The User Agent: An approach for service and profile management in wireless access systems

Carsten Pils
Informatik 4 (Communications Systems)
RWTH Aachen, 52056 Aachen, Germany
pils@i4.informatik.rwth-aachen.de

Jens Hartmann
Ericsson Eurolab Deutschland GmbH
Ericsson Allee 1, 52134 Herzogenrath
jens.hartmann@ericsson.com

Abstract

With their special ability to operate disconnected and autonomously mobile agents are well suited for wireless access networks: Mobile user can dispatch mobile agents to the fixed network. While the agent operates autonomously in the fixed network, the user's mobile device disconnects. Reconnection is only required when the agent needs user feedback or returns results. Thus, mobile agents have the ability to save wirless network resources.

While agents operate disconnected they require a certain knowledge of the user's preferences for service trading. But user preferences might change frequently and thus agents require a central information storage which provides up to date user preference descriptions. In this paper we propose the User Agent which manages user profile entries and acts as the user's central service trader. Agents can request services at the User Agent which correspond to the user's preferences.

Keywords: *Mobile agents, service trading, agent management, disconnected operations, personal mobility.*

1 Introduction

Although 3rd generation communication systems like UMTS will offer mobile users network access at considerable quality of service, radio resources are still scarce and suffer from interferences. Thus, applications programs specially designed for mobile users should be optimised for saving network resources and must tolerate frequent disconnections. With their special ability to operate disconnected and autonomously mobile agents are well suited for wireless access networks [1]: An agent based application running on a mobile device can dispatch a mobile agent to the fixed network.

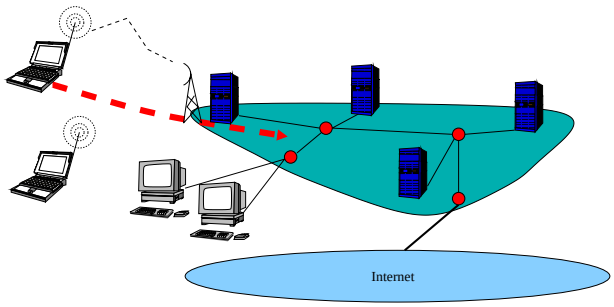

Figure 1. Wireless Access Network

While the agent autonomously operates in the fixed network the device disconnects. Reconnection is only required when the agent needs user feedback or returns results.

Thus, mobile agents have the ability to save networks resources and tolerate frequent disconnections. In recent years, many research projects have investigated mobile agent usage for wireless access networks [2], [3], [4], [5], [6]. The objective of these projects is the development of mobile agent management approaches to support terminal and personal mobility.

Most management facilities offer agents a certain degree of freedom with respect to quality of service. Agents can request different quality of services degrees by submitting parameters or use different services. To discover services there is a need for a service trader. Park et al proposed a Service Center which allows agents to retrieve service references from a distributed database [7]. If required the Service Center returns a list of appropriate services. Thus, agents can choose a service based on user preferences out of a multitude of alternatives to complete their tasks.

As user preferences and registered services might frequently change, fixed service lists are not useful for agents. To solve this problem agents can carry ser-

vice descriptions and trade services right before service usage. But the latter solution still has some disadvantages: First, the agent size is increased by service descriptions. Second, processing of agents is delayed due to the service discovery and selection process. Third, once the agent is launched the user cannot modify the service descriptions.

To solve this problem, we propose the User Agent which supports mobile agent service trading and management. Basically, the User Agent acts as an active extension of the user profile. It is envisaged to implement the User Agent in JAE (Java Agent Environment) [6].

The remainder of this paper is organised as follows: Next we briefly describe the JAE agent technology in section 2 and the User Agent in section 3. Following, we give an example of how the User Agent supports mobile agent management 4. Performance considerations are addressed in section 5.

2 The Java Agent Environment

Each JAE agent system provides the core technology for mobility, security and services. An integral part of an agent system is the Service Center. The Service Center takes care of the administration of local services and provides a trading mechanism for remote services. Remote service trading requires a central repository for all available services. The current release of JAE uses the LDAP directory service for that purpose, as it has the advantage of providing a distributed, standardised, and publicly accessible database.

