Efficient Agent Communication in Wireless Environments

Heikki Helin and Mikko Laukkanen

Abstract. In wireless environments, communication should be tailored to enable an efficient use of scarce and fluctuating data communication resources. In this chapter we consider software agent communication in such environments. We introduce a layered model of agent communication in the context of the FIPA agent architecture. We have designed and implemented efficient solutions for wireless agent communication for each layer of this communication stack. Further, we thoroughly analyze the performance of agent communication in slow wireless environments. The analysis shows that agent communication in wireless environments could be improved significantly as long as all communication layers in the agent communication stack are appropriately taken into account.

1. Introduction

The progress in wireless network technologies and mobile devices changes the ways in which people access services. A user may access the same services as she would using her desktop computer, but in the nomadic environment she is able to do so anywhere, at any time and even using a variety of different kinds of devices. Such an environment places new challenges on the architecture implementing the services. Nomadic environments differs from stationary environments in two fundamental ways. Firstly, the user may be situated in an environment, where multiple data communication networks may be available. Because of the different network types and characteristics of the networks, for instance the values of Quality-of-Service (QoS) parameters may change dramatically based on the network that the user is currently connected to. Secondly, the user may access the services using a variety of different mobile or stationary devices. The characteristics and limitations of a device dictates the constraints on how the user is able to access the services and what kind of content the user is provided with. Further, the other contextual parameters, such as user preferences, need to be taken into account. For instance,

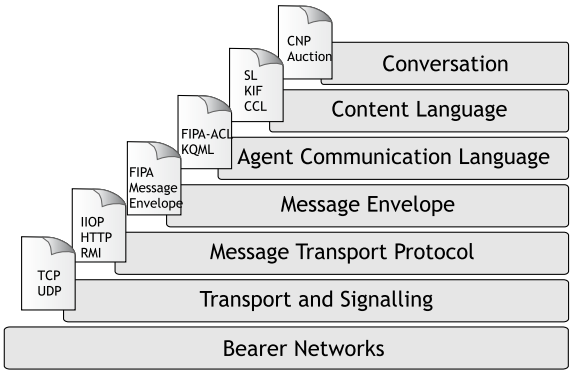


FIGURE 1. Agent communication layers

the user may prefer not having pictures at all, even if the terminal device and network connection would allow them.

We consider agent-to-agent communication over a wireless communication path. In wireless environments, the agents need to communicate efficiently and the communication should be reliable. Therefore, the communication stack should be tailored for the wireless environment. We will take a rather pragmatic view of agent communication. In particular, we neither consider why the agents are communicating, nor we consider the semantics of the messages. However, we assume that agents are communicating with each other and that at least a part of the communication path is implemented using wireless technology.

The rest of this chapter is structured as follows. In Section 2 we introduce a layered model of agent communication. Sections 3, 4, 5, 6, and 7 provide a performance analysis of message transport protocols, message envelopes, agent communication languages, content languages, and interaction protocols, respectively. Section 8 describes related research on using communicating agents in wireless environments. Finally, Section 9 summarizes our contributions.

2. Layered Model of Agent Communication

2.1. Overview of Agent Communication Stack

Figure 1 depicts a layered model of agent communication. The transport and signaling protocol layer should provide an efficient and reliable data transport service. Usually this layer should be transparent to agents. Therefore, the agents are typically unable to optimize anything at this layer by themselves. Given this, we will not discuss issues on this layer in more detail here. An overview of transport protocol issues in wireless environments can be found in [33], as an example.

A message transport protocol (MTP) defines the structure of messages sent using a transport protocol. Typically the MTP implicitly defines the transport

protocol as well. Should this not be the case, the agents must agree on which transport protocol to use. FIPA has specified three message transport protocols: IIOP [10], HTTP [9], and WAP [11]. The transport layer and the message transport protocol layer do not necessarily differ at all from the communication in traditional distributed systems. What makes the agent communication different is how the communication above these two layers is modelled. Once the agents have agreed on these two protocols, they are able to transmit data between each other. However, to be able to exchange arbitrary data does not mean that agents can communicate meaningfully.

Given that several message transport protocols can be used, and these protocols can have different behaviour, FIPA has defined the concept of an envelope. The message envelope defines how the message should be routed, for example, among other parameters. The message envelope is sometimes independent of the MTP, but sometimes they are tightly coupled. An example of tight coupling is the IIOP protocol in the FIPA architecture. In this MTP, the message envelope is built-in to the protocol definition, that is, the IDL interface defines the structure of the message envelope. In this particular case, the tight coupling is well justified. In what follows, we assume that any concrete message envelope encoding can be used with any MTP with an obvious exception of IIOP MTP. This assumption gives us more freedom, but also introduces a problem that communicating agents should be able to agree on which concrete message envelope encoding to use. However, we do not consider that problem here.

