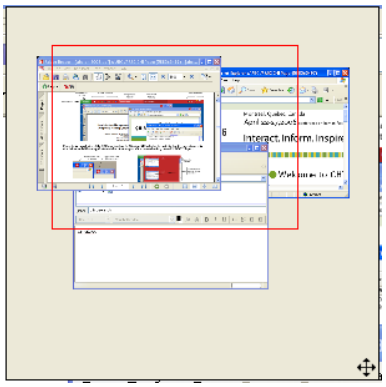

Figure 6. The radar view of an activity.

Clearly, activity roaming relies on corresponding distribution mechanisms for the data involved in an activity. It is beyond the scope of this paper to discuss this in great detail, but solutions to data distribution exist. Network file systems, mobile file systems, peer-to-peer file systems, or WebDAV may be used to distribute files, and in a hospital environment, most systems rely on a client-server architecture which handles distribution of the data layer. Furthermore, the ABC infrastructure contains mechanisms for attaching data to an activity, which may be exploited by an application.

Implementation

The software architecture of the Windows XP extension for activity-based computing is illustrated in figure 7. The 'Activity Controller' is the main component. It provides handles for resuming and suspending an activity, manages connections to the activity infrastructure and to other clients, and has two direct hooks into the OS. Basically it is the point of contact between the infrastructure and the client, as well as between the client and any user-interfaces which may be built on top of it. Hooks are handles used by the client layer to interface with binary components of the OS. The 'desktop hook' interfaces with the windowing system and grabs the thumbnail pictures of an activity just before suspension (see figure 4). The 'keyhook' listens for the ABC key-combinations, which are 'Ctrl+Tab' for activity switching and 'Ctrl+1' for zooming. Other external components can listen for arbitrary key combinations using this hook.

The 'Service Registry' has access to the tables which map installed applications to service descriptions in the activity. This allows for adaptation of an activity to locally available applications and for users to define which applications they prefer for which services. The 'Window Monitor' is responsible for managing running applications in the OS. It pub-

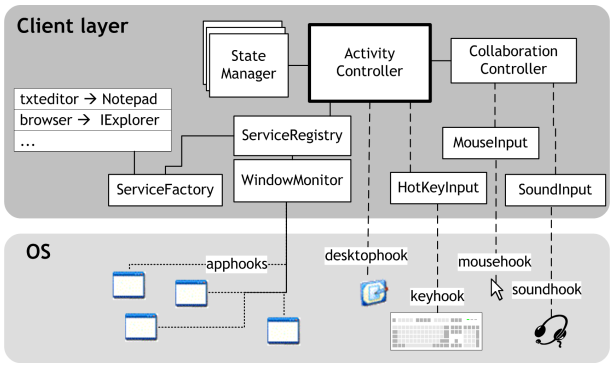

Figure 7. The software architecture of the Windows XP extension for activity-based computing.

lishes events whenever an application window is opened or closed. This is done by polling the OS at intervals for all visible windows. Events are then sent to the registry, which registers changes to the window. The monitor also installs the 'Activity Icon' in the title bar of the windows (see figure 3). This icon is used for pinning (adding) and unpinning (removing) applications to activities. Whenever an application is added to an activity (by pressing the activity icon), the registry asks the 'Service Factory' for a service which is capable of wrapping the specific type of window. This service is then added to the current activity through the activity controller. Once a service has been added to an activity the state of it is continuously monitored. This happens in the 'State Manager', which tracks changes in the state of all local services. If the activity is shared then a change in its state is immediately sent to all participants via the 'Collaboration Controller'. In this manner, when sharing an activity, the state of all services in the activity is synchronized on all participating devices – giving each participant in the activity an identical view of all applications. The 'soundhook' and 'mousehook' allows the collaboration controller to establish voice links and tele-pointers to collaborating peers.

