Real-time Collaboration in Activity-based Architectures

Jakob Eyvind Bardram and Henrik Bærbak Christensen
Computer Science Department, University of Aarhus
Aabogade 34, 8200 Aarhus N, Denmark
{bardram,hbc}@daimi.au.dk

Abstract

With the growing research into mobile and ubiquitous computing, there is a need for addressing how such infrastructures can support collaboration between nomadic users. We present the activity based computing paradigm and outline a proposal for handling collaboration in an activity-based architecture. We argue that activity-based computing establishes a natural and sound conceptual and architectural basis for session management in real-time, synchronous collaboration.

1 Introduction

Pervasive computing technologies where a multitude of interconnected computing devices are freely and easily available hold the promise to support nomadic and collaborative work. A major problem, however, is that the software architectures are still based on the desktop metaphor, that essentially assumes stationary and concentrated work in which computational artifacts play the central role. Nomadic work, of which clinical work at a hospital is a major example, is just the opposite: characterized by mobility, interruptions, collaboration, and a focus on people, not computers.

The Activity-Based Computing framework (ABC) [1, 4] and Project Aura [5, 6] have proposed a new paradigm: Activity Based or Task Level computing. In these paradigms, the human notion of a task or an activity is directly supported by the software architecture, thereby trying to provide a more explicit support for mobility, interruptions, and frequent change of devices. So far these proposals have focused primarily on the architectural aspects of handling an individual’s set of activities. However, in real life people collaborate to complete tasks and an activity aware architecture must address this important, but complex, topic.

The contribution of this paper is to report on architectural aspects of supporting collaboration in an activity-based computing context. We report that the activity-based computing paradigm is a natural and strong paradigm for supporting collaboration, especially in order to treat the problem of session management. Next, we argue that an activity management system alone cannot adequately handle collaboration due to the high volume of fine-grained information that must be exchanged between participating devices, and propose an architecture that introduce additional components to manage collaborative information. We present our initial architecture for collaboration support in the ABC framework and discuss observations made during evaluation in a number of workshops.

2 Activity-Based Computing

Central to the concept of Ubiquitous Computing is to pervade the everyday environment with large numbers of computers, which are used transparently and opportunistically by users. These UbiComp visions are highly relevant in hospitals, where clinical work is characterized by mobility, the need to handle large amounts of complex information, constant interruptions, and a high degree of collaboration. A future scenario where physicians may use the computer built into the patient’s bed to review a patient record, browse the medicine handbook on a hand-held he has just grabbed from the nearest docking-station, or discuss an X-ray image on a wall-sized screen with the expert radiologist located at his home, seems promising and appealing. There is no need to carry heavy or bulky equipment, rather patient records are available at any nearby device whose characteristics fit the task at hand.

Observing a physician using a computer for medical work will reveal that she is viewing an X-ray image, browsing the medicine handbook, and navigating the medicine schema in the patient’s medical record. However, if you ask her, she will tell you that she is prescribing medicine for this patient. Mainstream software architectures support the application level only: the X-ray viewer, the handbook browser, and the medicine schema application; however, it has no notion of the real activity, namely to prescribe medicine. Thus reestablishing working context when shift-
ing device is so cumbersome that it is seldom done in practice.

Figure 1. A single activity involves a set of computational services, each manipulating a number of data.

The Activity-Based Computing (ABC) Framework and Project Aura have proposed to support human activities and tasks as first class objects at the architectural level. Activities embody every running applications’ run-time state, and may be suspended and resumed on any computational device. This supports mobility and interruption: the physician in the middle of prescribing medicine that is forced to attend another patient briefly, simply suspends the activity. Later, she finds an available computer and resumes the activity, thereby restoring all applications as she left them. Thus, users do computing at the abstraction level of activities, as illustrated in fig 1.

Technically speaking ABC defines an activity as:

**Activity:** An abstract, but comprehensive, description of the run-time state of a set of computational services.

Given this description of activities, we then define:

**Activity-Based Computing:** A computing infrastructure, which supports suspending and resuming activities across heterogeneous execution environments and supports activity-based collaboration

### 3 Collaborative Activities

The scenario below gives an impression of collaboration support by the ABC framework.

Dr. Urban is working on his activity concerning patient Petersen. This includes open views on the patient record, medicine schema, medical notes, and X-rays. Something on the X-rays alerts Dr. Urban and he wants to discuss the details with the radiologist on duty. He invites the radiologist to participate in the activity. The radiologist on duty – Dr. John Jensen – is notified and accepts to join the activity. Upon joining, his laptop screen is reconfigured as it activates Dr. Urban’s activity, that is, services that support viewing X-rays, browsing medicine schemas, etc., are started; proper patient information fetched, and graphical views adjusted to reflect the same data and visual presentation as Dr. Urban is working on. Next, as the activity is collaborative, additional collaborative tools are started, for instance telepointers and audio link.

The scenario stresses a very important aspect of activity-based computing with regards to collaboration, namely that a human activity serves as the basis for collaboration. The key point is that as activities are first class objects in activity-based architectures they naturally becomes the center-point of collaboration. A recurring problem in Computer Supported Collaborative Work (CSCW) systems is that of session management, i.e. how do users get in contact with each other, how do they know who ‘makes the first move’, how is communication initialized, and what is the purpose of the collaborative effort? In activity-based computing, the activity per se becomes the conceptual anchor for session management, because an activity defines the participants of a collaborative session (Urban and the radiologist) and entail the purpose of the collaborative effort (X-rays of the patient). Furthermore, the activity on a technical level provides access to the technical setup of the collaborative session, such as the hostnames and location of the computers involved.

