Mobile Software Agents for Location-based Systems

Sebastian Fischmeister

Software Research Lab
Department of Computer Science
University of Salzburg
Phone: +43.676.7770167
Fax: +43.662.8044.6403
Fischmeister@SoftwareResearch.net
http://www.mobileshadow.net

Abstract. As mobile computing matures, location-awareness as part of context-awareness gains more attention; especially, location-aware services for mobile users. This paper concentrates on the software engineering issues of location-aware services and presents Mobile Shadow as an successful example of a design and an implementation of a scalable, fault tolerant, and component-based service infrastructure for location-aware services. The paper presents the basic concepts used in Mobile Shadow, the requirements, and the resulting component architecture.

Keywords: location-aware, context-aware, infrastructure, proactive services

1 Introduction

The continuous trend of miniaturization of hardware and the exponential growth of computation power has created a wide spectrum of small mobile computing gadgets. These gadgets enable the 'anytime and anywhere' communication paradigm, which has lead to the trend of wireless communication technology such as the global system for mobile communication (GSM) or the wireless local area networks (WLANs). These two trends form the basis of mobile computing.

Due to the high proliferation of mobile computing gadgets and technology, concepts such as context-awareness and especially location-awareness have regained interest. Only ten years ago, the location of a computer user did generally not change. Today, the user carries his notebook from one location to another, connects to the network, and continues using network-based services. A location-aware system would notice such a location change and would offer services specific to that new location.

An example of such a location-aware service is a reminder service. *John wants to carry books home with him for the weekend. Therefore, he wants to receive a reminder, when he is going home. Hence, he sticks a post-it on the exit door of the office building to remind him not to forget the books.* Instead of a post-it

note, John could use an electronic location-aware service to remind him not to forget his books. John would enter the reminder text and most constraints (e.g., building exit, after 5pm) and the service will send a reminder as soon as all constraints are fulfilled (i.e., John leaves the building after 5pm).

The Mobile Shadow project at the University of Salzburg provides an infrastructure for such services. The system aims at three issues: proactive location-awareness, scalability/fault tolerance, and components & adaptivity. A location system basing on 802.11 WLAN technology enables proactive location-awareness. A decentralized infrastructure and the use of mobile code technology tackle the scalability and fault tolerance issues. And the use of agent technology and a component-based design support adaptivity by manipulation of the user.

Related research projects mainly concentrate on the locating issues and not on software engineering issues; Example projects are Cricket [15], Cyberguide [2], Active Bat [3], EventManager [13], or active badge [17]. Each system usually has one proof-of-concept service. The most closely related system is the sticke document approach [4]. However, this system has never been implemented and only ran as a simulation at a workstation. Furthermore, the project did not concentrate on the software part of location-aware services. Another closely related project is the Lancaster Tour Guide [5]. The project provides a location-aware service for tourists in the Lancaster area. The project bases on an 802.11 WLAN and provides location-sensitive tourist information upon request. Thus, it is a reactive service only. The system bases on an extended version of HTML and uses thin client devices.

The remainder of the work is organized as follows. Section 2 introduces the concepts for proactive location-aware services and mobile code. Section 3 describes the Mobile Shadow system and presents a scenario and implementation details. Section 4 explains the different types of adaptivity available in the Mobile Shadow system. Finally, Section 5 concludes the paper.

2 Concepts

The following paragraphs provide an introduction to the core concepts of the Mobile Shadow architecture.

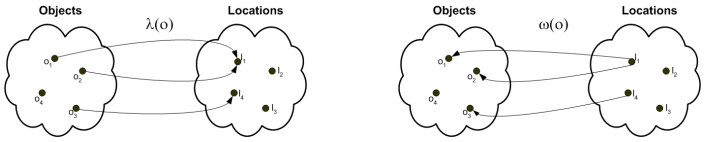
2.1 Locating versus Location

Systems combine the entities object and location in one of the two following relations:

1. Where is object o_1? — At location l_1.
2. Who is at location l_1 ? — The objects o_1 and o_2.

The first query refers to the mapping from objects to locations (see Figure 1(a)) and the second from locations to objects (see Figure 1(b)). We define systems using the first type of mapping (object → location) as locating systems

and the second (location → objects) as location system. The key difference between locating and location systems is the fact that locating systems focus on the identification of locations of objects, whereas location systems identify objects at locations. Each system requires different methods and mechanisms to provide the result to the query (for detailed discussion about the differences between these two mappings see [8]).