State Management

Activity suspending/resuming and activity roaming rely on the ability to access state information for running applications. We call such applications *stateful* [5]. In the present Windows XP implementation, we differentiate between three types of stateful applications: (i) legacy applications with no programming API, (ii) legacy application with an API, and (iii) applications with full access to the source code.

In the first type of applications, only the application's name, window size and position, and executable is stored as state information – this is the information that we can extract from the OS. This means that when such an application is moved from one device to another, the application window is restored but not the content. For the second type of applications, special-purpose ABC application wrappers are created. MS Internet Explorer, for example, provides a COM interface which is used to get access to the currently displayed URL and the scroll location on the page. Hence, application window and content can be restored when an activity roam. We have created application wrappers for MS Internet Explorer, MS Powerpoint, MS Word, the Eclipse IDE, and Notepad. These wrappers are quite simple and require little amount of code.

```
[StatefulField(current_url)]
public URL CurrentURL;

[StatefulField(current_position)]
public int CurrentPosition;
```

Figure 8. State annotations in a browser application. Just by annotating the URL and the page position, the browser becomes stateful and can participate in activity roaming.

The `StatefulApplication` interface in the ABC programming model is used to develop stateful applications. To help implement state management in the application, fields that represent 'state' information can be annotated as stateful. Figure 8 shows an example of state annotations for an internet browser which ensures that the URL and the scroll position is saved as state information for this service. Stateful applications and their annotated fields are managed automatically by the programming model and adds little extra work to application programming. Our argument is that requiring the programmer to denote stateful fields in his or her program is a small overhead and is negligible compared to other requirements which an operating systems imposes on applications and application-programmers[1] .

EVALUATION

Earlier versions of the activity-based computing platform have been evaluated extensively by clinical personnel from different hospitals [4], which has provided good evidence that the principles and suggested technologies are useful in a hospital setting. The goal of the work presented in this paper is, however, to investigate whether support for activity-based computing can be part of a contemporary OS, like XP. This raise the following usability questions:

- **Usefulness** – Our primary concern was to establish whether the whole idea of activity-based computing as embedded in the Windows XP operating system (OS) is useful, i.e. can ABC be part of a contemporary OS? Can activities exist in an OS or does it clash with the desktop metaphor? Is ABC useful in other, non-clinical, domains?
- **Ease-of-Use** – Our secondary concern was to establish if the ABC user interface for XP was easy to use. Our goal was that experienced Windows XP users would be able to use the ABC extended version with no prior training.

To answer these questions, we wanted experienced XP users to use the platform because they can give much more accurate test results that precisely address the stated research questions. Our previous evaluations have shown that clinicians are far from experienced computer users and many of

[1]The current programming model is based on the .net platform and the example in figure 8 is C#.

them had serious difficulties understanding and using basic functionality in XP. Furthermore, to assess if activity-based computing is useful outside a hospital, we need users and scenarios from a non-hospital setting.

Usefulness was more specifically evaluated by investigating how users were able to use the activity concept to aggregate services and data when handling different tasks, large amounts of digital material, parallel work, interruptions, migrating work across different devices when moving around, and adapting the user interface of an activity to the available screen real-estate.

Methods

We have refrained from measuring task completion time. It is trivial to see that users of an ABC platform like the one presented in this paper would out-perform standard Windows XP users if asked to perform tasks that require handling parallel tasks, accessing lots of digital material, coping with frequent interruptions, and moving across heterogeneous devices.

The goal was to provide objective measurements on the usefulness and usability of our design while, at the same time, investigating the underlying detailed user reaction to the ABC user interface in a more qualitative fashion. For this purpose we have devised and used a *multi-method evaluation setup* where we (i) ask users to perform a range of tasks while thinking aloud, (ii) do analysis by scenario [11], (iii) investigate perceived usefulness and usability based on a questionnaire [13], and (iv) make a semi-structured interview.