2.1 The JAE Service Center

The JAE technology distinguishes between mobile and fixed service agents. While mobile agents can migrate to different agent systems by means of the *Agent Transport Facility*, fixed service agents stay at the agent system where they have been launched. Since service agents cannot migrate they are considered trustworthy and are allowed to access system resources. Mobile agents must always interact with a service agent when accessing system resources. Apart from providing a controlled access to system resources, service agents can also provide application services. Anyway mobile agents must discover service agents. Therefore, agents issue a service description of the desired service to the Service Center (see figure 2). The latter either mediates the request to a suitable local service agent or provides information about another agent system that offers the requested service.

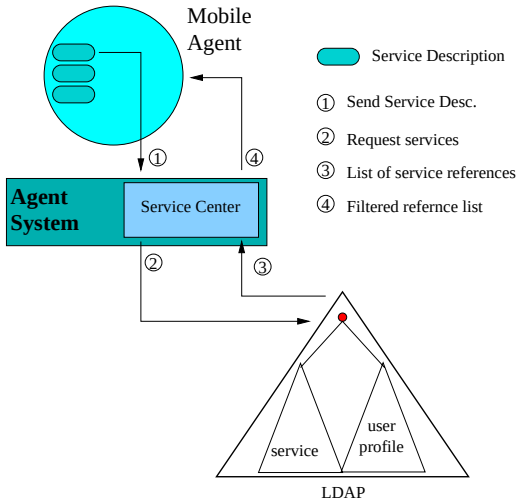

Figure 2. Service Trading

Non local services are looked up in the LDAP directory tree. Among others the LDAP directory has a service and a user profile subtree. Below the service subtree service descriptions as well as service reference information are stored [7].

2.2 The User Profile

User profiles are stored below the user profile subtree. They can be accessed via service agents and contain for instant the user's private as well as business email address, his phone and fax number, and the system he is currently logged in. The user's current agent system (i.e. the agent system he is currently logged-in) is used to provide personal mobility.

Personal mobility allows a user to access communication services through different devices, e.g., at home, office, mobile phone, notebook, PDA. This means that the user has several different access points. Agents which need user feedback information or return results must know the user's current agent system. By applying the user profile, agents can retrieve the current agent system of a user: When the user logs-in at an agent system, the latter writes its address to his user profile. Now if an agent wants to get in contact with the user the agent can look up the user profile and retrieve the current agent system's address.

3 The User Agent

The User Agent is an active extension of the user profile. Basically, its envisaged application is to act as the user's central service trader. Thus, mobile agents

can request services from the User Agent which fit the user's preferences.

To fulfil this task the User Agent comprises of a static and a dynamic database. The static database contains the user profile, his preferences, and service descriptions. Applying the stored service descriptions and the user's preferences the User Agent looks up services with the help of the Services Center. Those services which match both service descriptions and user preferences are stored in the dynamic database. Once the dynamic database is generated, mobile agents can request a service from the User Agent. The User Agent just retrieves a suitable service from its dynamic database and sends the reference to the mobile agent.

3.1 Service Trading

As mentioned before the User Agent retrieves service reference from the Service Center using service descriptions and preferences stored in the static database.

3.1.1 Service Descriptions

JAE service descriptions contain all information about a service agent. These informations are stored in attribute-value pairs. There are four attribute-value pairs specifying a service:

1. *ServiceType*: Description of the service category.
2. *Action*: Functions offered by this service.
3. *Parameters*: Parameters accepted or needed by this service.
4. *Return Values*: Type values returned by the service functions.

Since there must be common base values for these attributes, JAE introduces the so-called *Met-Keywords* that agent programmers use to specify services. Service agents use these key words to describe the service they offer, while mobile agents need them to specify the service they want to use. For example, a service description object of a bank could look as follows:

- *ServiceType* = BANK.MY_BANK
- List of ACTION descriptions:
  - *Action*= CHECK_BALANCE
  - *Parameters*= INTEGER.ACCOUNT_NUMBER
  - *Result Value*=OBJECT.STATEMENT
  - *Action* = GET_STOCK_INDEX
  - *Parameters* = STRING.STOCK_MARKET_ID
  - *Result Value* = INTEGER.INDEX

A service agent of "MY_BANK" could register its reference with this service description object at the Service Center. An agent which wants to use a banking service to get the index of a particular stock market could issue the following service description to the Service Center:

- *ServiceType* = BANK
- List of ACTION descriptions:
  - *Action* = GET_STOCK_INDEX
  - *Parameters* = STRING.STOCK_MARKET_ID
  - *Result Value* = INTEGER.INDEX

Among others the Service Center will return a reference of the BANK.MY_BANK service. Note that the service function filters service descriptions with respect to the *ServiceType*. Thus, if an agent issues the same service description to the Service Center, but *ServiceType* set to BANK.MY_BANK service, the Service Center would only return the BANK.MY_BANK service [7].

