ADAPTABILITY FOR SEAMLESS ROAMING USING SOFTWARE AGENTS

Mikko Mäkelä, Research Assistant
Oskari Koskimies, Project Manager
Pauli Misikangas, Researcher
Kimmo Raatikainen, Professor

Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23), FIN-00014 UNIVERSITY OF HELSINKI, Finland

Introduction

Software agent technology is a new computing paradigm. It is claimed that it solves many of the current problems in distributed mobile computing. In the research project Monads at the Helsinki University Computer Science Department we have researched the possibility to use agent technology to support nomadic end-users and successfully applied agents for nomadic web browsing. According to Prof. Leonard Kleinrock [1], "Nomadic computing and communications will dramatically change the way people access information—and a paradigm shift in thinking about applications of the technologies involved. It exploits the advanced technologies of wireless, the Internet, global positioning systems, portable and distributed computing to provide anytime, anywhere access. It is beginning to happen, but it makes everything much harder for the vendors."

A software agent is in general a software entity that performs actions for its owner. Agents are often described as being autonomous, goal-oriented and having social abilities to communicate with other agents [2, 3].

A nomadic application is an (distributed) application, which serves, as well as possible, end-users while they are on the move, at an intermediate stop, and at an arbitrary destination. In a nomadic environment, the end-users typically have use of mobile computers, PDAs, and intelligent phones as their terminal devices and different kinds of wireless communication means—WAP [4], GSM Data Service [5, 6], GPRS [7], UMTS [8], for example—to access the services in the fixed network.

In the strict form, seamless roaming is an action where a mobile terminal moves from one area of network coverage to another, so that the handoff and change of administrative domain is automatic and transparent to distributed applications. However, changes in the Quality-of-Service (QoS) of data communication are not usually transparent to distributed applications. Typically, there is a significant change is QoS when the underlying network technology changes; for example, when the mobile terminal moves from a GSM network operated by one operator to a UMTS network operated by another operator or when the mobile terminal moves from a GPRS network to a wireless local area network. Therefore, we use a looser form of seamless roaming, meaning that a distributed application is able to serve an end-user satisfactorily even if the underlying communications technology changes.

The ability to automatically adjust to changes in a wireless communication environment in a transparent and integrated fashion is an essential property since nomadic end-users are

usually professionals in other areas than computing. Intelligent agents could play a significant role in implementing adaptability.

The rest of the paper is organized as follows. In the next section we elaborate the role of adaptability in seamless roaming. In the following section, we discuss the modeling of user behavior and available communication capabilities, two important factors for seamless roaming. Before reporting our experiment results from a Web-browsing application, we discuss the software agents in Monads and how they co-operate.

Adaptability for Seamless Roaming

The research problem of seamless roaming can be described as how to adapt to changes, which affect the nomadic user's view of smooth operation. There are many sources of such changes. The underlying communication technology may change suddenly, as for example due to a network change, the changes in physical environment may affect the QoS or terminal characteristics may change and so on. Actions used for adaptation for these changes include prioritization, compression, prefetching and many kinds of preparations like negotiations concerning future node change.

Adaptability via Predictions

When the changes in question happen unexpectedly, implementing seamless roaming can be an impossible task, as many times the actions listed above are effective only, if performed well before the change. If for example the user, while watching a video-stream via his laptop, suddenly unplugs the laptop from the fixed network, it might be too late to adapt the video-stream to a higher compression rate for the reduced bandwidth without dropping the connection.

In many cases, however, situations do not come completely unexpectedly, but they can be, at least roughly, predicted beforehand. For example, the user could have a habit of switching from fixed to wireless network during certain time-periods. If we have gathered information about his habits (e.g. by observing the network conditions of the laptop), we can estimate the probability that he will do it also at a given occasion.

Making Good Decisions

The probabilistic nature of predictions is important because it helps make good (decision theoretically sound) decisions. For example, the estimation that the user has around 10% probability of changing the network may not be enough to justify the preparations involved in it, especially, if there are other, more essential tasks to carry out. On the other hand, direct input from the user suggesting such a change should surely map near to 100% assurance and thus start the preparations immediately.

