Supporting Nomadic Agent-based Applications in the FIPA Agent Architecture

Mikko Laukkanen, Heikki Helin, Heimo Laamanen
Sonera Corporation, Sonera Corporate R&D
P.O.Box 970 (Teollisuuskatu 13)
FIN-00051 SONERA, Helsinki, Finland
{mikko.t.laukkanen,heikki.j.helin,heimo.laamanen}@sonera.com

ABSTRACT
A high variety of Quality of Service (QoS) in data transmission over wireless networks creates challenges that have not been adequately addressed in today’s Internet-based services. Whereas today’s distributed applications may result in treating rapid and extreme changes in QoS as failures, in the nomadic environment comprising wireless data communications we consider them as a usual case. Therefore, the complexity of data transmission should be hidden from the nomadic user and applications and managed by an intelligent middleware. In this paper we present an agent-based middleware providing means for building adaptive applications for nomadic users. Our model introduces two agents for monitoring and controlling the wireless link, a QoS ontology and an efficient way for agent communication over the wireless link. Monitoring and controlling of the link is carried out by Monitor Agent and Control Agent. The ontology defines a QoS terminology and methods to access the services of monitor and control agents. The efficient way for agent communication consists of a bit-efficient encoding of agent messages and a message transport protocol designed for wireless links. Our implementation is built on top of FIPA-compliant agent platform.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—intelligent agents, multiagent systems;
C.2.4 [Computer-Communication Networks]: Distributed Systems—distributed applications

General Terms
Design

Keywords
Software agent, agent communication, nomadic application, wireless networks, adaptation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS’02, July 15-19, 2002, Bologna, Italy.
Copyright 2002 ACM 1-58113-480-0/02/0007 ...$5.00.

1. INTRODUCTION
The data communications environment of nomadic services—especially wireless data communications—creates many challenges that have not been adequately addressed in today’s Internet-based services. Wireless wide-area networks are in a phase of rapid development. High Speed Circuit Switched Data (HSCSD) and General Packet Radio Service (GPRS) are already in the market and the Universal Mobile Telecommunications System (UMTS) will be launched in 2002. Utilizing wireless LANs (WLAN) as public “hot spot” access networks to the Internet is currently under investigation and implementation.

Each of these networks has its specific data transmission characteristics. However, the basic challenge of wireless wide-area communications is as follows: In the environment of wireless data communications the Quality of Service (QoS) (such as line rate, delay, throughput, round-trip time, and error rate) may change dramatically. For example, when the user roams from a UMTS cell to a GPRS cell, the throughput may drop from 1 Mbits/s down to 24 Kbits/s. In addition, it is foreseen that seamless roaming between different network technologies (e.g., between UMTS and WLAN) will be needed in the near future, leading to an increase in the variability mentioned above. This high variety and high volatility environment of Internet-based services creates a need for adaptability, because users will demand services that will automatically and transparently adjust to the changes mentioned above.

We believe that software agent technology is an ideal methodology to design and implement a system, which addresses the problems mentioned above. Agents are able to reactively receive events about changes in the environment and autonomously act based on them. For instance, upon detecting that the current throughput of WLAN drops dramatically, an agent specialized in wireless communication may make a decision to seamlessly roam from WLAN to GPRS. After roaming to GPRS, the agent may inform application(s) about the new QoS. Having this information, applications are able to react based on the changing QoS, for instance by applying content adaptation.

In this paper we describe our agent-based system, which provides means for building adaptive applications for nomadic users. The work is a part of the CRUMPET\textsuperscript{1} [22] project, which is funded by EC IST\textsuperscript{2} 5th framework. CRUMPET\textsuperscript{1}

\textsuperscript{1}Creation of User-friendly Mobile services Personalized for Tourism.

\textsuperscript{2}European Commission, Information Society Technology.
PET’s aim is to implement, validate, and trial tourism related services for nomadic users. The services utilize different wireless networks, such as GSM, GPRS, and WLAN. CRUM-PET software agent environment is based on an open-source FIPA compliant agent platform called FIPA-OS [21] and its variation, MicroFIPA-OS [24], for handheld devices.

