Partitioning Applications with Agents

Oskari Koskimies and Kimmo Raatikainen

Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 UNIVERSITY OF HELSINKI
Firstname.Lastname@cs.helsinki.fi
http://www.cs.helsinki.fi/research/monads/

Abstract. The environment of mobile computing is in many respects very different from the environment of the traditional distributed systems of today. Bandwidth, latency and delay may change dramatically when a nomadic end-user moves from one location to another or from one computing environment to another. The variety of terminal devices which nomadic users use to access Internet services also increases at a growing rate.
Dynamic adaptation of a service to the properties of terminal equipment and available communication infrastructure is an attractive feature. With application partitioning, an application consisting of co-operating component agents can be dynamically distributed on both sides of the wireless link. By selecting a partitioning configuration based on terminal characteristics, an application can be adapted to the capabilities of the terminal. Partitioning can also be used for adapting to wireless link quality, by repartitioning the application when link quality changes sufficiently. We have designed a service for performing the partitioning decisions, and used a prototype implementation to prove that the communication delays incurred by repartitioning are acceptable.

1 Introduction

The environment of mobile computing is in many respects very different from the environment of the traditional distributed systems of today. Bandwidth, latency, delay, error rate, interference, interoperability, computing power, quality of display, and other non-functional parameters may change dramatically when a nomadic end-user moves from one location to another, or from one computing environment to another – for example from a wired LAN via a wireless LAN[3] (WLAN) to a GPRS[11] or UMTS[17] network. The variety of mobile workstations, handheld devices, and smart phones, which nomadic users use to access Internet services, increases at a growing rate. The CPU power, the quality of display, the amount of memory, software (e.g. operating system, applications), hardware configuration (e.g. printers, CDs), among other things ranges from a very low performance equipment (e.g. hand held organizer, PDA) up to very high performance laptop PCs. All these cause new demands for adaptability of data services. For example, palmtop PCs cannot properly display high quality images designed to be looked at on high resolution displays, and as nomadic users will be charged based on the amount of data transmitted over the GPRS network, they will have to pay for bits that are totally useless for them.

Software agent technology has gained a lot of interest in the recent years. It is widely regarded as a promising tool that may solve many current problems met in mobile distributed systems. However, agent technology has not yet been extensively studied in the context of nomadic users, which exhibits a unique problem space. The nomadic end-user would benefit from having the following functionality provided by the infrastructure: Information about expected performance provided by agents, intelligent agents controlling the transfer operations, a condition-based control policy, capability provided by intelligent agents to work in a disconnected mode, advanced error recovery methods, and adaptability.

The research project Monads [10] examines adaptation agents for nomadic users [15]. In the project we have designed a software architecture based on agents and we are currently implementing its prototypes. Our goal is not to develop a new agent system; instead, we are extending existing systems with mobility-oriented features. The Monads architecture is based on the Mowgli communications architecture [14] that takes care of data transmission issues in wireless environments. In addition, we have made use of existing solutions, such as FIPA specifications [5] and Java RMI [16], as far as possible. However, direct use was not sufficient but enhancements for wireless environments were necessary [2, 13].

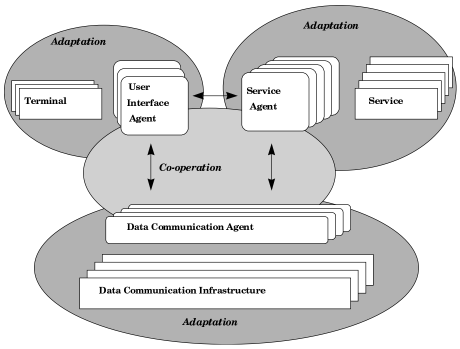

Fig. 1. The Adaptation Triad

By adaptability we primarily mean the ways in which services adapt themselves to properties of terminal equipment and to characteristics of communications. This involves both mobile and intelligent agents as well as learning and predicting temporary changes in the available Quality-of-Service (QoS) along the communications paths. The fundamental challenge in nomadic computing is dynamic adaptation in the triad service–terminal–connectivity (see Figure 1) according to preferences of the end-user.