(a) **Mapping objects to locations (λ).**

(b) **Mapping locations to objects (ω).**

Fig. 1. Locating systems map objects to locations, whereas location systems map locations to objects.

2.2 Reactive versus Proactive Behaviour

Another important concept in the area of context-awareness is what we call proactivity versus reactivity. Currently, most location-aware systems offer reactive services only. The technology to build such reactive systems is already available [11] whereas current technology does not support proactive services out of the box [7].

Reactive services rely on the prevailing request-response communication mechanism, where the service complements the client's incomplete knowledge. The client actively "pulls" information from the service by issuing an explicit request. The most well known service, which uses this mechanism, is the World Wide Web (WWW) and the Hypertext Transfer Protocol (HTTP). When a user wants to visit a site, she enters the name, presses return (thereby issues the request), and will get the answer (the WWW server of that site will return the requested document).

Proactive services deliver or "push" information to a client without explicit request. Therefore, the user need not send an explicit request and will receive information automatically. Such proactive services work autonomously as background processes and they prompt/inform the user as configured. To use such a service, the client must subscribe to the service. During the subscription process, the client creates a profile, which forms the basis for distinguishing wanted and unwanted information. The reminder service mentioned in the introduction is a typical proactive service. The service would miss its purpose, if the client always has to query whether a reminder is set or not. Therefore, the reminder service runs in the background and prompts the user, whenever a reminder is set.

2.3 Mobile Code

The advent of Java in 1994 and its built-in support for simple network programming revived mobile code paradigms. Although, prior projects existed, many researchers started researching mobile code at that time [10]. In general, mobile code is about moving data and/or code. There are four different mobile code paradigms: code-on-demand, remote evaluation, and mobile agents.

The *Code-on-demand* paradigm got widespread with the advent of Java applets. The client requests code from a server, the server returns the binary code, and the client runs this code. In this paradigm, only code is transferred via the network. The *remote evaluation* paradigm bases on transferring code. The client transfers code to the server, the server executes the code and returns the result to the client. Finally, the *mobile agent* paradigm is a mixture of the previously mentioned paradigms. A mobile agent consists of code, data, and a program state (i.e., it resumes operation at the remote host where it has left off before). The key property of it is "autonomous". The client sends a mobile agent to the server and the server executes it. After the execution, the mobile agent can autonomously decide to move to another server or to return to the client. In this paradigm, the program code, the data, and the program state are transferred through the network. In this work we use mobile agent and mobile code component synonymously.

3 Mobile Shadow

The University of Salzburg has built an architecture for proactive cell-based location-aware services. It is called *Mobile Shadow* [1]. In contrast to related work, Mobile Shadow bases on a locating system; it is optimized to answer queries such as who is nearby the building exit.

In this project concerning the software architecture, we aim at particular goals: modifiability/adaptivity, scalability, and fault tolerance. Modifiability is important, because we want to provide a platform for location-based services and thus want to add new services later on. To ease the user interaction, we also need to support adaptivity. By adaptivity we mean adaptation by manipulation by the user (see [12] for a discussion about the different concepts). The other goal is scalability; scalability is important, because most location-based services do make sense only when they cover a large area and to cover a large area and many users, the service must be scalable and the service platform must provide scalability features to services. Another aim is fault tolerance; if the system covers a large area, the system must not fail completely, if one service crashes or one cell crashes.

3.1 System Architecture

Several cycles of architecture design and evaluation lead us to a decentralized and localized architecture (see Figure 2). Each cell has its own communication

technology and a connection link to the adjacent cells. In the figure each cell includes only a service pool which runs a replica of a local service (i.e., S_1 to S_7). Each user accesses the services directly in his cell. For instance, user A stands in cell A and accesses the service S_5 and S_7 in Cell A through the cell's own communication system. User B stands in cell B and also accesses the services through the cell's own communication system.