Our nomadic application support (NAS) architecture is based on FIPA Nomadic Application Support specifications [9], and is divided into three major parts: monitoring and controlling a wireless link, a QoS ontology and efficient agent communication. Monitoring and controlling of the link is carried out by Monitor Agent (MA) and Control Agent (CA). NAS ontology defines a vocabulary to access the services of MA and CA. The efficient agent communication consists of bit-efficient encoding for agent messages and a message transport protocol (MTP) designed for wireless environments.

NAS implementation can be plugged in to a Java-based FIPA-compliant agent platform. Once NAS-enabled, the agent platform and the agents running on it are able to roam between different networks, communicate efficiently over the wireless link and are provided with tools for adapting to changing QoS. However, in this paper we do not define any adaptation schemes nor policies; these are out of the scope of this paper.

The rest of this paper describes the design and implementation of NAS and is organized as follows. Section 2 specifies the architecture and design of NAS. In Section 3 the implementation of NAS is described in detail. Section 4 introduces an application utilizing NAS. In Section 5 a glance at the related work in this area is given. Finally, Section 6 concludes this paper.

2. ARCHITECTURE

Our architecture for nomadic application support is depicted in Figure 1. Mobile devices access the system through a wireless network, which can be for instance GSM. The access node connects the mobile devices to the services that can reside anywhere in the fixed network. In our implementation the mobile devices run a lightweight agent platform, whereas on the access node we have a full-blown FIPA-compliant agent platform. We call these lightweight and standard platforms, respectively. However, we do not rule out other configurations; for instance, a full-blown platform could be run on both mobile device and the access node. Both lightweight and standard platforms host MA and CA. In addition, both of the platforms may host additional agents that can use the services of MA and CA. For example, in the CRUM-PET architecture in the mobile device we have a client agent to take care of user interactions and in the access node there is a content adaptation service agent (CASA) that adapts the content based on the QoS information MA provides.

In order to allow the lightweight and standard platforms to take the wireless environment into account, we have added a number of components to them. For minimizing the bits transferred over the wireless, agent messages are encoded bit-efficiently according to FIPA specifications [6, 11]. To be able to recover sudden disconnections, messages are buffered in a similar way that is specified in [14]. At the lowest level we have a message transport protocol, which is designed for wireless environments. All of the components are described in more detail in Section 3.

Our architecture allows a centralization of the handling of the wireless link characteristics in one place, to the access node. Therefore, the agents in the fixed network do not have to take the characteristics of the wireless communication into account. Thus, agents in the fixed network do not need to know that the agent they are communicating with may reside behind a wireless link.

In order to achieve a wide degree of compatibility within FIPA implementations, the design is divided into two parts: the core and platform-specific parts. The core part is common to every platform, but the platform-specific part has to be configured for each platform. The configuration specifies how the NAS components (see Figure 1, NAS components in darker color) are plugged into the platform. This basically means that the developer has to implement certain interfaces in order to allow the core part to interact with the platform.

---

**Figure 1: Architecture for nomadic application support. Our components are painted in darker color.**
3. IMPLEMENTATION

The implementation of our architecture is mostly Java 1.1-compliant code. After studying the available JVMs suitable for our target device, we chose to use Sun JDK1.1.8. There are alternatives though, for instance J2ME CLCD/CDM, clean-room JVM implementations (such as Kaffe), and Java 2. Of these, we found that J2ME CLCD is too limited for our purposes, because it does not provide JNI interface, which is needed to access libraries written in C/C++. CDC does not, although having a lot smaller footprint, provide sufficient performance enhancements. Kaffe has problems in threading causing deadlocks, whereas Java 2 has too large a footprint for handheld devices.

Some functionality in the core part of our implementation is written in C, but the platform-specific part is written in pure Java. Furthermore, we have implemented some functionality using third-party software. However, we are not restricted to use these tools, but instead, they can be easily replaced.