Agent Communication Language (ACL) defines both the syntax and the semantics of agent messages. Several agent communication languages are developed, such as FIPA-ACL [13] and KQML [25]. The ACL layer consists of two sub-layers: An abstract layer that defines the semantics of the language and a concrete layer that defines the syntax of the language. For example, the abstract FIPA-ACL defines the message semantics, but is unconcerned with the encoding of the message; another layer defines the syntax of messages. FIPA has specified three encoding schemes for FIPA-ACL: String-based [5], XML-based [6], and bit-efficient [4].

Typically, an ACL lacks means for defining the content of the message. For example, by using the REQUEST communicative act in FIPA-ACL, the sender of the message applies to the receiver to perform some action. In ACL, the sender defines that the message is a REQUEST-message, but says nothing about the action that the receiver should perform. The action is described in the content language. FIPA content language library defines several content languages [14]. Each of these languages has one concrete encoding scheme, but in the future they may have different encoding schemes. To define the message content, the content language alone is insufficient, as it typically fails to define the terminology used in communication. Therefore, to have a common understanding of the message content, the communicating agents should share a common ontology.

The agent communication typically falls into common patterns. In FIPA specifications, these are called interaction protocols. Perhaps more typically in the

literature these are called conversation protocols or conversation patterns. An interaction protocol defines a common pattern of conversations used to perform a task.

If only those choices defined by FIPA for various layers are taken into account, there are a total of 60 different possible combinations. If proprietary choices are taken into account, the number of possible combinations explodes drastically. Now, the problem is how the agents can efficiently agree on different issues. Usually, the environment reduces the appropriate possibilities. For example, when operating in a wireless environment, encoding schemes and protocols designed for these environments should be used. In these cases, the selection can be done using prior knowledge. On the other hand, even if the agent itself is operating in a wireless environment, the peer agent may be operating in a wireline environment and especially it may be unaware of the possibility of a wireless environment. In these cases, perhaps the best approach is to use a gateway that can perform necessary translations between incompatible choices [24, 27].

2.2. Analytical Performance Model

In wireless environments, the agents need to communicate efficiently and the communication should be reliable. Therefore, the communication stack discussed above should be tailored for the wireless environment. At the conversation layer communications patterns should be optimized so that agent message exchanges are carried out with a minimal number of round-trips. This is especially important when using a high-latency communication path. It is important to notice, that 'minimal' here does not necessarily mean the absolute minimal value; sometimes it is better to use more round-trips to achieve a better result. The encoding of the content language, the agent communication language, and the message envelope should be selected so that the scarce communication path is utilized as efficiently as possible. The MTP should be able to transfer messages over a wireless link reliably and efficiently. As noted earlier, selecting a message transport protocol may affect the selection of a transport protocol. For example, if the transport protocol is also reliable in wireless environments, the MTP implementation can be much simpler. Typically, however, this is not the case, and therefore reliability should be implemented into the MTP. In the following sections, we discuss these issues in more detail, and point out some optimization techniques.

The size of an agent message consists of six parts:

$$D_{msg} = D_{tp} + D_{mtp} + D_{env} + D_{acl} + D_{cl} + D_{ont} \tag{1}$$

To exchange messages efficiently, the D_{msg} should be minimized, which can be achieved by minimizing each component on the right side of the equation. Firstly, D_{tp} defines the overhead caused by the transport protocol. This component is typically dependent of other components. For example, one can easily determine the size of a TCP segment header. However, the total size of the other components defines how many TCP segments are necessary to transmit the whole message. Obviously, there are also other aspects that affect, such as the MTU size. Secondly,

D_{mtp} defines the overhead caused by the MTP. This is typically independent of other components; especially in each MTP defined by FIPA there is at most one MTP header by an agent message. Thirdly, D_{env} and D_{acl} define the overhead caused by the message envelope and the ACL, respectively. Fourthly, D_{cl} and D_{ont}, the overhead caused by the content language and the ontology, respectively. D_{ont} depends on D_{cl} because typically the content language defines how the terms in a given ontology are encoded into the message.

In the following sections, we consider reducing the overhead caused by each component. However, it is important to notice that although sometimes a minimal encoding of a given component is inappropriate. There are at least two reasons for this. Firstly, the computing power needed to encode the component may be too much compared to saving gained from efficient encoding. For example, assume the size of a given component is x bytes and the link bandwidth is $2x$ bytes per second. Now, if the component is encoded more efficiently giving the output size $3/4x$ bytes, but the encoding time is $1/2$ second, obviously the encoding was unnecessary and, more importantly, harmful. Such situation is likely when using mobile phones with very limited processing power. Secondly, having an efficient non-standard encoding scheme deteriorates interoperability. This is especially true in the transport protocol layer and in the message transport protocol layers, as usually network components that are not aware of agent systems must understand these layers. For example, although it is possible to define a transport protocol especially suitable for agent messages, such a protocol probably will not be widely accepted by the Internet community, and therefore it is an unattractive choice. On the other hand, the encoding of an ACL is an "agent-level" issue, and thus we have more freedom at that layer, as an example.