The ability to automatically adjust to changes in the wireless environment in a transparent and integrated fashion is essential for nomadicity – nomadic end-users are usually professionals in other areas than computing. Furthermore, today's distributed systems are already very complex to use as a productive tool; thus, nomadic end-users need all the support, which an agent based distributed system could deliver. Adaptability to the changes in the environment of nomadic end-users is the key issue. Intelligent agents could play a significant role in implementing adaptability. One agent alone is not always able to make the decision how to adapt, and therefore adaptation is a co-operation effort carried out by several agents. Thus, there should be at least some level of cooperation between adapting agents.

Dynamic adaptation of a service to the properties of terminal equipment and available communication infrastructure is an attractive feature. We have previously explored predictive adaptation to available bandwidth with a Web browsing agent [15]: When the network connection is slow or predicted to become slow, the browser agent may automatically use different kinds of compression methods or even refuse to fetch certain objects.

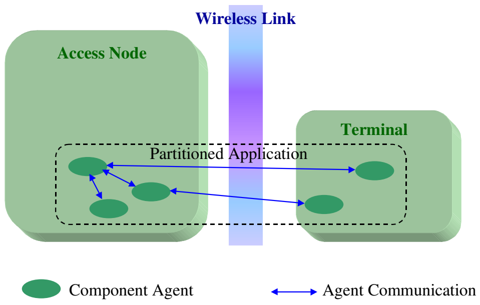

Fig. 2. Application partitioning

We are now also examining adaptation to terminal equipment through *application partitioning*. With application partitioning we refer to the idea of dividing an application into component agents that communicate using e.g. FIPA ACL[7] messages. In this way, the application is running in a distributed fashion on both sides of the wireless link, as depicted in Figure 2.

Partitioning can be either *static*, *partially dynamic* or *fully dynamic*. *Static* partitioning, where component agent configuration is determined at compile time and cannot be changed, is of little interest to us. In *partially dynamic* partitioning, the location of a component agent is determined dynamically during application initialization, but cannot change during the application session. The most interesting is *fully dynamic* partitioning, which allows the component agents to be moved at any time during the application session. This is useful when bandwidth (or other dynamic resources such as memory) changes radically, and the change is expected to last for some time. For bandwidth, such a drastic change would typically be a vertical handover or a disconnection.

Partially dynamic partitioning is sufficient for adapting to different terminal types, but adapting to bandwidth changes (including disconnections) requires fully dynamic partitioning. Note that fully dynamic partitioning requires the component agents to be mobile, whereas partially dynamic partitioning does not.

2 The Need for Terminal Adaptation

The QoS of a wireless link can vary wildly, due to interference, network load and vertical handovers. In some cases, like vertical handover, these changes can be quite drastic. In order to provide smooth operation, an application needs to adapt to changes in QoS. Traditionally, the adaptation is done by compressing and reducing content that is transferred to the terminal.

However, adapting to just QoS is not enough. Unlike the users of fixed networks, who almost all have fairly similar, workstation-class terminals, the terminals of nomadic users vary from smartphones to powerful laptops. Even the same user is likely to use different classes of terminals, e.g. a laptop on business trips and a PDA or smartphone during his free time. However, currently the nomadic user often has to use a different application for the same purpose on different classes of terminals. For example, the calendar software on a laptop is likely to be very sophisticated and offer an advanced graphical user interface, whereas the software on a smartphone would probably offer only minimal functionality. Even though the different applications may be able to exchange information, the situation is still undesirable for two reasons:

1. An application is limited to the terminal class it was designed for, and
2. Users have to learn a different application for each terminal class.

The application can, of course, be implemented separately for each terminal class, but this wastes implementation effort, and a version for smaller terminals may have drastically reduced functionality. What is required is an application that can adapt to different terminal classes without sacrificing functionality[1].