Experimental Setup

To recruit experienced Windows XP users, we asked 16 graduate computer science students to participate in a semi-controlled evaluation. They were tested separately, each test lasting just over 1 hour. All participants were skilled programmers. Mean age was 27 years.

The test was set up to simulate well-known tasks for the participants, in this case a programming assignment for a company. The experiment was run in 3 rooms by 2 experimenters and a 'scenario judge', responsible for the analysis by scenario. Room 1 was the participant's 'office', room 2 was the office of his colleague (played by experimenter no. 2), and room 3 was the 'board room' of the fictional company at which the participant worked. Experimenter no. 1 and the scenario judge followed the participant around as he changed locations. The participant had a desktop PC, a laptop, and a telephone in his office. The board room had a computer with a wall-sized display. The 3 computers all had varying screen resolutions and displays, the ABC XP extension, the Eclipse IDE, MS Word, MS PowerPoint, Internet Explorer, plus five custom-built medical applications.

Tasks

Six overall scenarios were the backbone of the test: (i) Code; prepare a presentation; perform private activities; be ready for support calls. (ii) Take care of personal matters. (iii) Answer a support call from a client wanting support for medical applications. (iv) Consult a colleague in a nearby location to get help for the code. (v) Take care of personal matters. (vi) Present your work to the Board Members.

Experimenter no. 1 conducted the test according to a script which both guided the participants through the overall scenarios and their subtasks, and at the same time introduced interruptions and asked the participants to start working on parallel tasks. Examples of interruptions are different kinds of phone calls introducing new tasks and people entering the office. Examples of parallel tasks are asking the user to do some brainstorming in MS Word.

Procedure

After signing the informed consent form, the participant was given a quick introduction to the ABC user interface and the think-aloud method. He was then given a description of the tasks he was to perform. While performing these tasks, the scenario judge observed the participant and decided, based on a breakdown factor and on a 5-point scale, how much of the ABC Framework's potential the participant used for each scenario. This yielded an 'analysis by scenario' score [11].

A questionnaire was administered to the participants after the experiment. It consisted of 39 questions covering perceived ease of use and perceived usefulness. The latter were further sub-divided into the factors listed above (see also table 1). Each factor was covered by a number of redundant questions, the order of which was randomized, and framed in both positive and negative ways. The questionnaire is an extension of the 'perceived usefulness/ease-of-use' questionnaire [13]. Participants were asked to make self-predictions about their likely future use of the ABC Framework. This, we believe, will increase the ecological validity since the user to a lesser degree is evaluating the ABC Framework with regard to specific laboratory tasks but evaluating its potential role in their own work. As argued by Davis and others, these kinds of self-predictions "[...] are among the most accurate predictors available for an individual's future behavior" [13, p. 331]. Participants were asked to make self-predictions based on questions like: "Using ABC in my daily work would enable me to handle more information than today." Answers were given on a seven-point scale with 'likely' and 'unlikely' as end-point adjectives.

After the questionnaire, a semi-structured qualitative interview was conducted to get evaluation feedback in the participants' own words. It featured questions such as 'In your own words, what is the best and the worst thing about ABC?' and was designed to address everything from the participant's sentiments about the test to any suggestions for improvement they might have. The whole test was video-taped, including the follow-up interviews, and subsequently analyzed and partly transcribed.

Results

The results from the questionnaire are shown in table 1. Low scores indicate 'likeliness', e.g. a score on 1 is 'extremely likely', a score on 2 is 'quite likely', a score on 3 is 'slightly likely', and a score on 4 is 'neither likely or unlikely'. Scores

	Avg.	*Std. dev.*
Usability	2.09	0.89
Usefulness	3.07	0.96
Activity aggregation	2.67	1.07
Large amount of data	3.36	1.37
Parallel work	2.97	1.15
Interruptions	3.16	1.27
Roaming	2.31	1.07
Het. Devices	3.11	1.73
Zoom feature	4.07	1.70

Table 1. Results from the questionnaire.