On both mobile devices and the access node we run Linux. The reason for choosing Linux is that it has a good networking support, it is well configurable and because it is open-source, we can make modifications to the kernel if needed. Moreover, Linux is available for our target mobile device (Compaq iPAQ H3630). However, we are not ruling out other operating systems, such as PocketPC. Because most of the code is written in Java, we believe that porting from one operating system to another is not hard, although not trivial.

3.1 Nomadic Application Support Ontology

Our NAS ontology is loosely based on the FIPA Nomadic Application Support specifications [9], but we have made modifications and enhancements so that our NAS better serves its users [15].

We have modeled the ontology as Java objects so that all the complexity is hidden behind a NASOntologyFactory object, is able to create an ontology object out of the content-field of the ACL message. Figure 2 illustrates how this is implemented. We have three logical layers: tokenizers, content parsers, and ontology objects. Tokenizer layer contains parsers for different kinds of ontology-field encodings. Tokenizer performs two functions: it identifies the type of the content field and it outputs the parsed tokens to the content parser layer. Content parsers then create the correct ontology object. For instance, if XMLContentParser finds out that the content represents an OpenCommChannel function, it creates an OpenCommChannelParser, which in turn creates an actual OpenCommChannelOntology object out of the tokens.

3.2 Monitoring Quality of Service

Monitor Agent (MA) can serve its clients in two different ways. Firstly, it can answer a single query about the current QoS of a wireless link. Secondly, it can inform its client on a subscription basis about changes in the QoS of the wireless link. MA is accessed externally through an ACL interface using NAS ontology as described in previous section. Internally the platform-specific part accesses the core part of MA using MAFunctions interface. Having first parsed an incoming ACL message, MA passes the created Ontology object to MAFunctions-interface, which in turn, based on the object, creates and executes a corresponding Function object. This procedure is illustrated in Figure 3. In the case of a query for current QoS, MA delegates the acquisition of the QoS to QoSManager, which in turn uses Collector-interface to get the current value. Collector-interface is implemented by concrete Collector objects, each specializing in a certain QoS parameter(s). For the actual measurement of the QoS on the wireless link we use a tool called Nettimer [18], which either passively listens to network traffic or actively probes the network. However, implementations of NAS architecture are not restricted to use only these collectors, but any kind of a collector, which suits better to an application area in question, can be implemented. This can be done by writing a concrete collector object that implements the Collector-interface.

In a subscription an agent may define constraints for the variation in QoS so that when the QoS changes over the constraint, the agent is informed. We use a concept of watermarks to define constraints in the QoS of wireless data communications [29]. Two kinds of watermarks, high and low, are used to specify the constraints in a parameter value. When a high watermark is crossed from below, or a low watermark to define constraints in the QoS of wireless data communications [29]. Two kinds of watermarks, high and low, are used to specify the constraints in a parameter value. When a high watermark is crossed from below, or a low watermark from above, the watermark is said to be violated meaning that the parameter value has gone out of the constraints. Should this happen, the corresponding subscriber is notified. Furthermore, watermarks provide ways for avoiding jittering effect.

3.3 Controlling the Wireless Link

Control Agent (CA) provides three kinds of services: it can be asked to open/close a communication channel or activate/deactivate a message transport protocol, it may do those autonomously and it is able initiate roaming between different network technologies, such as GSM and WLAN. Each of these services are described in upcoming subsections.
3.3.1 Request-based operation

Whereas MA is accessed through MAFunctions-interface, platform-specific part accesses CA through a CAFunctions-interface, as shown in Figure 4. Behind the CAFunctions-interface there are a set of concrete Function classes, which are instantiated by CAFunctions, based on the request. All of the Function classes delegate link control functionality to ConnectionManager, which is able to open/close communication channels and activate/deactivate message transport protocols. This kind of design makes the concrete Function classes quite simple, thus it would be tempting to remove the Function class layer altogether and move its functionality to either CAFunctions or ConnectionManager. However, we do not want to rule out the possibility that CA would include functions, which need not to use ConnectionManager. Moreover, moving the functionality to CAFunctions is not an option, because we want to separate the interface from the concrete implementation.