3. Message Transport Protocol Layer

In comparing the performance of the MTPs, we conducted exhaustive experiment in a simulated wireless environment. From possible MTPs, we selected IIOP, HTTP, Persistent HTTP (P-HTTP), WAP (CFW), Java RMI, and MAMAv2, which we will next analyze thoroughly. The IIOP and HTTP will used as specified by FIPA. The persistent HTTP (P-HTTP) is similar to that of HTTP protocol, but the sender does not close the TCP socket after receiving the reply, but uses the same TCP socket for subsequent messages. However, for each interaction protocol, two TCP sockets are needed, since the P-HTTP protocol allows only sending messaging to one direction over one TCP socket. The WAP implementation used in our evaluation is a WAP emulator in which message sequences are same as in WAP protocol. Therefore, we assume that the performance of our implementation and a real WAP are similar, since the protocol overhead as well as message sequences in these two protocols are about the same. Java RMI, although being a non-standard option, is taken into account because it is used in many FIPA-compliant agent platforms for intra-platform communication (e.g., in Jade [1]).

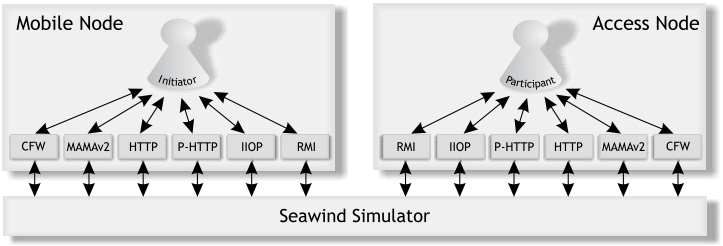


FIGURE 2. High-level overview of the MTP experiment configuration

The MAMAv2 protocol is designed by authors [26]. The objective of this protocol is to provide efficient and reliable agent message transport services over a (slow) wireless link for higher layers. As a basic service, the protocol offers a bi-directional semi-reliable message channel for the message transmission over a (wireless) link.

We use a typical client-server scenario to perform experiments, where the client (initiator) is executed at a mobile node and the server (participant) in an access node (see Figure 2). Each MTP we analyze using 8 different wireless network configurations (connection rates 9600bps, 28,800bps, 57,600bps, and 115,200bps; each with two propagation delay values (150ms and 300ms)), with three different conversation patterns (see Figure 3), and four different message payload sizes. The wireless link is simulated using the Seawind wireless link simulator [29]. Each experiment is repeated 11 times, but only 10 last repetitions are taken into account. The reason for this is that most of the software is implemented in Java and Java does some (heavy) initialization when particular classes are used first time.

In each case, four different payloads (message envelope + ACL) will be used. The smallest payload is about 0.5 kilobytes and largest about 10 kilobytes. Using a different size of ACL message generates different payload size, that is, the message envelope is constant through the experiment, about 250 bytes. However, the actual payload varies depending on the MTP. For example, in the IIOP protocol, the message envelope is expressed in terms of IDL. This means that fields are encoded using binary codes, and therefore the message envelope size in the IIOP protocol case is slightly smaller than with other MTPs. On the other hand, each MTP adds its own overhead. For example, in the HTTP protocol, the message headers are expressed using ASCII characters, which obviously increases the actual payload. This can be seen especially in those experiments where the message payload is small.

In the first test case, we initiate a FIPA-Query protocol [21] by sending a QUERY message to the participant. The participant replies by sending back an INFORM message to the initiator (see Figure 3 (a)). The purpose of this test is to measure the round-trip time in agent communication. In the second test case, we initiate a FIPA-Request protocol [22] by sending a REQUEST message to the