The probabilities of future communications characteristics by themselves are, of course, not enough for many decisions – we also have to estimate, how important the task we are considering is as compared to other tasks, which are competing for the same resources.

In the next section we take a look at two important prediction classes for seamless roaming.

Modeling User Behavior and the QoS

We cannot base our system on an assumption that the user would always inform the system well before he is going to change the communication media, where he is going to go, what he sees as the most important tasks in the near future and so on. The design of the user-interface is very important, but the dilemma is that, on the one hand, we would like to get as much feedback as we can, while on the other hand, we should try to minimize the need for the user to give it. This means that we must try to model the user's behavior, learn his habits. In fact, we can then take advantage of this learning also in the user-interface, using such techniques as anticipation [9].

There are various things we can learn about the user: when does he arrive at and leave his office, the routes he uses when moving between

places, his daily routines with the computer, and so on. In other words, we are interested in modeling anything that happens regularly, can be observed with the computing equipment in use, and affects the available QoS or required bandwidth.

For example, we can learn the regular routes of the user and the QoS on them. The quality of a wireless connection such as GPRS depends heavily on location, so by predicting the future location we can also predict the QoS. Moreover, the whole communication media may vary according to the location. On the way to work, the user may be using a GSM connection, but as soon as he arrives at the office, the terminal is plugged into a fixed local area network. While moving inside the building, the user switches to a wireless local area network. Thus, if we know the destination and the arrival time, we can predict a change in the communication media and prepare for it in advance.

Obviously, the time of the day must also be taken into account. Most users have a fixed time schedule for working days, which makes predicting certain events trivial. For example, every day at 9 am Joe comes to work and plugs his terminal to the LAN, and at 5 pm he unplugs it and goes home. Predicting a sudden drop in the QoS in the afternoon is quite straightforward in these circumstances. For predicting more irregular events, a calendar agent may be of great help. If the user has an appointment on the other side of town in an hour, it is very likely that the LAN connection will soon be lost, for example.

Software Agents in Monads

In the previous section, we have outlined what kind of adaptation is needed for seamless roaming. The principal idea has been that nomadic applications are offered information about the future quality of the connection, and they are supposed to adjust their behavior to meet the forthcoming situation. In this section, we will describe how predicting the link quality can be implemented in practice.

Taking the Practical Approach

Our approach is based on software agents, to allow a flexible, scalable and fairly fault tolerant system design.

There are numerous definitions for software agents (see [10] for good coverage) and many of them set quite burdensome requirements for a software entity to be called an agent. Our approach has been practical: we see a software agent essentially as a design tool. This means that if we think a certain task in hand is best performed with an agent, we call the implemented software component an agent from the start, without considerations about how simple its implementation might be.

The technical requirements for our agents have elaborated their autonomy: Monads agents run each on their own thread and are able to communicate asynchronically, in timely fashion. While every agent by itself does not necessarily meet many of the definitions for agents, we believe that different combinations of Monads agents do – we just like to keep individual agents as simple as possible and have more complex features emerging from the agent co-operation.

Especially, we make heavy use of a special kind of agents we have named Modeler Agents. They are described in the next subsection. In the following subsections, we show how agents co-operate to provide the desired functionality and give an example of co-operation.

Modeler Agents

A learning (or adaptive) agent is usually defined as an agent, which changes its behavior based on its previous experience. While agents act, they learn various things about the environment, the effect of their actions, and the behavior of other agents.