3 Adaptation by Partitioning

We are examining adaptation to terminal equipment and QoS variation by partitioning an application into components. Because adaptation of the overall application requires that the individual components can cope with varying component configurations, the components must both be themselves adaptive, and also have a degree of autonomy. Thus, it makes obvious sense to model the components as agents. To give an example: An email application can work with different types of terminals by partitioning the application in different ways, depending on terminal and wireless link characteristics. Assume the email application consists of four agents:

1. *User interface agent*: Either a standard FIPA UDMA agent [8] or a dedicated UI agent that has an advanced graphical user interface and supports voice input.
2. *Core email agent*: Handles basic email processing.
3. *Email filtering agent*: Groups and prioritizes messages, and handles any automated email processing.
4. *Email compression agent*: Compresses messages prior to transferring them to the terminal (this includes lossy methods such as leaving out attachments).

[1] However, *usability* may suffer. No amount of adaptation can make the keyboard or screen of a smartphone match that of a laptop.

With low-performance equipment (a high-end mobile phone, for example), only the user interface (a standard FIPA UDMA agent) of the application runs on the terminal. If the terminal has more capabilities (like a PDA), also the core email agent can run on the terminal, with the email filtering and compression agents running on the network side. Finally, with a laptop terminal, all agents except the compression agent can be run on the terminal.

Partitioning can be used for adapting to available bandwidth as well. For example, if bandwidth is very low, it is better to run the filtering agent on the network side, since it may be able to handle some emails automatically, and prioritizing means the user will get the more important messages first. On the other hand, if bandwidth is high, it is better to run the filtering agent on the terminal, since that allows it to more easily communicate with the user about filtering decisions, enabling more fine-grained filtering control. By using mobile agents, this kind of adaptation can be dynamic, with agents moving to optimal locations while the application is running. We call this *repartitioning*.

Another facet of partitioning is the possibility to use different agents for different terminals, or not to use a particular agent at all. For example, with a high-performance laptop, a large, dedicated email UI agent can be used instead of a small, generic FIPA UDMA agent. Or, when bandwidth is high, the use of an email compression agent becomes unnecessary.

3.1 Assumptions

The design of our partitioning system is based on a few assumptions:

Applications are specially designed We assume that the application is specially designed to support partitioning, for the following reasons:

1. An application that wasn't designed to be partitioned is unlikely to be easily divided into separate independent components.
2. Partitioning, and especially repartitioning, requires the component agents to be designed flexible so that reconfiguration is possible. It is unrealistic to expect this from applications in general, although applications built by component composition (from "off-the-shelf" component agents) in the future might have the required flexibility.
3. Practical agent applications are still scarce, so one can assume that the most important agent applications are still to be built, and their design is thus still open.

Application metadata is available Since applications are already assumed to be specially designed for partitioning, it is not unreasonable to assume that some metadata (such as agent resource requirements) has been made available as well.

Repartitioning is heavy and uncommon The actual time taken by any single repartitioning operation depends on available bandwidth and the size of the agents involved, but we assume that repartitioning is a heavy operation, due to both its transactional nature and the need to transfer agent state. Thus, repartitioning would be worthwhile only when drastic changes in circumstances occur (e.g. vertical handover).

Note that the initialization of a partitioned application is a much lighter operation, but may still require application code to be transferred over the wireless link.

3.2 How to Partition an Application

Our project has previously produced a system for predicting near-future QoS fluctuations[15]. We utilize these predictions, along with profiles, to make partitioning decisions.

Three types of profiles are used:

Application profiles The Application profile lists the component agents in the application, and a set of possible configurations for them. For each configuration, agent locations and a communication profile are given, as well as an *utility* value that represents how "good" (in terms of usability) that configuration is for the user. If the application is willing to share some agents with other applications, that information would be here also.

Note that alternative (such as different GUI agents) and optional (such as compression agents) agents can be represented by configurations where all agents are not included.

Agent profiles Agent profiles contain resource requirements for the agent, as well as startup cost and repartitioning (movement) costs. Additionally, there are three flags:

- *External Agent*: The agent does not understand partitioning messages. It can still be used in a partitioned application, but the other agents have the responsibility for ensuring that its state is preserved across repartitioning. From the point of view of the partitioning service this flag simply means that the agent will be excluded from partitioning processes except for telling it to move to a new location or to shut down.
- *Movable*: The agent is mobile. If it is an external agent, its movement must be accomplished through normal agent management operations.
- *Replaceable*: The agent can be replaced by another agent. This requires that the agent is either stateless or saves its state to other agents before repartitioning.