The ConnectionManager component manages a set of concrete Connector classes. For each message transport connection (MTC) there is a corresponding Connector. In our implementation we are supporting Connectors for WLAN, GSM and GPRS connections, but adding new MTCs is easy by implementing a Connector for it.

Unlike managing MTCs, which is quite a low-level issue, managing the MTPs is dependent on the platform being used. For managing MTPs we have a set of Activator classes, one for each MTP. These classes are expected to be able to activate and deactivate a given MTP. However, we only provide implementation for HTTP MTP and our wireless MTP, which are plugged into FIPA-OS and MicroFIPA-OS. For other protocols, specific Activator classes must be implemented. As with MTCs, new MTPs can be easily added by implementing a new Activator for a corresponding MTP.

3.3.2 Autonomous operation

CA is also able to act autonomously based on the changes in the environment. It is aware, which network interfaces are attached to the device and what is their status. Figure 5 illustrates the autonomous operations.

Network devices are controlled by DeviceHandler, which generates events about network interface status, for instance when a network interface is attached to the device. We are currently controlling PCMCIA slots and serial port. The event is delivered for all the EventListeners, which are registered to DeviceHandler, including CA. Based on the events, CA can make autonomous decisions on how to react to them. For instance, upon receiving an event of WLAN card insertion at PCMCIA slot, the obvious reaction from CA would be to check if the WLAN coverage is available, and if it is, make a connection to the access node using WLAN.

However, end-users might not be delighted about having an autonomous system to make connections on its own. Therefore, the decisions CA makes can be controlled or biased by user preferences. While the user preferences mainly consists of information on how to connect to a certain network and access node, they also dictate the rules on how prefer a certain connections over another.

3.3.3 Support for Roaming

Roaming between networks is initiated either by the user or autonomously by the system. In the former case the user replaces the network adapter herself, whereas in the latter case a change in the QoS of the wireless link results in a notification from MA to CA, which in turn may replace the MTP or close the current MTC altogether and open another one, possibly with some other MTP. Therefore, only the latter case can be considered as real seamless roaming while the first one is more like handling a disconnection on the wireless network connection.

We assume that all the communication is carried out on top of IP. Roaming always includes a re-connection to the access node and it may happen that the mobile device is assigned a different IP address than before. In order to handle IP level roaming, each mobile device is assigned a static and unique mobile device ID, which is advertised to the access node and kept in the connection setup. Access node handles the mapping between mobile device ID and IP address to correctly identify devices.

When user removes a network adapter, the CA’s DeviceHandler notifies ConnectionManager about the removal (see Figure 5). ConnectionManager in turn brings down the network interface, and frees any resources associated to it. At this point the communication is suspended and agent messages are being buffered.

As the user reinserts a network adapter, ConnectionManager is again notified by the DeviceHandler. Based on the network adapter, ConnectionManager asks MA for available access networks. If a suitable network is available, the user preferences are checked for valid login parameters for the available access network. The parameters depend on the access network. For instance in the case of GSM network they can be phone number, login ID, and password of the Internet service provider. If the parameters are found, a connection is established to the access node and the possibly newly assigned IP address and mobile device ID is advertised. After this, the communication can be resumed and agent message buffers at both mobile device and access
node are flushed.

System may also implicitly initiate roaming when the QoS of the wireless link changes dramatically, for instance so that the throughput drops below a user-defined limit. Should this happen, the CA's ConnectionManager receives a QoS event from MA (see Figure 5). At this point the CA has to make a decision about the possibility for seamless roaming. First of all, there has to be another access network available. Secondly, the mobile device has to have another network adapter inserted, which is able to access the network. Thirdly, the user has to have valid access parameters for the network.

If the CA decides to roam, it first brings up the new network interface and establishes connection to the access node. The agent messages are then routed through the new network interface. After that the old connection is brought down and any resources associated to it are freed.