This architecture satisfies our requirements. Concerning scalability, different users in different cells access different replica of the same service. Therefore the service access will not become a bottleneck. Also, we can multiply the number of service pools in one cell, so we also can perform dynamic load balancing. Concerning fault tolerance, if the communication technology in one cell fails, it does not affect the other cells. For example, if the communication technology in cell A fails, User B in cell B can still access the services.

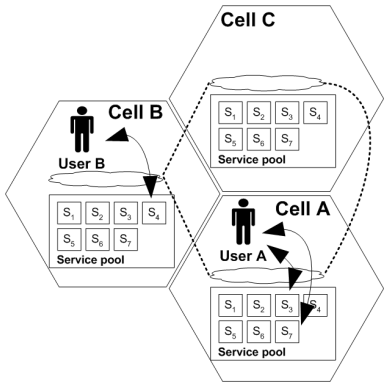


Fig. 2. The decentralized and localized architecture of Mobile Shadow.

3.2 Computation Model

Mobile Shadow meets the stated requirements and implements the suggested scalable, fault tolerant, and modifiable architecture through different models. What we call space model describes how we map the reality onto the system architecture (see below for a description of all models). The user model and the service model show how we represent users and services in this virtual reality. The user-space and the service-space model explain how we handle user and service movements. And finally the user-service model shows how we model user access to services.

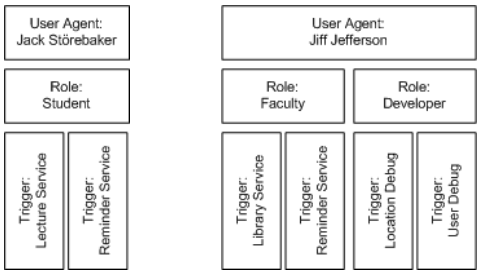
Space Model. In the Mobile Shadow system, each physical place also has exactly one logical counterpart. The physical environment is split into several separate small cells and each of these cells has a virtual representation. For example, the location "building exit" exists as "shadow building exit" location, too.

Furthermore, the space model allows adding a hierarchical structure as a tree on top of it. Several cells can be virtually merged into one larger cell and several cells can be treated as one larger cell but still exist separately. For example, our computer science building consists of several cells, however, the whole building also represents a virtual location "computer science building".

User Model. Each user owns a virtual "alter ego" (a mobile code component). This component, called user agent, always resides at the logical counterpart of the current user's location. Therefore, if a user agent resides at the virtual "building exit" location, then the real user also resides at that location.

Additionally, each agent is associated with several roles. Each role defines the specific set of services available for the user. For example, staff members can access different services than students or visitors.

Each service consists of a trigger and a service implementation. The user agent runs the trigger and the local infrastructure runs the service implementation. A trigger consists of trigger constraints, a personal configuration, and a trigger action. The user can configure what triggers his agent has and the user can also set the personal configuration of each trigger via a small command line application or a web interface. The trigger constraints define conditions in order to call the trigger action such as time, location, or available services. Once all these constraints are satisfied, the trigger action activates the local service and transfers the personal configuration of the user for this service.



(a) **A student user agent.**

(b) **An example user agent for a Mobile Shadow developer.**

Fig. 3. Example user agents. Each user can have several roles and different triggers for each role.

Figure 3 shows two user agents, their roles, and the included triggers. Although some roles may share the same triggers such as the local MessageBoard (see below), they also offer a different set of services (e.g., see Figure 3(b) the developer role,) or have different access permissions (e.g., the lecture reminder service is equivalent to the reminder service, however, the user has predefined reminders).

User-Space Model. To provide location-aware services, Mobile Shadow transforms user movements into virtual movements. Thus, if the user moves from location A to location B, then the user agent will move from the virtual location A to the virtual location B; Once the user agent arrived at the new place, it accesses the local resources such as a local database, other local user agents, and local available services.