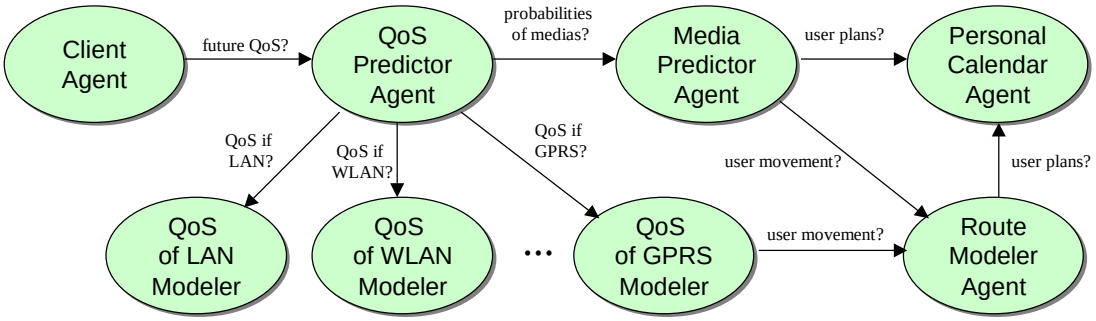

Figure 1: Agents involved in QoS predictions

In Monads, we have taken a somewhat different view of learning with agents. The reason for this is obvious, most of the agent systems and theories involving learning are still in the very early stages while we want to apply learning agents in practice now. Therefore, we use the combinations of intelligent agents that have expert knowledge of the task and modeler agents that bring learning capabilities to the system.

Modeler agents' primary mission is to learn, but they do not use the results of learning by themselves. They just collect data and build models of the phenomenon they are trying to learn and are used as assistants to intelligent agents. They are created by the Monads Learning Service to solve a specific learning task defined by the client agent. During startup, the modeler agent is told the type of learning task (at the moment we support classification and sequence learning problems), which learning algorithm to use, and how the data can be obtained. After this, the modeler starts collecting data and building the model, and is ready to answer questions made by the client agent.

The modeler agent does not need to understand the semantics of the symbols and values it handles. The learning is purely technical—just following the steps of the learning algorithm. However, the client agent the modeler is working for must understand the meaning of the input and output values of the learning task. So, learning does not affect the behavior of the modeler agent directly, but it should have influence on the decisions made by the client agent. We can say that the client agent is the "intelligent" one, and the modeler is just its dumb slave that is good at only one thing: learning. The client may act as a teacher, giving feedback about the correctness of the modeler's output. For example, if the modeler makes an incorrect classification, the client may "penalize" it and possibly also tell the correct answer.

Modeler agents solving the same learning task (possibly on different agencies) may also share data and models to help each other. Whether this is worth doing or not depends on the learning task. Sharing data/models may be essential for some tasks, but may do more harm than good for others. Therefore, the one who defines the learning task must carefully consider if sharing should be enabled. We avoid many of the problems of the traditional knowledge sharing approach (see for example [11]), because the modelers are all identical and created by the same Learning Service. Thus, they have no problem understanding each other. We do not need any special knowledge representation language since we can transfer the models as data structures by serializing the Java object representing the model.

Co-operation of Agents

Figure 1 shows the agents that implement the QoS prediction services needed by nomadic applications. Each agent has a very specific purpose and the overall functionality is a result of co-operation between the agents. Although each agent is rather simple, by co-operating they can reach an overall behavior that may seem intelligent to the user.

The QoS Predictor Agent provides short-term predictions about future QoS for client agents. It can answer the following kind of questions:

- How much data at least can be transferred in time *t* with *p%* certainty?
- How much time at least is needed to transfer *x* kilobytes of data with *p%* certainty?
- What is the probability that *x* kilobytes of data can be transferred in time *t*?

The overall QoS prediction is a combination of predictions provided by several QoS Modeler Agents. Each of them models and predicts the QoS for one network technology. For example, the QoS of WLAN Modeler Agent models the QoS of a wireless local area network. QoS Modeler Agents may need help from other agents, as is the case with QoS of the GPRS Modeler Agent. Since the quality of a GPRS connection depends on the location of the terminal, the modeler needs information about the terminal/user movement, which is provided by the Route Modeler Agent.

The Route Modeler learns the regular routes of the user and is capable of predicting user movement. Predictions of future routes are achieved by examining the user's movement history (see [12] for details). However, sometimes the exact destination could be read from the user's calendar. Therefore, the Route Modeler co-operates with the Personal Calendar Agent. This is especially useful when the user movement cannot be perceived by the system.

The Calendar Agent may be of great help also to the Media Predictor Agent, whose job it is to predict which communication media will be in use in the near future. Besides the calendar, movement predictions and a user behavior model are also used for estimating probabilities for each media. The probabilities are used by the QoS Predictor Agent to combine the predictions of QoS modelers as described in the next section.