### 3.4 Efficient Agent Communication

In distributed systems, communication is an essential component of the systems. This is also true in multi-agent systems. To exchange their knowledge, agents should be able to communicate with one another. In wireless environments, the communication should be tailored to provide an efficient use of scarce and fluctuating data communication resources. Furthermore, sometimes, efficiency is not so important an aspect as, for example, reliability.

Agent communication in wireless environments should be tailored on each communication layer, from the transport protocol layer up to the application layer. Sometimes, applying changes only on one layer might be meaningless as an inefficient solution on another layer deteriorates all the changes on other layers. Our NAS implementation provides both a message transport protocol designed for wireless environment and an efficient way to encode message envelopes and ACL messages. In the following sections, we will describe these in detail.

#### 3.4.1 Wireless Message Transport Protocol

An important function of the message transport protocol is to provide reliable and efficient message delivery. Our NAS implementation supports two MTPs: HTTP and Wireless Message Transport Protocol (WMTP). The former is used as specified in [8], and the latter is designed and implemented by us. Currently, both of the MTPs can be plugged into the FIPA-OS and MicroFIPA-OS.

There are three main problems with using HTTP in wireless environments as specified in [8]. Firstly, the specification does not mandate the usage of HTTP 1.1 persistent connections [5], although persistent connections can be used. In a high-latency wireless connection, opening a TCP socket typically takes several hundred milliseconds. Therefore, using a separate socket for each agent message is inappropriate. Implementing persistent connections only in the mobile device is not enough if the destination platform in the fixed network does not support them. However, using an HTTP proxy in a fixed network, which supports persistent connections, may solve this problem. Secondly, the HTTP headers are encoded using ASCII characters, which causes additional overhead. This overhead can be considered somewhat significant, especially if the agent message size is small. Lastly, the HTTP protocol does not provide sufficient reliability; it mainly relies on the transport protocol, TCP is this case. TCP may not recover properly in the case of a wireless link disconnection, thus behaving unreliably. Furthermore, it is well known that TCP does not performs well in slow wireless links (see e.g., [20]). In addition to these, it should be noted that operators typically do not let incoming connections access the devices in their network. Given this, it is impossible for an agent on the fixed network to initiate a connection to a mobile device in an operators network.

To overcome these problems we designed a Wireless Message Transport (WMTP) [15], which provides both efficient and reliable agent communication. WMTP uses TCP as a transport protocol, but whereas HTTP MTP has to establish a TCP connection for every message separately, WMTP uses persistent connections for communication with a peer. To compare the performance of WMTP over the HTTP MTP we run two simple test cases. We also included HTTP MTP with persistent connections (P-HTTP) for comparison. The wireless link used in measurements was implemented using the Seawind wireless network simulator [17].

In the first test case, we initiate a FIPA-Request protocol [10] by sending a request message to the participant. The participant replies by sending back agree and inform to the initiator. In Figure 6, the measured times (in milliseconds) of the received agree and inform messages at the initiator are illustrated. The payload size of each message is about 2Kb.

In the second test case we use a subscription protocol, where the initiator first sends a subscribe message, to which the participant replies by sending back a sequence of inform messages. Figure 7 shows the measured times (in milliseconds) of the received inform messages at the initiator. The payload size of each message is about 2Kb.

The results show that HTTP MTP performs significantly worse than WMTP and P-HTTP. The main reason is that establishing the TCP connection for every message in the HTTP MTP results in significant performance overhead as compared to WMTP and P-HTTP. By definition, the TCP needs one round-trip for connection establishment; only after that, the actual message can be transferred. Because the delay in slow wireless networks is relatively long (in our tests we used 300 millisecond delay), the extra round-trip really makes a difference; when the throughput increases, the relative difference between these protocols increases. Furthermore, as agent messages are usually relatively small, the
TCP slow-start algorithm affects to performance significantly. The results also show that in the first test case the P-HTTP cannot match the performance of WMTP, but in the second case the difference is insignificant. The reason for this is that with only a few messages the connection establishment overhead in HTTP and P-HTTP results in additional overhead. Furthermore, especially with slow wireless links, HTTP cannot match the performance of WMTP, but in the first test case the P-HTTP perform worse than WMTP.