Service Model. Services, similar to users, have a virtual representation. For example in a tour guide service, a museum painting offers a specific service; e.g., an insight into the art of the previous century. This painting owns a virtual service at its location. The service offers this information to user agents who visit this location. The service can even notify the user of its existence, depending on the trigger configuration of the user agent.

Real world services and services—i.e., services that are only available in the virtual user space—are available in Mobile Shadow. Such services are often real world services transformed and extended into digital services. For example, a local message board service (see GeoNotes [6]); the service is similar to a blackboard, however, the notes are in an electronic form and the blackboard provides different notes at different locations. A user can read, add, or remove notes at any location she wants to.

Service-Space Model. The service-space model is static in contrast to the user-space model. So far, the Mobile Shadow project only consists of real and virtual services that do not move. However, one could think of moving services such as services that chase viruses. Service such as the painting service have a specific location and work with localized or personalized data.

User-Service Model. When a user agent moves to a new location, this location change activates the triggers of each service. The trigger tries to fulfill the trigger constraints. In case it succeeds, the trigger action activates this service and transfers the personal configuration parameters. Afterwards, the activated and personalized service processes the local data. If the service finds some information that may be interesting for the user, it will notify her according to the personal setup (e.g., via short message service or via display message or via email).

3.3 Mobile Shadow Component Architecture

Each location-aware service consists of two parts: the service and the trigger. The *trigger* contains the user specific configuration parameters, the trigger constraints, and the trigger action. In simple cases, the configuration cannot be changed and the action is a simple method invocation. The *service* contains all server-side business logic such as database access or data processing. The *user agent* manages the roles and triggers (e.g., deactivating them before moving to a new location or activating them after moving) and provides the basic functionality for the triggers (e.g., finding the service dock). The *service dock* controls

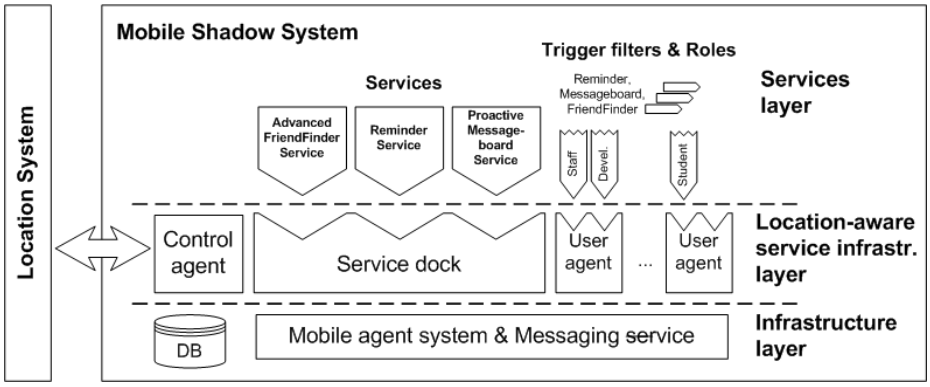


Fig. 4. Component concept of Mobile Shadow. The system has a modular design, which enhances support for adding and exchanging components.

the services (e.g., registration of new services or starting and shutting down services) and provides some basic functionality (e.g., finding a service or gaining database access). The *control agent* manages the communication between Mobile Shadow and the location system and provides interfaces to external resources such as partner research projects. Furthermore, the control agent manages the user agents, e.g., it is responsible to move user agents to their new location. Finally, the *mobile agent system* provides the basic infrastructure for mobile code, which also includes an inter-agent messaging system. Finally, the *location system* tells the control agent, when an agent did move from one place to another.

3.4 Example Scenario

A location-aware service is the reminder service. The introduction already presented the objectives of the service. Among other services, Mobile Shadow provides this reminder service.