Note that an agent might be neither movable nor replaceable. For these agents, relocation is either not possible or not desirable. Repartitionings that would affect these agents are not possible.

Terminal profiles The capabilities of terminals, such as memory and screen size, are listed in terminal profiles.

Figure 3 illustrates how a partitioning decision is made. When an application is started, the partitioning service first loads the application profile. The list of agents is then used to get the resource requirement profiles for the agents. These requirements are compared against the terminal capability profile to get a list of possible configurations for this terminal, and the communication profiles for those configurations.

In simple terms, for each possible application configuration, its bandwidth use (from the communication profile) is added to the bandwidth use of currently running applications, to get combined bandwidth use. This is then deducted from the predicted total

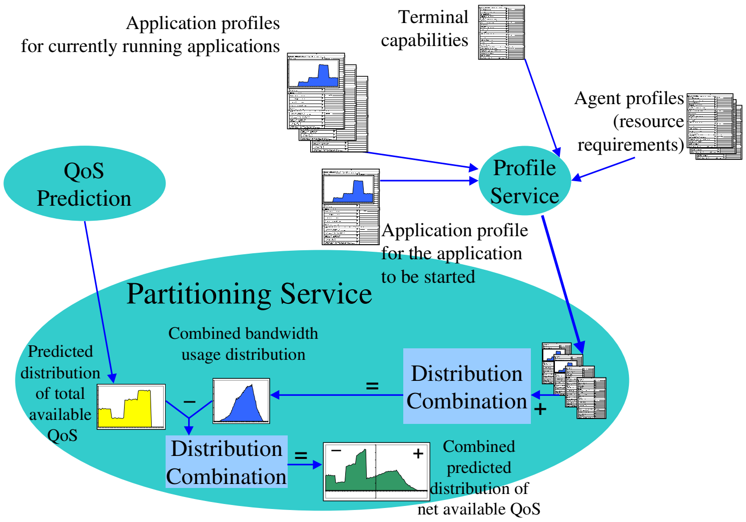

Fig. 3. Partitioning decision

available bandwidth, to get the predicted net available bandwidth. This is the prediction of the remaining bandwidth after the application is started.

The reality is only slightly more complex: A prediction of available bandwidth is not a single value but a distribution that gives the probability for getting a specific bandwidth. Likewise, the bandwidth use of an application varies according to a probability distribution. Thus, the result of the above calculation is also a distribution.

If the resulting distribution gives a high probability for having a negative net (remaining) bandwidth, that means that the configuration that is being examined cannot be run with the available bandwidth. Otherwise, the resulting distribution is given an *utility score* that is based on how much bandwidth was taken by it, and the utility value of that configuration for the user (available from the application profile). The score is penalized if the configuration exhausts available resources on the terminal. The configuration with the best final score is then selected.

3.3 Starting the Application

The application startup sequence is shown in Figure 4. The original startup request (1) is directed to the partitioning service. First, the partitioning service selects a unique ID for the application session. Then it queries the yellow pages (e.g. a FIPA DF[6]) in the terminal and on the network side (2) to see if some agents are already running and previously registered (2b), and then issues create-agent requests for those agents that are not running (or cannot be shared) (3).

To start an agent, the partitioning service may use a Factory service that advertises itself via the yellow pages, or it can contact the agent platform directly (e.g. a FIPA