3.4.2 Bit-efficient Message Envelope

The purpose of the message envelope layer is to enable an independent handling of the transport protocol, for example routing of messages without touching the ACL. FIPA has specified an XML-based syntax [7] for the message envelope to complement the XML-based syntax for FIPA-ACL [13] and HTTP MTTP [8]. However, the bit-efficient [6] message envelope encoding is meant for the wireless environment. This syntax is designed to complement bit-efficient encoding of FIPA-ACL [11]. We found out that the size of a message envelope using XML encoding is over twice as large as when using bit-efficient encoding. For more detailed test results, see [16].

3.4.3 Bit-efficient ACL

For FIPA-ACL three standard encoding schemes have been specified. First one is based on ASCII strings [12], and therefore it is non-optimal. The second one is based on XML [13], which is a highly verbose syntax. The third one is bit-efficient syntax [11], which is especially designed for wireless environments.

In bit-efficient FIPA-ACL there are two primary ways to reduce the transfer volume over the wireless link: data reduction and intelligent caching. First, FIPA-ACL messages are encoded efficiently by using one octet codes for predefined message parameters and other common parts of messages. By intelligent caching we mean that similar parts of subsequent messages are not transmitted multiple times over the communication channel, but subsequent occurrences are replaced by short codes.

The number of bytes needed to represent a FIPA-ACL message is significantly lower than in String and XML-encodings. Using the string encoding we need two times and in XML encoding even three times more bytes than in bit-efficient encoding. In the case of creation and parsing times of a bit-efficient FIPA-ACL messages, we are looking at speed-ups of about 30% and 50%, respectively, as compared to String encoding. For more detailed information about the performance of bit-efficient FIPA-ACL, see [16].

3.5 Disconnection Handling

Disconnection handling is an essential operation when operating in wireless environments, where sudden disconnections may happen frequently. We are handling disconnections by buffering outgoing agent messages upon a disconnection in a similar way that is specified in [14]. Because buffering of agent messages consumes resources, we allow the specification for buffering time on message basis by giving agents an opportunity to set a buffering time parameter to the message envelope. However, the system may override this parameter and empty the buffer before timeout, if the resources on the device are running too low.

4. APPLICATION

CRUMPET system builds service adaptation on top of nomadic application support. In CRUMPET the NAS is utilized mainly by three agents: CRUMPET Client Agent (CCA), Dialog Control Agent (DCA) and Content Adaptation Service Agent (CASA). In addition to these, any service agent providing content can benefit from NAS. Moreover, for the autonomous link control, CA in the mobile device subscribes to QoS changes from MA.

CCA resides in the mobile device and handles user interaction. If a user requests explicitly to open/close communication channel or change MTP, CCA delegates these to CA. For making autonomous decisions CA uses QoS information provided by MA. DCA is assigned to CCA as the CCA establishes connection to the access node. The main task of DCA is to route user requests from CCA to appropriate service agents. Along with the connection establishment, the CCA sends the terminal type identifier to the DCA. DCA in turn passes this to CASA, which maps the identifier to a terminal profile. The terminal profile is used as a basis for static content adaptation, which means adaptation to for instance QoS parameters that do not change while the CCA is connected to the access node. CASA is an agent specialized in content adaptation both based on the terminal profile and current QoS of the wireless link that is provided by MA.

In the following we describe a scenario, with which we illustrate, how the cooperation between these agents enables adapted services for a nomadic user. We begin with an initial situation, where we assume that the system is up and running, but the user has not logged into the system. The scenario is illustrated in Figures 8 and 9, to which the numbers in parentheses in the text refer. Dashed lines in the figures imply ACL communication, whereas solid lines refer to data exchange, which may be applied using non-ACL communication.