User John wants to carry books home with him for the weekend. Therefore, he wants to receive a reminder, when he is going home. In the Mobile Shadow system, he opens the WWW interface to add location-based reminders. He enters/selects following data: cell id (pull down list), reminder text (maximum of 140 characters[1]), reminder start time and start date, reminder end time and end date, how often he wants to be reminded of this one event (e.g., John could wish to be reminded every day to lock his door, then the number of reminders is infinite). The cell identification is equivalent to the location. The correct cell for this example is the building exit cell (i.e., *AP Hall*). The reminder text is "Hey! Don't forget the books for the weekend!". The reminder start time is "5pm" and the start date is "today". The reminder end time is "11pm" and the end date is "today". The reminder is only active between the start time and the end time.

[1] Messages may be delivered via the short message service to for cellular phones. Such messages have a limit of 140 characters.

This prevents for instance, that John also receives the reminder, when he leaves for lunch. Finally, John can also define the number of reminders that he wants to receive. In this example, John sets the reminder count to "1". Therefore, John will receive only one reminder and then the system will deactivate the reminder. After the John filled out all the required fields, he submits the data and thus configures the trigger carried around by his user agent.

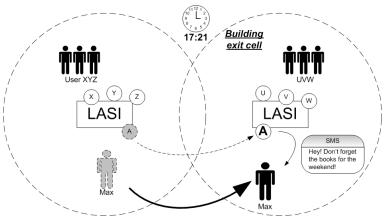


Fig. 5. Example Scenario "John and the books.". John must not forget to carry books home for reading, thus he did set a location-based reminder in the exit cell of his building.

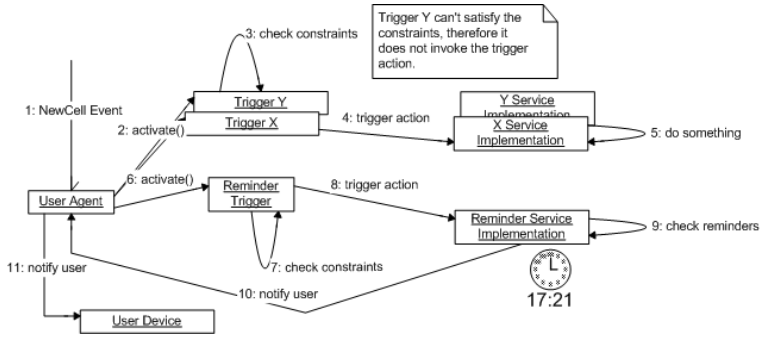


Fig. 6. Each time the agent arrives at a new place, it receives an event from the agent system and then it activates the triggers.

Figures 5 and 6 show what is going to happen, when John is leaving the building after 5pm. John moves through the building and thus enters and leaves several locations. At each location the user agent receives the *NewCell* event from the Mobile Shadow platform. Finally, John and his user agent enter the building exit cell. Arriving in this cell, the user agent again activates the triggers. The reminder trigger checks its constraints and executes the trigger action. Because it is past 5pm and before 11pm, the date is today, and the reminder is still active (the reminder count is greater than zero), the service will return a valid

reminder. Now, the user agent uses the short message service (SMS) and sends the reminder to the John's cellular phone.

3.5 Implementation

The Mobile Shadow research project has consisted of two phases. Researchers at the University of Constance built the first prototype of the Mobile Shadow system. This first prototype implemented the whole functionality except the location system. Therefore, user agents had to be moved manually to demonstrate the system. The first prototype was implemented in Java, used the Aglets mobile agent system as basic infrastructure, and MySQL as database management system. Researchers at the University of Salzburg built the second prototype of the Mobile Shadow system. This prototype now includes the location system, the interface to Mobile Shadow, and three running services. The second prototype is implemented in Java but uses the Grasshopper Agent system [16] (to increase independence from the underlying system) as basic infrastructure and MySQL as database management system.