An Example of Agent Co-operation

Let us consider the following example. Joe has just left his home on a Monday morning and is currently on a train going to the city. On the train, he starts reading his e-mail with a small laptop computer using a GPRS connection. Joe has 15 new e-mail messages, three of which include large files as attachments, each file being about 10 megabytes. Transferring the attachments via GPRS would take approximately 9 minutes per file. Should the system fetch the files or not?

Joe has traveled this route several times before. According to those experiences, Joe's Route Modeler Agent can predict that with 80% probability Joe will switch from train to bus[1] in 7 minutes and be in his office in 13 minutes. The other possibility (with 20% probability) is that Joe continues with the train and arrives downtown in 25 minutes.

Using Joe's behavior model, the Media Predictor Agent can foresee that if Joe goes to his office, the terminal will be connected to LAN with 60% probability and to WLAN with 40% probability. However, if Joe goes downtown, he will certainly continue using GPRS. Thus, we can calculate the overall probabilities for different network technologies by taking the movement prediction into account. In all cases, GPRS will be used for the next 13 minutes. After that, the probabilities for LAN, WLAN, and GPRS are 48%, 32%, and 20%.

The QoS of the GPRS Modeler can tell that the quality of a GPRS connection is fairly good on the way to the office, but becomes worse while nearing downtown, being almost non-existent there at this hour. The speeds of WLAN and LAN connections are also relatively low at the moment, due to the heavy load every Monday morning, but they are still 20-400 times faster than GPRS. Table 1 shows the predicted throughput on the possible routes (named after the final connection type), and the total amount

[1] In practice, the Route Modeler Agent does not know about the vehicles Joe is using, but notices a change in speed and direction of movement.

of data that can be transferred during intervals 0-13 min, 13-20 min, and 20-30 min. The required certainty was 80%, so the probability for the real throughput being less than this is 20%.

Route	Prob.	t∈ [0,13[	t∈ [13,20[	T∈ [20,30[
LAN	48%	150 kbit/s 14.3 Mb	5 Mbit/s 262.5 Mb	6 Mbit/s 450 Mb
WLAN	32%	150 kbit/s 14.3 Mb	1 Mbit/s 52.5 Mb	1.5 Mbit/s 112.5 Mb
GPRS	20%	100 kbit/s 9.5 Mb	45 kbit/s 2.3 Mb	15 kbit/s 1.1 Mb

Table 1: Throughput on Possible Routes

Now, Joe's e-mail agent should make the decision whether to send the attachments and the right moment to do it. Therefore, it asks the QoS Predictor Agent how long it would take to transfer 30 megabytes with 80% certainty. First, the QoS Predictor Agent calculates the answer to this question separately for each possible route. From Table 1, we can see that, for the first two routes, the answers are about 13.5 min and 15 min. However, in case of the route downtown, we can transfer only 12.9 megabytes during the next 30 minutes. Thus, the question goes beyond the prediction horizon and we cannot calculate the exact answer. The best we can do is to assume that the throughput remains at the level of the last predicted value (15 kbits/s), which gives the total transfer time of about 3 hours.

Thus, the answer is 13.5 minutes with 48% probability, 15 minutes with 32% probability, and 3 hours with 20% probability. Since we have an 80% certainty (which was the certainty level required) that the user will not go downtown, we can ignore the third route and look only at the two better choices. Hence, the overall answer is the answer of the second route, that is, 15 minutes. Intuitively, the answer means that with 80% certainty the user will select a route on which the time needed to transfer 30 megabytes is at most 15 minutes.

It is up to the e-mail agent to decide whether to send the attachments or not. The decision should be based on the predicted transferring time, importance of the files, and Joe's preferences. Quite likely, Joe does not want the e-mail agent to consume most of the available bandwidth for the next 15 minutes, thus slowing down all other network activity. Instead, the e-mail agent should decide not to send the files now, and ask the QoS Prediction Agent when it will be a better moment to do it. Obviously, the optimal decision would be to wait 13 minutes until the terminal is connected to LAN or WLAN. This frees the GPRS connection for other uses and allows closing the connection to save expenses when it is no longer needed.

In case the user goes downtown, transmission of the files should not be attempted at all, because the GPRS connection is too slow. However, sometimes there may be a possibility to optimize the network technology used. For example, the system could automatically switch to using a high-speed GSM or UMTS connection while approaching downtown if they offered better QoS than GPRS. This, of course, assumes that the user's profile allows the costs incurred by such a change.