4.1 The Login Procedure

When the user starts up the system on her mobile device, the CCA asks CA for a connection to the access node (1). CA then queries MA for a list of available wireless networks (2), and based on the user preferences (3), it establishes a connection to the access node. CA informs CCA about suc-
4.2 Accessing Adapted Services

Once logged in, the user asks CCA for some content, for instance to draw a map of neighboring area with points of interests. CCA forwards the request to DCA (7), which in turn brokers the requests to a specific service agents. Service agents, unaware of neither current dynamic nor static QoS parameter, reply to DCA with unadapted content (8). It should be noted that in this scenario we do not take any stand on the content type itself, but assume it to be of discrete type, for instance text, images, video clips etc. DCA uses CASA to adapt the content based on the device profile and current QoS of the wireless link (provided by MA). Based on the static and dynamic QoS parameters CASA decides the appropriate content adaptation algorithm and replies to DCA with adapted content. Finally, DCA replies to CCA, which outputs the content to the user.

5. RELATED WORK

Recently, a number of research efforts have looked at supporting nomadic users and applications, but only a few of those are agent-based. JADE-LEAP [3] is a lightweight agent platform that is executable on small devices such as PDAs and phones. JADE-LEAP is based on open-source agent platform JADE [2]. Just like with FIPA-OS, the NAS implementation could be plugged into JADE and JADE-LEAP; in fact, bit-efficient codec of the NAS implementation is already included in JADE.

RAJA (Resource-Adaptive Java Agent Infrastructure) [4] is a generic architecture, where domain-specific functionality is separated from resource and adaptation concerns. It is divided into system and applications levels. The system level provides resource management services implemented by a set of agents. The application level implements domain-specific computation, and contains two kinds of agents, Basis Agents and Controller Agent. Controller Agents are attached to resource-unaware Basis Agents to provide the adaptation support. Controller Agents in turn make use of the system level services. Nomadic application support could be plugged in to the system level for enriching the RAJA resource management.

Agents2Go [1] is a research project motivated by Mobile Electronic Commerce, which requires that the system provides mobile users with high-quality, precise and context-relevant information. For eliminating the congestion related problems of TCP, Agents2Go proposes a protocol called CentaurusComm, which provides a reliable and message-oriented data transmission targeted at wireless wide-area networks. Whereas CentaurusComm seeks to replace TCP, our WMTP operates on top of TCP, thus not needing any modifications to protocol stacks.

Monads project [19] has implemented an agent-based architecture providing adaptive services. The principal idea is 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. Our system operates only on parameters of present time, which is enough when there is no upper bound to the delays from changing for instance content encoding, but if we were to support for instance adaptive streaming video, the only option to guarantee low delays is to predict the changes in the environment.

6. CONCLUSIONS

In this paper we described an agent-based middleware for supporting adaptive applications. Monitor Agent provides real-time information about the current quality of service, based on which the applications may tune the content transferred over to the nomadic user. Control Agent manages the wireless link by establishing connections and activating message transport protocols. It acts either by request or autonomously based on the current environment.

For efficient agent communication we discussed a message transport protocol specifically designed for wireless environment. We demonstrated that it is significantly more efficient than FIPA-specified HTTP message transport protocol. For minimizing the bits transferred over the wireless link we encode the message envelope and ACL message bit-efficiently according to FIPA’s specifications.

Our system is mostly written Java and it runs on Linux. The target device is Compaq iPAQ H3630 handheld device, but it works on any device running Linux and having sufficient resources. In the future we are going to integrate our nomadic application system to other FIPA-compliant agent platforms, such as JADE-LEAP. We are also considering of porting the code on to PocketPC/PocketPC 2002 devices, which would enrich the variety of supported devices.

We will test our system along with CRUMPET system, which will be trialled in the end of Summer 2002. Some parts of the system are already being used in FIPA-OS and JADE, but after the trials the whole system will be made available as open-source.
7. REFERENCES