3.1.2 Preference Definitions

Along with the service description, a service agent can store quality of service information in the service directory. These information are stored in attribute value pairs. While the attribute is just a string the value is an object allowing a complex quality of service description. For example, the BANK.MY_BANK service could describe the costs of a single transaction as follows: attribute=COST.TRANSACTION, value={ float=0,1; STRING="USD" } — a single transaction amounts to 0,1 US dollar.

It is the User Agent's task to order services with respect to user preference. To this end the User Agent calculates a preference number for each service.

Numerous attribute-value pairs can be associated with one single service. The user will value each quality of service parameter differently. To ease individual calculation of preference numbers (with respect to individual services) we propose the use of preference objects. Each preference object is associated with a service and has two methods: STRING getKey() and INTEGER getPreference(service). getKey() returns the name of the associated service and getPreference() returns the preference number. To calculate a service preference number the getPreference() method requires its quality of service descriptions (retrieved from the Service Center) as input parameter. Quality of service information is stored in the service reference object.

Preference objects are created by the user and submitted to the User Agent. Each service entry in the

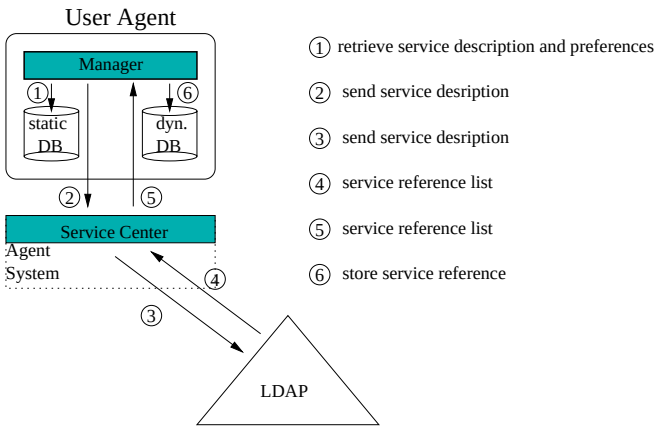

Figure 3. Generating dynamic entries

static database is identified by a service key. The entry comprises of service key, service description, and preference object.

3.2 Manager

Basically, the Manager controls the User Agent. Among others it allows modification of the static database and generating of the dynamic one.

3.2.1 Generating dynamic entries

Generating of dynamic database entries is depicted in figure 3. The Manager fetches an entry from the static database and issues the service description to the Service Center. The latter responds with a list of service references. Next, the Manager applies the preference object to the reference list and sorts it in order to the calculated preference numbers.

Finally, the Manager stores the best service reference in the dynamic database. Again the entry is identified by the service key.

An agent which requests a service reference from the User Agent must only provide the service key. Having the key it is straightforward to retrieve a service reference from the dynamic database. Note that it is required that agent applications and User Agent must agree about a service key.

A dynamic entry is always updated when the user modifies the preference object. As services might change their quality of service parameters, it is required that dynamic entries are updated at regular intervals. To this end the user specifies an update interval for each entry. Furthermore, a service entry is updated if an agent reports that the recommended service is not available.

3.2.2 User profile management

Hitherto, the User Agent showed to be the user's central service trader. But being an active extension of the user profile its ability goes far beyond mere service trading. We motivate additional features of the User Agent with the help of the following example:

A user runs an agent based application in his office. During his free time he might be committed to a honorary position and runs a special agent application. Thus, the user might want to run three current agent systems: one in his office, one at home related to the honorary position, and one "private" agent system. For each of these agent systems the user is expected to have a different user profile. Thus, the user is registered to the agent world three times. But registering three user profiles has several disadvantages: First, each user registration consumes system resources. Second, the user must access several user profiles if he wants to modify a common entry. Third, removing or adding a user profile is quite complicate.