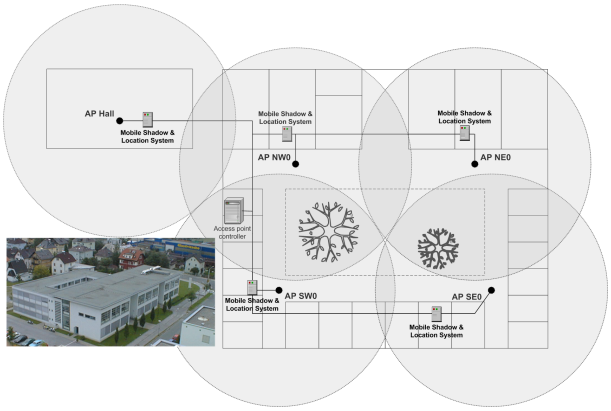


Fig. 7. Computer science building ground floor and integrated Mobile Shadow infrastructure. The building plan shows five WLAN access point and how the Mobile Shadow system is integrated into it.

Based on an evaluation of wireless communication technologies [7], the second prototype was built on top of an 802.11b wireless local area network [14]. Figure 7 shows the ground floor of the computer science building. In addition the figure shows five access points of the WLAN and the integrated Mobile Shadow system. There are a total of 11 access points distributed on three levels of the building. However, the component architecture in Section 3.3 showed that the location system is only loosely coupled, so we can easily exchange the WLAN location system by commercial ones or research projects such as Cricket [15], Cyberguide

[2], or Active Bat [3], which would provide a more exact locating system than the WLAN system.

To realize the location system, the WLAN hardware vendor Proxim cooperated with the university and provided information about the implementation of the hand-off protocol (a client switches from one access point to another). In order to detect location changes, the locating system of Mobile Shadow analyses the network traffic between the access points and a central access point controller. When a client switches from one access point to another, then the new access point registers the client at the central controller. Therefore, the location system knows when to move the user agent. The system determines the destination of the user agent by querying adjacent access points. Once it has found the destination, it notifies the control agent of the location change.

In addition to a virtual place for each location, the Mobile Shadow system also requires a place to store 'homeless' user agents. This place is called *garage.* Once the user turns off the WLAN device, the locating system can no longer track the user. After a timeout, the location system issues a location change to the garage place. Then, the Mobile Shadow system moves the user agent to the garage. Besides easier maintenance, this simplifies the restore process once the user again turns off the device. In case of a restore (the user joins the WLAN again), the system knows where it can to find the agent[2].

4 Adaptivity

The computation model on which Mobile Shadow bases and its modular design allow adapting the system in three simple and efficient ways: (1) the user can define an specific configuration for each service (e.g., the user can define the list of friends for the FriendFinder service or the kind of notification of reminders), (2) the user can add triggers and thus subscribe to new services, and (3) the system designer can change user roles so they suit the new requirements.

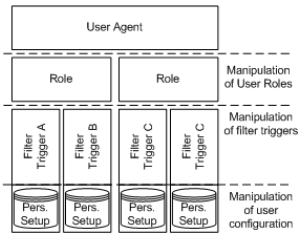


Fig. 8. Manipulation of the user agent setup. The user can adapt her user agent at three different layers.

[2] To provide a scalable solution for the *garage*, we use a peer to peer model that is beyond the scope of this paper.

Manipulation of the User Configuration. The easiest way of adapting the system to the personal needs of a user is to change the configuration of the trigger. Each service offers different parameters and each user can adjust this parameters to her personal needs via a web service. For example, the main intention of the FriendFinder service is to automatically notify the user whenever one of his friends enters the same location. The basic setup is to define this list of friends (i.e., enter the phone numbers of them). But there are also general settings, which are not specific for a certain trigger. For example, Mobile Shadow uses SMS as default notification service, but some users prefer other notification services. Therefore, users can also configure the notification mechanism (e.g., show a message on the device or send an email).

Manipulation of Triggers. The second possibility for a user to adapt the system is to add or remove triggers to and from his user agent. A separate WWW interface allows configuring these components and thus allows the user adapting her subscription setup. Due to use of component technology and Java, the user can execute the changes during runtime and no restart is necessary.

Manipulation of User Roles. Finally, the third possibility to adapt the system is to introduce new role components. The user agent is only the vehicle that carries and manages the roles. The roles provide the basic features to the triggers. Different users may need different basic features and therefore need different types of roles. For example, staff members belong to a different role than students (see the previous example in Section 3.2).