#	*Scenario*	*Avg.*	*Std. dev.*
1	Code	4.33	1.35
2	Personal tasks	4.33	1.29
3	Support call	4.27	1.49
4	Colleague	4.87	0.52
5	Personal tasks	4.87	0.35
6	Presentation	4.53	0.52

Table 2. Results from the analysis by scenario.

above 4 indicate 'unlikely'. Table 1 shows that on average the participants found it quite likely that ABC for Windows XP would be easy to use (2.13, std. dev. of 1.05). The ease-of-use questions included questions on learnability, understandability, and flexibility. Furthermore, the participants found that on average it would be slightly likely that ABC would be useful to them (3.06, std. dev. of 0.92). Looking more specifically into the underlying factors, the mechanisms for activity aggregation and activity roaming were perceived quite likely to be useful. In general, it is interesting to note that on average none of the evaluated factors in table 1 were perceived unlikely to be useful (i.e. all scores are 4 or below). The standard deviation for all factors is around 1, which indicates that the answers were rather consistent.

The results from the analysis by scenario by the scenario judge is shown in table 2. In this measurement, high figures (5) indicate that the experimenter judged the participant to make extensive use of the features of ABC. Table 2 shows that on average participants had a high score in the use of ABC and its features. These figures indicate that participants quickly learned how to use and utilize the features of ABC, supported by the results from the ease-of-use questionnaire. Note, however, that the figures in table 2 also reveal that users learn as they use ABC – the score for the same scenario ('Personal tasks') is higher the second time.

DISCUSSION

Going back to the original work by Bannon et al. [3] we feel that the current ABC extension to Windows XP has come a long way in creating computational support for activity management. As our evaluation shows, users found it likely that ABC would help manage parallel tasks and interruptions, "reduce mental load when switching tasks" and help users "suspend and resume activities" [3, p. 54].

Our approach has been to embed activity support into the operating system because we intent to extend, rather than replace, the way users use files and applications today. Based on their studies of task switching and interruptions in Windows XP, Czerwinski et al. [12] reaches a similar conclusion: "It is clear that more can be done within the operating system and software applications to help users multitask and recover from task interruptions, hence potentially increasing productivity." [12, p. 181]. The need for activity support on the operating system level was also pointed out by one of the users during the follow-up interview:

> *I think this is smart – to associate things to activities. I've often thought about having 'association tags' on files, thereby being able to associate them to different things instead of them just being a file. [...] It's appealing to think of this as a greater framework for using your computer where files just have disappeared. [...] I think all of this is nicely done, and I'm certain that if you guys don't do it, then others will do it, because it is completely obvious to think this way...*

This user finds the idea of having activities as an organizing concepts around files and applications 'smart' and this way of organizing the operating system for 'completely obvious'. Another user similarly argued that the simplicity of the approach was a benefit, i.e. that the Windows XP ABC mechanisms introduce a minimum of additional 'things' to the operating system.

> *It is extremely simple – that's also a good thing. It doesn't try to be advanced in any way. It can do a few things and it does it very nicely – that's the best thing about it.*

Going back to table 1 the support for 'activity roaming' was perceived useful, which were also backed up by the interviews:

> *The best thing is the ability to move your 'state' from one computer to another. The whole idea of making it persistent... It's extremely nice to be able to close your computer and then it comes back up in the same state. That's the reason why I newer turn off my Linux machine – it just runs for weeks as does all of the applications on it.*

The original motivation behind activity roaming was to loosen the tight one-to-one binding between a user and a (personal) computer. Hence, by supporting activity roaming users would be free to use different computers and to move around more easily. This hypothesis was also confirmed in our experiment, as illustrated by the following quote:

> *[I]t is a sensible way to be independent of precisely this particular computer. That you have the opportunity to, well ... close the book and then open it again another place and you get the same back.*

The interviews also contained questions regarding areas for improvement to the current system and its user-interface. Users had some more general comments on the user-interface

and suggestions for improvement. They found that the zoom function was necessary in the adaptation on different display sizes, but they found that it could be greatly improved and work as smooth as the Exposé function in Mac OS X. Currently, the ABC zoom functionality is simply too slow to be useful. One user also had suggestions for creating automatic adaptation to the size of the screen. Furthermore, there were comments on providing better feedback to the user on how a service and a piece of data (e.g. a file) were related to an activity. We regard these issues as important, but they are more related to further development of the user-interface for Windows XP, rather than more fundamental research issues.