At first sight, activities may even suffice. As activities conceptually “snapshot” the run-time state of a set of application we can, at least in theory, use this mechanism to provide full collaborative WYSIWIS\(^1\), namely by treating each user event in any service as an implicit snapshot operation, followed by a broadcast of the resulting activity to all participants of the activity, and perform an immediate resume operation on the participants’ devices. Thus every device would be kept in complete lock-step with all others.

In practice, however, this is not feasible. Consider a collaboration activity with five participants. Anytime a person moves a scrollbar ten pixels, state information is collected in all services; packed in a state object; sent to the server; broadcasted to all collaborative clients; unpacked; and parsed by each ABC service, which reconfigures the state of the service according to this state object. This is clearly infeasible due to the performance overhead.

---

\(^1\)What You See Is What I See.
4 Architecture

We have responded to the performance challenge by keeping the main ‘snapshot-broadcast-resume’ idea, but to introduce a more fine-grained perception of state information in activities; and provide infrastructure to distribute fine-grained state changes in a more light-weight manner. The activity concept is outlined in the UML class diagram in figure 2.

An activity maintains a list of participants working on it. Associated to the activity is a session object that provides auxiliary collaborative aspects such as voice-link and telepointers. The session object furthermore maintains a list of active participants, i.e. the participants that are actively engaged in collaborative work. Objects implementing the SessionListener interface may attach themselves to sessions using the observer pattern protocol and are notified when sessions change state, like active participants joining or leaving the work on a given activity.

Finally, the set of state descriptions is refined. Instead of a single state description per ABC service, it is now refined into a set of component descriptions. A component description is a fine-grained state description typically holding the state of a single graphical user interface component, like a text entry field, a scrollbar, etc. It may, of course, also be non user-interface related information. This means that rather than packing-broadcasting-resuming full state information for a number of services we need only pack, broadcast, and ‘resume’ information for a very small delta-state, like for instance only the typed character in a text field, the 10 pixel movement of a scrollbar, etc.

The ABC Framework allows ‘activity-aware’ services to be programmed with minimal effort. It basically provides two things:

1. A run-time infrastructure (middleware) consisting of a central server that stores and manages a set of activities and services related to activities; and a client-side run-time middleware component that allows the user to manage, inspect, and suspend/resume activities as well as handle distribution of fine-grained component state changes to activity participants.

2. A framework for developing activity aware services. This takes the form of a small set of interfaces and contracts that developers must implement and adhere to in order for their services to communicate with the ABC middleware. Basically, services must implement functionality to pack their run-time state into a state object (for suspend) and parse this object back into a running service (for resume). Collaboration support is provided through an elaboration of (a subset of) Java Swing components that provide default packing and parsing of state descriptions. See [2] for details.
• **Activity Controller** is a client side component that handles the communication with the server side activity manager. The activity controller has a user interface that allows users to browse his/her activities, and resume any from a list.

• **State Manager** is responsible for handling resume and suspend of a given activity. For activities where two or more participants are active, the state manager is notified on every component state change (delta state changes); this delta state is forwarded to the activity manager that broadcasts it to every participant’s device. Upon reception of a delta state, the state manager immediately notified the relevant ABC service that is then responsible for interpreting and adjusting accordingly.

• **Session Manager** is responsible for the collaborative tools like telepointers and audio link. As performance are more important than lost information, session managers communicate peer-to-peer using UDP.

• **ABC Service** provides domain specific functionality and must enable state object packing (for suspend) and parsing (for resume). Collaborative, graphical, services can utilize a set of collaborative Swing components.

Thus, collaborative information takes two forms: delta state changes in ABC services and dataflows in collaborative tools like telepointers. Service state changes are broadcast through the activity server using an RMI protocol while collaborative tool state changes are broadcast directly between clients using UDP. Activity changes must be guarantied to be broadcast hence the more restrictive protocol; a few lost packages on telepointer activity is in contrast not very critical. Also, it is important that the server side activity manager is kept up-to-date with the precise state information within an activity in order to handle latecomers.

We have implemented the architecture discussed above in the ABC Framework [1]. We have conducted 15+ whole-day workshops where clinicians from the University Hospital of Aarhus have role-played a wide range of clinical scenarios using the ABC Framework and an ‘ABC-aware’ Electronic Patient Record to carry out these scenarios. An important observation was that collaboration was transparently initiated between clinicians and the ‘activity’ proved to be a natural concept for a collaborative session. The only main issue raised concerned the linkage of the audio connection to the activity – if a user is alternating between two or more activities, the audio link is temporarily lost. Performance and quality of collaboration was sufficient for the workshop setting of two to three persons working on the same activity.

Further detail can be found in Bardram et al.[3].

5 Conclusion

In this paper, we have outlined the basic idea of activity-based computing and discussed the issues of collaboration within this paradigm.

Our main contribution is to open a debate on how activity-based architectures handle collaboration issues along several lines. First, we have experienced how the activity concept is indeed useful for users to structure collaborative behavior, as an activity or task is a natural focal point of collaboration. Second, the computational activity becomes the natural architectural component to manage collaborative dataflows and define protocols.

Our second contribution is an architecture proposal for collaboration support in activity-based architectures. We have argued how the activity-based paradigm at the logical/conceptual level can support collaboration. However, as argued above additional refinement as well as architectural components are necessary in order to achieve proper performance.

While we cannot claim to have a detailed and in-depth quantitative evaluation of the architecture, we never-the-less have important qualitative data from our workshop evaluations. Our experience from these is that the activity concept indeed feels natural to clinicians for organizing their collaborative efforts when using a computing system. The proposed architecture was buildable and performed adequately with a limited number of participants on an activity.

References