To ease managing multiple user profiles we propose that the User Agent allows maintaining more than one user profile. It distinguishes different profiles by adding a suffix to the user's login name. With respect to the given example a user could have the following profiles: *user:business*, *user:honorary_position*, or *user:private*. Our approach has several advantages:

- System resources are not wasted since the user's different profiles share information.
- The user must access only one User Agent to modify his user profile.
- Removing and adding a user profile is simple since it must not be registered by the network provider.
- Since the User Agent can access the user profiles, it can forward information from one profile to another. For example, a user does not want that agents related to his business profile send messages to his private profile. But if an agent has some important business information then the agent should send the information to the user's current agent system. As each agent must authenticate to the User Agent, the latter knows the agent's owner identification. Based on agent and agent owner identification the User Agent can decide in accordance to user defined rules which information should be forwarded to the agent. A simple approach for user defined rules are access control lists. But access control lists do not allow description of complex forwarding rules like: "If agent X returns result to my business profile indicating *emergency* give all business related agents

access rights to my private user profile entries: current agent system, email address, and fax". There is a need for a rule based information exchange management.

With its ability to maintain multiple profiles the User Agent eases profile management and allows flexible personal mobility management.

Equipped with service trading and user profile management functionality the User Agent possesses powerful tools to shield the user from agent management tasks. Furthermore, our approach supports agent management services. In the next section we show how a management service can utilise the User Agent.

4 On utilising the User Agent

When an agent requires user feedback or returns its result while the user is not logged-in or disconnected, the agent might decide to wait until the user logs-in or reconnects. If this waiting is a busy waiting the agent consumes agent system resources. Since the time until the user's current agent system is available again has no upper limit, agent systems are in danger of getting blocked by waiting mobile agents. Thus, agent systems should provide a mechanism which allows suspending agents while the user's current agent system is not available. This would allow the agent system to store waiting agents persistently and free up memory and processing resources.

One of the first agent systems which considers agent waiting support is Agent TCL [2]. But the so-called docking mechanism scales only if the docking machines are appropriately distributed. Furthermore, personal mobility is not considered: agents wait for disconnected agent systems.

The Java based agent system Concordia addresses agent waiting support by a Queue Manager [5], [8]. The latter provides a store and forward mechanism and allows to enqueue agents in a asynchronous manner. Agents can be stored in the queue of a local server while a remote server is disconnected. When the remote agent system comes back online, the local server would forward the agent to the remote server. Again personal mobility is not considered.

AMASE is a European ACTS Project adapting an existing mobile agent system (SWARM system of Siemens AG) to the requirements of a wireless computation environment.

AMASE supports agent waiting support by implementing a so-called Kindergarten Service [9]. Although personal mobility support is considered, a Kindergarten Service is not implemented on each agent system due to memory constraints of small devices. But particular small mobile devices require Agent Waiting support.

Following, we propose the Waiting Support Mechanism which suspends mobile agents and resumes them when the user's current agent system is available.

The Waiting Support Mechanism comprises of three agents: the Waiting Support Service Agent, the Waiting Support Proxy Agent, and the User Agent. Each Agent System runs a Waiting Support Proxy, while User Agent and Waiting Support Service are running only on wired connected agent systems. We assume that in general wired connected agent system have more processing as well as memory resources than wireless connected ones. Furthermore, the probability that a wired connected agent system is disconnected or fails is negligible.

4.1 The Waiting Support Proxy Agent

Each agent system runs a Waiting Support Proxy Agent. The Waiting Support Proxy is responsible for handling frequent disconnections of the agent system. If an agent system is disconnected, mobile agents running on this system can neither reach another agent system nor a Waiting Support Service. In that case mobile agents can use the local Waiting Support Proxy. The Waiting Support Proxy has a reduced functionality set and refuses mobile agent registration requests if the agent system is connected to the agent world.

The Waiting Support Proxy also runs a notification service. If the Proxy detects that its agent system has recently reconnected then it sends requests to the User Agent. The latter might send notification messages to registered Waiting Support Services (see protocol description below).

4.2 The Waiting Support Service

From the mobile agents point of view, we must distinguish two scenarios: First, a Waiting Support Service is available. Second, no Waiting Support Service is available (local agent system is disconnected).

4.2.1 A Waiting Support Service is available

We have depicted this case in figure 4. The agent system on which the mobile agent is running is connected to the Agent world while the user's current agent system is temporarily disconnected. If an agent detects that a destination agent system is not available it requests a Waiting Support Service from the User Agent **(1)**. The mobile agent migrates to the Waiting Support Service **(2)** and calls its registration method indicating