Users raised, however, also more fundamental issues. First of all, the apparent lack of support for having the same service (window) in more than one activity. As argued by Bannon et al. [3] you need "multiple perspectives on the work environment" and need to support 'multiple windows' as done in the Rooms system [2]. The current implementation of the ABC extension to Windows XP actually supports this technically – the same window can be part of more than one activities and its state may be different in the different activities, as also pointed out in the Rooms paper to be important. The users just cannot do it because there is no user-interface handles for doing it; we simply have not been able to come up with a good design for doing this in a simple manner.

The other more basic issue raised by some users were concerned with the life cycle of an activity, i.e. when does an activity emerge, when do you create one in the user-interface, how does it end, and how is it related to other activities. As argued by one of the users:

> *The worst thing? Well [...] if you have to put everything into activities, then you need to constantly consider 'where does this one belong'. In many situations something just appears quickly and then you start up some application and do some things in it. And if you don't get it categorized into some activity, then it may disappear? Or you may forget which activity it went into. Then you may need to search for it.*

This user addresses the basic notion of how an activity emerges, or more specifically how it often may overlap or intertwine with another activity. In a previous version of the ABC user interface, application windows which were opened were automatically attached to the currently resumed activity. In our own use of the system we however discovered that this was inconvenient – often your windows and files are opened 'inside' an activity, but they do not have anything to do with this activity. They may, for example, result from an interruption. As the user interface works now, the user is able to launch new applications and files and then suspend the currently resumed activity. This will remove application windows related to the activity, and the user is left with only those new windows, which have been opened due to the interruption. These windows may then form the basis for a new activity, or may just be used without any activity support.

CONCLUSION

In this paper we have presented an approach for embedding support for activity-based computing in the Windows XP operating system, with special emphasis on the single user experience. The core principles for activity-based computing was presented, which pivots around the support for aggregating services and data in coherent sets called 'activities', support for activity suspend and resume, support for activity roaming between different computers, and the support for activity adaptation to different display sizes.

We presented the design and implementation of the ABC extension to XP. The core user-interface components are the Activity Bar, which replaces the Windows Taskbar, the Activity Icon which integrates activity-support to native Windows application windows, the Activity List showing a user's activities, and the Activity Zoom which support spatial 2D layout of the windows within a service.

Finally, the ABC extension to XP was evaluated using a multi-method evaluation setup. This evaluation revealed that users found the user interface easy to use and that activity-based computing support would be useful for them in their work. Due to the multi-method approach, we were also able to establish more precisely what part of the user interface that needed to be improved. These results are being used in our current enhancement of the system, which also targets the support for roaming files together with activities. Hence, files belonging to an activity would move between computers as the user roams.

Acknowledgments

The ABC project is funded by the Danish Research Council under the NABIIT program. Henrik B. Christensen has been much involved in the early work on ABC and Martin Mogensen has helped with the implementation.

REFERENCES

1. P. D. Adamczyk and B. P. Bailey. If not now, when?: the effects of interruption at different moments within task execution. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 271–278. ACM Press, 2004.
2. J. Austin Henderson and S. Card. Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics (TOG)*, 5(3):211–243, 1986.
3. L. Bannon, A. Cypher, S. Greenspan, and M. L. Monty. Evaluation and analysis of users' activity organization. In *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 54–57. ACM Press, 1983.
4. J. E. Bardram. Activity-Based Computing – Principles, Implementation, and Evaluation. Manuscript submitted to the 'ACM Transactions on Computer-Human Interaction (ToCHI)'. Submission date: April 2004.
5. J. E. Bardram. Activity-Based Computing: Support for Mobility and Collaboration in Ubiquitous Computing.