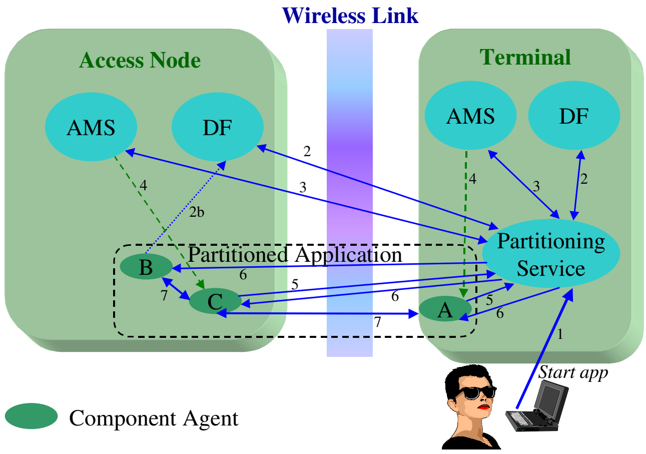

Fig. 4. Application startup

AMS[6]). Either way, a new agent is created (4). If agent profiles contain a startup cost, the fact that an agent is running can also be taken into account when selecting configurations. When an agent has successfully initialized itself, it sends a message (5) to the partitioning service to confirm that it is ready.

Finally, the application session ID is sent to all participating agents, together with information about their partners (6), binding them together to form the application. The application session ID is especially important for shared agents, who use it to manage their state information. Application communication can now start (7). In fully dynamic partitioning, this phase can be redone during repartitioning to remove the need for rerouting messages of moved agents.

3.4 Repartitioning

The weakness of partially dynamic partitioning is that while the partitioning decision depends on both terminal capabilities and the QoS of the wireless connection, only terminal capabilities are relatively static. QoS may change drastically during the lifetime of an application session, making a previously made partitioning decision invalid. Fully dynamic partitioning becomes necessary: The application must be *repartitioned* by moving its component agents to new locations. Thus, mobile agents are required.

In fully dynamic partitioning, the decisionmaking process is rerun when QoS is predicted to change drastically. Frequent reruns of the process can be avoided by ignoring smaller changes, or by only considering vertical handoffs and disconnections.

If a rerun of the decisionmaking process shows that another configuration is superior to the current one, the difference of the utility scores of the configurations (the profit), multiplied with the time the new conditions are predicted to last, is compared to a penalty calculated from the repartitioning costs of the agents. If the profit is greater, repartitioning is initialized.