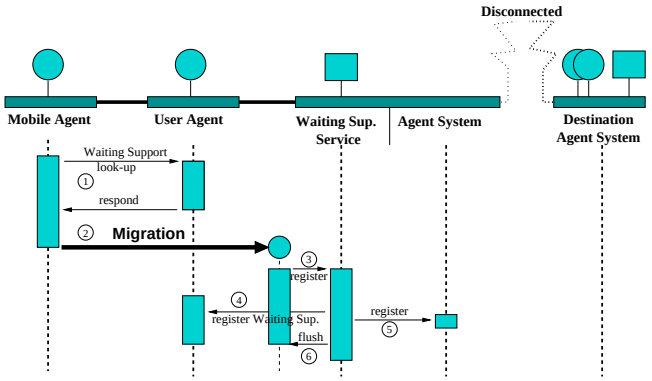

Figure 4. Waiting Support Service is available: Registration

the name of the agent's destination, the maximum suspension time, and optionally, the agent's resource requirements **(3)**. The Service must register at the user's User Agent **(4)**. In the next step the Service must also register with the local agent system to receive communication events addressed to the mobile agent (the agent should be resumed to process any communication request) **(5)**. Finally, the Waiting Support flushes (suspends and stores persistently) the mobile agent and registers it with the agent's destination user name in its local database **(6)**.

Mobile Agent wake-up An agent can be resumed for three reasons: The agent's suspension timer expires, the Waiting Support Service receives an communication event, or the user reconnects.

In the two former cases the Waiting Support Service de-registers from the User Agent and resumes the mobile agent.

If a destination agent system reconnects its Waiting Support Proxy (which has registered with the agent system to receive network events) detects this and starts the Waiting Support notification process. The notification and wake-up process is depicted in figure 5: After detecting a system reconnect a local Waiting Support Proxy notifies the user's User Agent **(1)**. The notification message includes the destination agent system address and an agent system resource description. The User Agent forwards the notification to the registered Waiting Support Services (according to multi user profile forwarding rules) and stores the current agent system address in the dynamic database **(2)**. Upon receiving the notification, the Waiting Support Service retrieves the resource requirements of all mobile agents waiting for the agent system from its local database and compares them to the agent system's resources. The Waiting Support Service selects only those agents for resumption which meet the agent system's resource requirements (agents are sorted by waiting time) and multicasts a message to them indicating the destination agent system's identification **(3)**. Since the Waiting Support Service monitors the communication events of the suspended mobile agents it detects communication events (issued by itself) for the mobile agents and resumes them **(4)**. After resumption the mobile agents detect the message of the Waiting Support and start migrating to the destination agent system **(5)**.

Those agents which are not allowed to migrate to the destination agent system due to resource constraints need a special handling. We propose that the Waiting Support Service sets a timer for each of these agents. On timer expiration the Waiting Support Service resumes these agents. Note that each agent also has a maximum waiting timer. If this timer expires the agent is resumed irrespective of the availability of the destination agent system.

4.2.2 No Waiting Support Service is available

The agent system on which the mobile agent runs is disconnected and thus the agent must use the local Waiting Support Proxy.

As mentioned before the Proxy has almost the same functionality as the Waiting Support Service, but while the latter resumes agents if the user's current agent system is available again, the Proxy resumes agents if the local agent system reconnects.

4.3 The User Agent

The User Agent forwards the user's current agent system address according to user profile information exchange rules and discovers a suitable Waiting Support Service. Since agents request a Waiting Support Service from the User Agent it is expected that almost all agents are waiting at the same Waiting Support Service. If agents would have discovered services themselves they would probably be distributed over several Waiting Support Services. But having just one Waiting Support Service has two advantages: First, network traffic is reduced as only one registration and notification message is required. Second, the Waiting Support Service can negotiate resources in behalf of the user's current agent system (thus reduces network traffic).

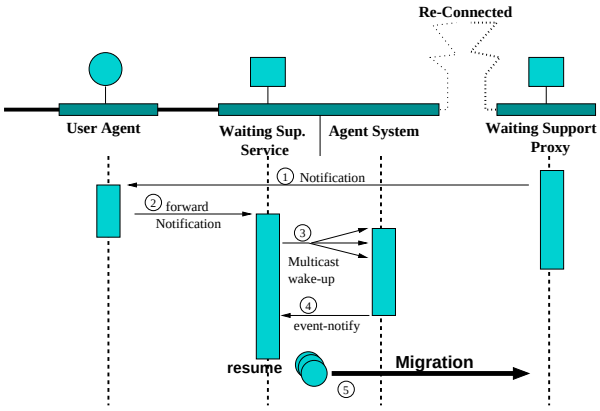

Figure 5. Waiting Support Service is available: Reconnect

5 Performance Considerations

Like every central approach, the User Agent has its weaknesses. If the User Agent fails all applications depending on it will fail too. Furthermore, due to its importance network traffic is increased because all mobile agents of a user are expected to interact with the user's User Agent. It seems that the weaknesses surpass the strengths of our approach, but there is way to mend these problems.