Personal and Ubiquitous Computing, pages 312–322, July 2005.

6. J. E. Bardram and C. Bossen. Mobility Work – The Spatial Dimension of Collaboration at a Hospital. *Computer Supported Cooperative Work.*, 14(2):131–160, 2005.
7. P. Baudisch, N. Good, and P. Stewart. Focus plus context screens: combining display technology with visualization techniques. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 31–40. ACM Press, 2001.
8. V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 345–352. ACM Press, 2003.
9. H. B. Christensen. Using Logic Programming to Detect Activities in Pervasive Healthcare. In *International Conference on Logic Programming, ICLP 2002*. Springer Verlag, 2002.
10. H. B. Christensen and J. E. Bardram. Supporting Human Activities – Exploring Activity-Centered Computing. In *Proceedings of Ubicomp 2002*, pages 107–116. Springer Verlag, 2002.
11. G. Convertino, D. C. Neale, L. Hobby, J. M. Carroll, and M. B. Rosson. A laboratory method for studying activity awareness. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 313–322. ACM Press, 2004.
12. M. Czerwinski, E. Horvitz, and S. Wilhite. A diary study of task switching and interruptions. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182. ACM Press, 2004.
13. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3):318–340, 1989.
14. A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 75–82. ACM Press, 2005.
15. V. M. Gonzalez and G. Mark. "constant, constant, multi-tasking craziness": managing multiple working spheres. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 113–120. ACM Press, 2004.
16. C. Hess, M. Román, and R. Campbell. Building Applications for Ubiquitous Computing Environments. In *International Conference on Pervasive Computing (Pervasive 2002)*, pages 16–29. Springer-Verlag, 2002.
17. E. Kandogan and B. Shneiderman. Elastic windows: evaluation of multi-window operations. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 250–257. ACM Press, 1997.
18. V. Kaptelinin. Umea: translating interaction histories into project contexts. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 353–360. ACM Press, 2003.
19. B. MacIntyre, E. D. Mynatt, S. Vodia, K. M. Hansen, J. Tullio, and G. M. Corso. Support for Multitasking and Background Awareness Using Interactive Peripheral Displays. In *Proceeding of ACM User Interface Software and Technology 2001 (UIST01)*, pages 11–14, Orlando, Florida, USA, Nov. 2001.
20. G. Mark, V. M. Gonzalez, and J. Harris. No task left behind?: examining the nature of fragmented work. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–330. ACM Press, 2005.
21. M. J. Muller, W. Geyer, B. Brownholtz, E. Wilcox, and D. R. Millen. One-hundred days in an activity-centric collaboration environment based on shared objects. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 375–382. ACM Press, 2004.
22. J. Rekimoto. Time-machine computing: a time-centric approach for the information environment. In *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 45–54. ACM Press, 1999.
23. G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. R. Hutchings, B. Meyers, D. Robbins, and G. Smith. Scalable fabric: flexible task management. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 85–89. ACM Press, 2004.
24. G. Robertson, M. van Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel, and V. Gorokhovsky. The task gallery: a 3d window manager. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 494–501. ACM Press, 2000.
25. G. Smith, P. Baudisch, G. G. Robertson, M. Czerwinski, B. Meyers, D. Robbins, and D. Andrews. Groupbar: The taskbar evolved. In *Proceedings of OZCHI 2003*, 2003.
26. J. Vogel, W. Geyer, L.-T. Cheng, and M. J. Muller. Consistency control for synchronous and asynchronous collaboration based on shared objects and activities. *Computer Supported Cooperative Work*, 13(5-6):573–602, 2004.