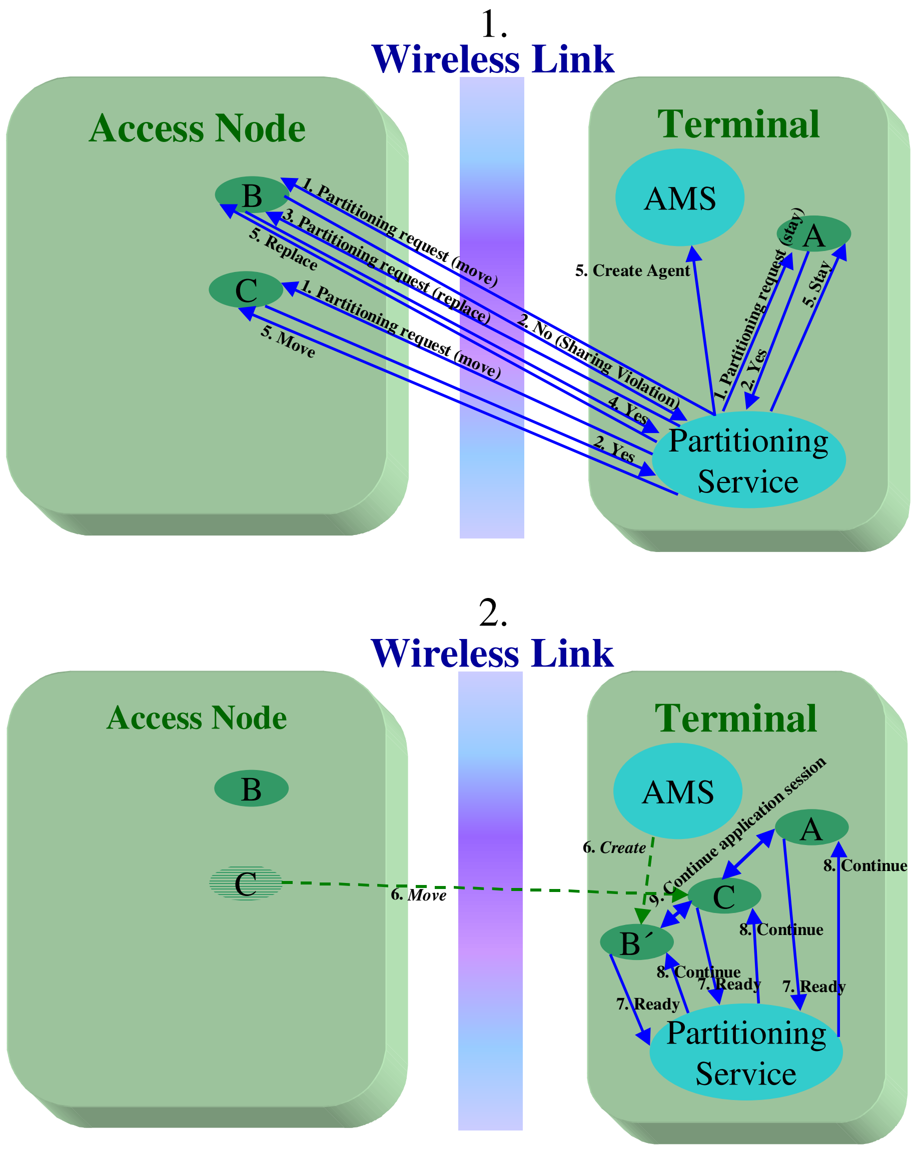

Fig. 5. Repartitioning

The actual repartitioning process, shown in Figure 5, is somewhat like a two-phase commit (see e.g. [9]). First, the partitioning service sends each component agent a partitioning request that contains the application session ID and the agent's orders (to stay, move, be replaced or shut down), and asks if the agent is able to participate. The agent checks if it can free itself of any application state that cannot be carried over the repartitioning process, and sends a `yes` answer if it can. Once the answer has been sent, the agent enters a state where it waits for the partitioning process to complete, refusing requests not related to the partitioning process. Note that even though an agent answers `no`, it should not continue normal operation, since other agents in the application may not be ready to continue yet.

Error handling follows the normal two-phase commit procedure, with one exception: An agent may be unable to move because it is shared by other applications. In that case, it sends back a `no` answer, and additionally indicates that the refusal is due to a sharing violation. The partitioning service may then immediately (without aborting) resend the partitioning request to the agent, but with replacement orders instead of movement orders. Since the additional message exchange takes time, service-type shared agents should preferrably be marked as non-movable in the agent profile.

If the partitioning service gets a `yes` answer from all agents, it then sends each agent a message which tells them what to do. Once an agent has completed the order, it sends back a reply to the partitioning service to signal that the order has been successfully executed. If the agent platform does not support adequate message rerouting after agent movement, the reply may contain the new messaging address of the agent.

If the new configuration requires agents that were not present in the old one, these are created the same way as in application startup, described in the previous section. If an agent in the old configuration is not present in the new one, its order will be to shut down, and it will send the reply just before shutting down.

Finally, once all agents have reported in, the partitioning service sends each agent a `continue` message to tell them that the repartitioning has completed, and informs them of changed messaging addresses and new, removed or replaced agents. The agents can now continue from where they left off. Note that in some cases this is not straightforward; for example, a simple user interface agent may have been replaced with a more complex one, or some agents may not be present in the new configuration. However, it is up to the agents themselves to adapt to the new situation.

3.5 Repartitioning and Personal Mobility

As shown by Figure 6, repartitioning can also be used for personal mobility. This requires only minor extensions to the system described above. Firstly, the partitioning service on terminal A must fetch the terminal profile for terminal B, and use it instead of the terminal A profile. Secondly, all agents that the repartitioning process would have placed on terminal A, are issued orders to move to terminal B. Agents that are specific to a terminal type may be replaced by others, but that is a normal part of the partitioning process. Finally, the partitioning service on terminal A must pass on the responsibility for the application to the partitioning service on terminal B.