Monads Experiences in Web Browsing

We selected web browsing as the sample application, mainly because the possibilities for adaptation in web browsing are wellunderstood (see e.g. [13, 14]). We also had access to a non-agent-based web optimization program, Mowgli WWW [14] that we were able to utilize both as a basis to build the sample application on, and as a baseline for evaluating the effectiveness of our application.

The Monads Web Agent controls the Mowgli WWW software by setting the compression levels separately for each image. In difference to plain Mowgli WWW, which uses static compression levels, the Monads Web agent adjusts compression levels dynamically depending on current and expected QoS, image size and image importance. Thus, larger images

are compressed more since the savings are greater, advertisements are exposed to extreme compression or dropped altogether, and during good QoS, minimal or no compression is used.

Principles of Operation

For each image, the agent attempts to determine an importance value. Currently an image can be broadly classified into four classes:

1. Advertisement banners,
2. link icons,
3. normal inline images, and
4. unknown images.

Advertisement recognition is based on whether the image is located on the same host as the HTML page, or whether its URL contains certain keywords. While primitive, the system does illustrate the principle. Link icons and inline images are easily recognized from the HTML page that contains them.

Once the importance of an image is known, the Web agent asks the QoS management to provide the current throughput, as well as a prediction for the transfer time of the image. These values, together with the size of the image, are given to an Image Compression Modeler Agent for classification. The returned classification is the degree of compression that should be used. Suitable compression parameters are then selected and given to the Mowgli Web proxy as additional HTTP parameters for the image.

Agent Interaction

The Monads Web agent uses two other agents, the QoS Prediction agent, which provides transfer time predictions, and the Image Compression Modeler agent, which allows the Web agent to learn from user feedback how images should be compressed (initially, a default compression model is used).

When the agent is started, it first creates a Modeler Agent for learning how images should be compressed. The decision depends on four variables:

1. Current QoS,
2. estimated image transfer time,
3. image size, and
4. image importance.

Attributes representing these variables are added to the Modeler, and an instance-based learning algorithm is selected. Then, a sample database is loaded that contains examples of how a certain type of image should be compressed with a specific set of current attribute values. The examples are given to the Modeler agent, and after that it is ready to function as an Image Compression Modeler and receive classification requests or feedback information.

Results

The performance of our system was measured against plain Mowgli WWW. It has to be noted that the system cannot out-perform Mowgli WWW, since it is built on top of it. Any decrease in load times is due to increased compression.

Since one of our main goals was to reduce load time variation in the face of variable QoS, we measured the load time standard deviation for images of different sizes[2]. Our tests showed that the load time standard deviation for a single image is significantly (10 – 35 %) smaller for the Monads Web agent than for plain Mowgli WWW. However, there are still too many factors that the agent has to estimate or guess (such as how compressible an image is, or how much data is currently buffered and waiting to be sent over the wireless link) for the load time to really be stable.

Our system works well in the case where a page is loaded when the QoS is about to drop. Plain Mowgli WWW is unable to adjust, and shows a dramatic increase in load time due to the QoS drop, while our system can anticipate the QoS drop and increase compression levels

[2] It should be noted that the test images were all large (20+ kB), since compression has no major effect on the load time of small images.

correspondingly. Our tests showed a load time decrease of 40-60% for pages with large images, which was usually enough to allow the pages to be loaded before the QoS drop.

The extra processing introduces a mostly static overhead for each image loaded. This is felt most heavily with pages that have a large number of small images. The delay depends on the processing speed in the agent (both machine and Java runtime speed), but can be as much as several seconds per web page. This is not too surprising, since our prototype makes load time estimates and compression decisions for all images, even small ones, although compression is of little benefit for those small images. In the next version of our prototype, we are planning to add a lightweight model for determining the circumstances in which it is useful to employ the heavier estimation models, and when it is best to simply pass on the image uncompressed.

As to the instance-based compression model, our experience was that attribute weights are crucial and hard to configure correctly. For example, since the load time of an image is what we are really interested in, that was given a large weight. However, this was found to be problematic for very large images, since the load time is then so long that, combined with the weight, it will overshadow the image importance attribute.