We do not allow users to assign or program roles themselves. The main reason is security. The role concept allows us to run a strict security policy (e.g., developers can access anything, staff services have read-write database access, student services have read-only database access) that the underlying agent platform can enforce. Otherwise we risk running into security problems as agent systems are not yet secure enough [9]. Therefore, if a user needs a special role, then she must ask the system designer and developer to assign or implement the new one.

5 Conclusion

Mobile Shadow is an infrastructure that provides support for proactive location-aware services. The main aims of the research project are to provide a sound software architecture that supports modifyability/adaptivity, scalability, and fault tolerance. The paper presents details about the concepts used in Mobile Shadow, its design and implementation that allow us to meet the requirements. In contrast to the related work, Mobile Shadow does not concentrate on technical solutions of locating an object but it concentrates on the software architectural issues of service platforms for location-aware services.

As future work, the researchers plan to open the agent system codebase so any user can program his own user agent. Due to the success and the novelty of

the approach, the researchers plan to evolve the Mobile Shadow prototype to a product.

References

1. Mobile Shadow. WWW Site. `http://www.mobileshadow.net/`.
2. G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide. *Baltzer/ACM Wireless Networks*, 3(5):421 to 433, October 1997.
3. M. Addlesee, R. Curwen, S. Hodges., J. Newman, P. Steggles, A. Ward, and A. Hopper. Implementing a Sentient Computing System. *IEEE Computer*, 34(8):42 to 48, August 2001.
4. P. Brown. The stick-e document: a framework for creating context-aware applications. *Electronic Publishing*, 8(2&3):259 to 272, Jun & Sep 1995.
5. N. Davies, K. Cheverst, K. Mitchell, and A. Efrat. Using and Determining Location in a Context-Sensitive Tour Guide. *IEEE Computer*, 34(8):35 to 41, August 2001.
6. F. Espinoza, P. Persson, A. Sandin, A. Nyström, E. Cacciatore, and M. Bylund. Geonotes: Social filtering of position-based information. In *Proceedings of Ubicomp 2001*, Atlanta, GA, 2001.
7. S. Fischmeister. Cell-based Pervasive Networking Technologies for Location-Aware Services. Technical Report C46, Software Research Lab, University of Salzburg, Austria, February 2002.
8. S. Fischmeister and G. Menkhaus. L^2: A Novel Concept for Cell-based Location-Aware Services. Technical Report C45, Software Research Lab, University of Salzburg, Austria, February 2002.
9. S. Fischmeister, G. Vigna, and R. Kemmerer. Evaluating the Security Of Three Java-Based Mobile Agent Systems. In *Proc. of IEEE Mobile Agents 2001*, Lecture Notes in Computer Sciences, December 2001.
10. A. Fuggetta, G. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342 to 361, May 1998.
11. J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, 34(8):50 to 56, August 2001.
12. E. Horvitz. Principles of Mixed-Initiative User Interfaces. In *Proceedings of CHI'99*, ACM SIGCHI Conference on Human Factors in Computing Systems, May 1999.
13. J. F. McCarthy and T. D. Anagnost. EventManager: Support for the Peripheral Awareness of Events. In P. Thomas and H. W. Gellersen, editors, *Handheld and Ubiquitous Computing*, number 1927 in Lecture Notes in Computer Sciences, page 227 to 236. Springer Verlag, Germany, September 2000.
14. B. O'Hara and A. Petrick. *The IEEE 802.11 Handbook: A Designer's Companion*. Standards Information Network IEEE Press, 1999.
15. N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket location-support system. In *Proc. of the sixth annual international conference on Mobile computing and networking (MobiCom 2000)*, page 32 to 43. ACM Press, 2000.
16. IKV++ GmbH Informations und Kommunikationssysteme. *Grasshopper: The Agent Platform - Technical Overview*. IKV++ GmbH Informations- und Kommunikationssysteme, February 1999.
17. R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.