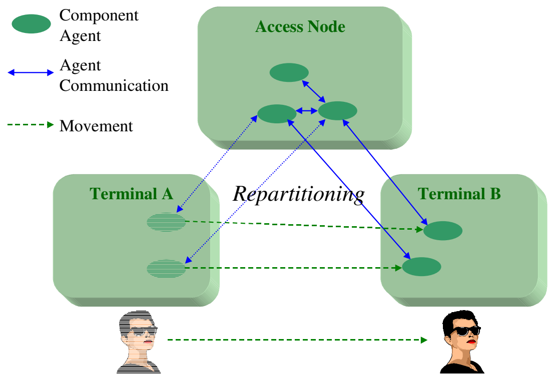

Fig. 6. Personal Mobility

3.6 An Example Scenario

To give an idea of the communication costs involved in partitioning, we have built a prototype of the partitioning system, using the JADE 1.4 [1, 4] agent platform[2]. It was tested with a test application consisting of four agents, mimicing the email application example. Each agent had only the functionality necessary for the partitioning process, and some dummy state data.

Table 1. Example Agents

Agent	Description	State Information	Footprint
UDMA	Generic user interface agent	10 kB	100 kB [3]
DGUI	Dedicated UI	10 kB	1 MB
Core	Core Email Agent	100 kB	1 MB
Filter	Email Filtering Agent	20 kB	200 kB
Compr	Email Compression Agent	–	200 kB

The agents, their state data and memory requirements (as contained by the profiles) are described in Table 1. The test scenario is outlined in Figure 7:

1. The user is in her office, using a desktop computer with a 100 Mbps LAN connection. She starts reading her mail. The partitioning system chooses a configuration with a dedicated UI agent and without an email compression agent, where everything is running on the terminal, and starts the email application.

[2] The reason for using Jade is that it is also otherwise used by our project. However, JADE's implementation of agent mobility is not optimized, so using it also has the benefit of getting conservative results.

[3] The generic user interface agent can be shared by other applications.

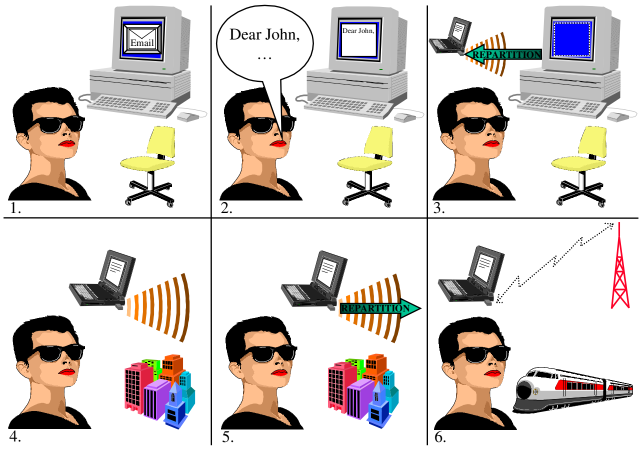

Fig. 7. Example Scenario

2. The user reads her mail. After a while, she comes across one that requires a reply, and switches to voice input, dictating the reply message.
3. After dictating about half of the reply, she notices it is time to go home, and switches to her palmtop, which has a 2 Mbps WLAN connection in the office, and a 28.8 kbps GSM High Speed Data connection[12] while outside[4]. The partitioning system moves the email application to the palmtop, using now a configuration with a generic user interface agent. The partial reply is saved to the state of the core email agent, and moved with it.
4. The user finishes writing the reply while waiting for the elevator. Because a generic UI agent is used, the GUI of the email application is now less polished, and no longer accepts voice input, but the user can continue to write the reply using normal text input.
5. Based on the time, and the fact that the user left the office, the QoS prediction system[15] gives a high probability that bandwidth will soon drop dramatically. The partitioning system recalculates the optimal configuration. The resulting configuration has an email compression agent and the email filter agent on the network side, and the other agents in the terminal.
6. The user continues to read her email using this configuration while sitting in a train. Although the user interface of the email application is more bare (generic UI), attached images lose some detail or are omitted (compression), and messages from mailing lists are ignored (filtering), she is still using the same application, and even the same application session, as when she started reading her email.

The scenario was tested with our prototype. The test was run using a Pentium III Linux workstation as the access node, a Pentium II Linux workstation as the desktop terminal and a Pentium Linux laptop as the palmtop computer. A real WLAN was used,

[4] All these connection types are already commercially available.

but the GSM data link was simulated with software. The results are given in Table 2. Note that the times are median values from five test runs.