The feedback system, which allowed a user to add examples to the model, was found to be useful, but problematic because the complexities of attribute weighting sometimes caused new examples to have an unexpected impact on the overall behavior of the model.

In conclusion, the QoS adaptation and prediction works well, but another level of adaptation is needed to determine how much processing should be devoted to the QoS adaptation task. The compression model also requires further work. We are of the opinion that an instance-based model was perhaps not the best choice, and are evaluating other model types for the next version of our prototype.

Conclusion

Provision of seamless roaming between different administrative domains is a challenging task even when the underlying network technology remains the same. When predictions reliable enough are available, then the negotiation between the domains can be started early enough so that authentication and admission control are already done before the need of handoff. To provide seamless roaming—even in the looser form—between different technologies is much more challenging. In this case the applications need to adapt their behavior to the available QoS of communications resources.

In this paper we have described how software agents can be used to provide infrastructure services so that applications have time to adjust their behavior to the forthcoming changes in QoS. Although our research is still in an early stage, the results are encouraging. Modeling user behavior and available QoS separately enables predictions of reasonable accuracy. In this approach, the models are also quite small and simple. However, a lot of research is still needed in refining various models and in combining various predictions. For small devices, the distribution of functionality between terminal devices, network elements and application servers is an important issue.

Acknowledgements

This paper is based on work carried out in the research project Monads at the Helsinki University Computer Science Department. The project is funded by the National Technology Agency of Finland, Nokia Mobile Phones, Nokia Research Center and Sonera, Ltd. The authors are grateful to their colleagues in the Monads project, in particular to Markku Tamski from Nokia Mobile Phones, to Heikki Mannila from Nokia Research Center, to Heikki Helin and Heimo Laamanen from Sonera, and to Stefano Campadello, Henry Freedman and Sasu Tarkoma from the Monads group.

References

[1] L. Kleinrock, “Nomadicity: Anytime, Anywhere in a Disconnected World,” Mobile Networks and Applications, Vol. 1, No.4, January 1997, pp. 351-375.

[2] M. J. Woolridge and N. R. Jennings, “Intelligent Agents: Theory and Practice”, Knowledge Engineering Review, Vol. 10:2, 1995, pp. 115-152.

[3] OMG Agent Working Group, “OMG Green paper on Agent Technology”, OMG document ec/00-03-01, March 2000.

[4] WAP Forum, “Wireless Access Protocol Specifications”, http://www.wapforum.org/.

[5] M. Mouly and M.-B, Pautet, The GSM System for Mobile Communications, 1992.

[6] M. Rahnema, "Overview of the GSM System and Protocol Architecture", IEEE Communications Magazine, Vol. 31, No. 4, April 1993, pp. 92-1000.

[7] GSM Technical Specification, GSM 02.60, GPRS Service Description, Stage 1, 1999. Version 6.1.0.

[8] UMTS specifications. Available from 3GPP Web site: http://www.3gpp.org/.

[9] Cypher, Allen: “Eager: Programming Repetitive Tasks by Example”, CHI '91 Conference Proceedings. ACM Press, New York, 33-39, 1991.

[10] S. Franklin and A. Graesser: “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents”, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[11] F. S. C. da Silva, J. Agustí, A. C. V. de Melo, W. W. Vasconcelos and D. Robertson: ”Why Ontologies Are Not Enough for Knowledge Sharing”, IEA/AIE-99, Cairo, 520-529, 1999.

[12] P. Misikangas, M. Mäkelä and K. Raatikainen: “Predicting QoS for Nomadic Applications Using Intelligent Agents”, Impact '99 Workshop, 1999.

[13] V. N. Padmanabhan and J. C. Mogul, “Improving HTTP Latency”, Computer Networks and ISDN Systems, 28 (1 and 2), pp. 25-35, December 1995.

[14] M. Liljeberg, H. Helin, M. Kojo and K. Raatikainen, “Mowgli WWW Software: Improved Usability of WWW in Mobile WAN Environments”, Proc. IEEE Global Internet 1996 Conference, pp. 33-37, November 1996.