5.1 On mending the flaws of a centralised approach

The User Agents stores those static and dynamic database entries in the LDAP directory (user profile subtree) which are of interest to agent applications. Thus, agent's can retrieve information from both the LDAP directory or the User Agent. In that way the User Agent utilises the replication mechanism of the directory service to distribute information. If the User Agent fails agent applications can still retrieve required information from the LDAP directory. Furthermore, most agent's are expected to interact with the directory and thus traffic is kept local (use replica server).

Still there are agents which must interact with the User Agent (e.g. the Waiting Support Service, user profile modifications). Most of these agents are management services and located close to the user's access point. Implementing the User Agent as a mobile agent which follows the user's movement keeps User Agent interactions local. Although the User Agent might have a huge code and data size, its migration costs are low: Since it is expected that every user has a User Agent its code is well distributed and almost available at each agent system in the fixed network. Thus, it is expected that the agent's code is not transfered to the destination agent system. Furthermore, most entries of the static and dynamic database are stored in the user profile subtree — therefore, static and dynamic database entries are not transfered during agent migration. We conclude that User Agent migration can be performed at considerable costs.

5.2 On analysing the central trader

Apart from network traffic caused by User Agent interactions we must also consider the traffic caused by dynamic database updates.

Basically, the User Agent should trade those service which are frequently used by mobile agents. If the average number of service entry updates is larger than the number of requests, updating the service entry at regular intervals is useless.

There is a need for a intelligent service entry update strategy as some services like the Waiting Support Service are only requested in certain situations. Obviously, it makes no sense to trade Waiting Support Services at regular intervals while the user is logged-in. At that instant the user disconnects the User Agent should discover a Waiting Support Services.

6 Conclusions

In this paper we proposed the User Agent which is an active extension of the user profile. Among others the User Agent acts as the user's central service trader. Therefore, it maintains a dynamic database which holds service references. The dynamic database is constantly updated according to service descriptions and user preferences. Our approach allows frequent modifications of services descriptions and user preferences. These modifications affect all agents including those which are already dispatched and still running.

It is envisaged to implement the agent in the Java Agent Environment (JAE). JAE provides access to a LDAP directory service. To distribute data, the agent's dynamic database is stored in the directory. Mobile agents either retrieve service references from the agent itself or from the directory.

Apart from service trading the User Agent allows maintaining more than one user profile, e.g. private and business profile. The User Agent eases profile management and controls information exchange between different profiles.

Although, we give reasons that the User Agent scales and performs well further performance analysis is re-

quired. To this end we are implementing a simulation model.

References

[1] David Chess, Colin Harrison, and Aaron Kershenbaum, "Mobile agents: Are they a good idea?", Tech. Rep. RC 19887, IBM, October 1994.

[2] Robert S. Gray, David Kotz, Saurab Nog, Daniela Rus, and George Cybenko, "Mobile agents for mobile computing", Tech. Rep. PCS-TR96-285, Dartmouth College, Computer Science, Hanover, USA, May 1996.

[3] Dartmouth College, "D'agents", http://agent.cs.dartmouth.edu/.

[4] Ernoe Kovacs, Klaus Roehrle, Hong-Yong Lach, Bjoern Schiemann, and Carsten Pils, "Agent-based mobile access to information services", in *4th ACTS Mobile Communications Summit*, June 1999, pp. 97–102.

[5] Mitsubishi Electric ITA, Horizon System Laboratory, *Mobile Agent Computing — A White Paper*, January 1998.

[6] "JAE: Java Agent Environment", http://www-i4.informatik.rwth-aachen.de/jae/.

[7] Anthony S. Park, Michael Emmerich, and Daniel Swertz, "Service trading for mobile agents with ldap as service directory", in *IEEE 7th Intl. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE'98*. June 1998, IEEE.

[8] Tom Walsh, Noemi Paciorek, and David Wong, "Security and reliablity in concordia", in *31st Annual Hawai's International Conference on System Science (HICSS31)*, January 1998.

[9] Steffen Lipperts, Anthony S. Park, and Carsten Pils, "Mobile agents at the boundary of fixed and mobile networks", in *2nd International ACTS Workshop on Advanced Services in Fixed and Mobile Telecommunication Networks*, Singapore, November 1999.