Table 2. Test Results

Event	Bandwidth	State data	Time
Application start (1)	100 Mbps	–	0.7 s
Terminal change repartitioning (3)	2 Mbps	120 kB	4.3 s
QoS change repartitioning (5)	2 Mbps	20 kB	2.2 s

As can be seen, the communication delays are quite acceptable for this scenario. On the last partitioning, however, that is due to the successful QoS prediction that allowed repartitioning to be initiated while bandwidth was still high. If the repartitioning had been done *after* the drop in QoS, when bandwidth was down to 28.8 kbps, the repartitioning would have taken 14 seconds – still acceptable, but a very noticeable delay. Note that application classes were already present on access node and terminals.

4 Conclusions and Future Work

We have shown how partitioning can be used to adapt an application to varying QoS and terminal capabilities. Partitioning can be used for implementing personal mobility, as well. Our first tests indicate that the solution is feasible as far as communication costs are concerned.

Our next step will be to fully implement the partitioning system and use it in the Monads QoS prediction system, so that the prediction system can be optimally configured for different terminal types. The problem of how to minimize state loss during repartitioning is also worthy of attention. Finally, partitioning-related messaging must be optimized to reduce the partitioning overhead.

References

1. F. Bellifemine, G. Rimassa, and A. Poggi. JADE – A FIPA-compliant Agent Framework. In *Proceedings of the Fourth International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAM 1999)*, April 1999.
2. S. Campadello, H. Helin, O. Koskimies, and K. Raatikainen. Performance Enhancing Proxies for Java2 RMI over Slow Wireless Links. In *Proceedings of the Second International Conference and Exhibition on the Practical Applicaiton of Java (PA Java 2000)*, April 2000.
3. B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai. IEEE 802.11 Wireless Local Area Networks. *IEEE Comm. Mag.*, pages 116–126, September 1997.
4. Jade Web Site. Available electronically from `http://sharon.cselt.it/projects/jade/`.
5. Foundation for Intelligent Physical Agents. FIPA Web Site. Available electronically from `http://www.fipa.org/`.
6. Foundation for Intelligent Physical Agents. FIPA 97 Specification Part 1: Agent Management, October 1997. Available electronically from `http://www.fipa.org/`.

7. Foundation for Intelligent Physical Agents. FIPA 97 Specification Part 2: Agent Communication Language, November 1997. Available electronically from `http://www.fipa.org/`.
8. Foundation for Intelligent Physical Agents. FIPA 98 Specification: Human-Agent Interaction, 1998. Available electronically from `http://www.fipa.org/`.
9. A. Goscinski. *Distributed Operating Systems: The Logical Design*, chapter 5.4.8, pages 203–204. Addison-Wesley, 1991.
10. Monads Research Group. Monads Web Site. Available electronically from `http://www.cs.helsinki.fi/research/monads/`.
11. GSM Technical Specification, GSM 02.60. GPRS Service Description, Stage 1, 1998. Version 6.1.0.
12. GSM Technical Specification, GSM 03.34. High Speed Circuit Swithed Data (HSCSD), Stage 2, May 1999. Version 5.2.0.
13. H. Helin, H. Laamanen, and K. Raatikainen. Mobile Agent Communication in Wireless Networks. In *Proceedings of the European Wireless'99 Conference*, October 1999.
14. M. Kojo, K. Raatikainen, M.Liljeberg, J.Kiiskinen, and T.Alanko. An Efficient Transport Service for Slow Wireless Telephone Links. *IEEE Journal on Selected Areas in Communications*, 15(7):1337–1348, September 1997.
15. M. Mäkelä, O. Koskimies, P. Misikangas, and K. Raatikainen. Adaptability for Seamless Roaming Using Software Agents. In *XIII International Symposium on Services and Local Access (ISSLS2000)*, Stockholm, Sweden, June 2000.
16. Sun Microsystems. Java Remote Method Invocation – Distributed Computing for Java. White Paper, 1998.
17. Third Generation Partnership Project Web Site. Available electronically from `http://www.3gpp.org/`.