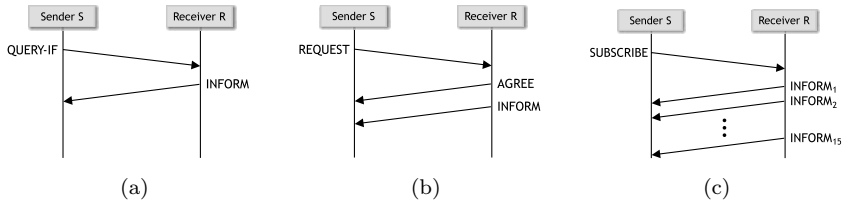


FIGURE 3. Message exchanges in interaction protocols used in the MTP evaluation

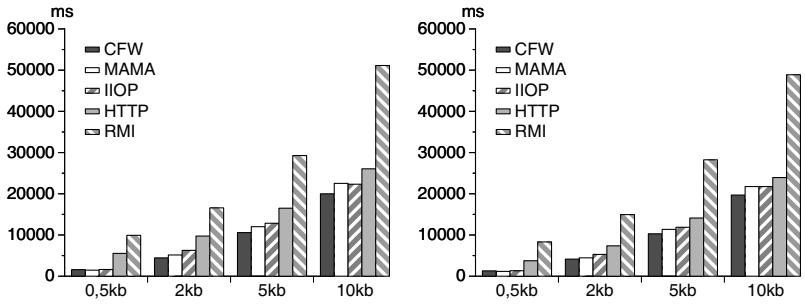
participant. The participant replies by sending back an AGREE message[1] and an INFORM message to the initiator (see Figure 3 (b)). Although this protocol is quite similar to that of the first case, this case is taken into account, as the FIPA-Request protocol is perhaps the most widely used interaction protocol in the FIPA architecture. In the last 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 (15) of INFORM messages to the initiator (see Figure 3 (c)).

3.1. Simple Round-Trip Case

Figure 4 compares the selected MTPs using FIPA-Query interaction protocol with 9600bps speed. Clearly, RMI protocol is the most inefficient. For example, having only 0.5kb payload and the slowest link, it takes more than 10 seconds to finish this interaction. Similarly, HTTP protocol is somewhat inefficient when the payload is small. This is due the fact that the protocol needs to open two TCP sockets, which takes most of the time when having small payload. However, for example in case (a) with 10kb payload, the HTTP is almost as efficient as MAMA and IIOP. In addition, the protocol overhead is bigger in HTTP, as HTTP headers are ASCII strings as well as MIME boundaries. MAMAv2 and IIOP protocols are about equally fast in these measurements. In the MAMAv2 protocol, no TCP sockets are opened during the interaction, but an existing TCP socket is used for all communication. IIOP protocol needs to open two TCP sockets. However, as noted earlier, the first interaction is not taken into account, and therefore IIOP performs reasonably well. The performance of the CFW protocol is the best in all cases, as was expected. However, it is important to note that the CFW protocol lacks sufficient reliability, and therefore its implementation is insufficient for real-life use, and therefore it is not directly comparable with other MTPs in this experiment. P-HTTP protocol is omitted in this case, as its performance would be exactly the same as HTTP protocol's. The results of FIPA-Query interaction using connection speeds 28,800bps, 57,600bps and, 115,200bps are similar to those of 9600bps.

Especially when the payload is small, an MTP that opens a new TCP socket for each message are highly inefficient. This was expected, because opening a TCP

[1]Although the AGREE message in the FIPA-Request protocol is optional, we use it here to show the negative effect of opening an "unnecessary" TCP socket in some MTPs.



(a) 9600bps (300ms) (b) 9600bps (150ms)

FIGURE 4. Comparison between MTPs using query interaction with 9600bps connection speed

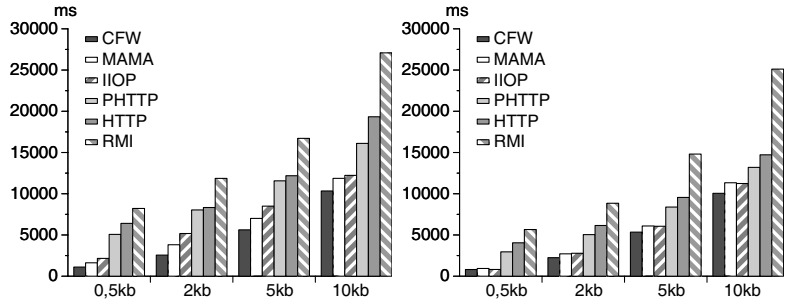
socket takes one round-trip, which is about 600 milliseconds when having 300 milliseconds propagation delay. Furthermore, in these cases, two TCP sockets are needed for sending two messages, which means that almost 1.5 seconds is needed just for opening the TCP socket. For example, when the payload is small, those MTPs that do not need to open a TCP socket for each message can complete the whole interaction in less than 1.5 seconds. Therefore, it is obvious that opening a new TCP socket per message is highly inefficient in environments where the propagation delay is relatively large.

3.2. FIPA-Request Case

Figure 5 compares MTPs using FIPA-Request interaction protocol using connection speed 28,800bps. As noted earlier, this interaction is quite similar to that of FIPA-Query; the only difference is that the participant sends two replies instead of one as in FIPA-Query interaction. Given the similarities of the interaction protocols, the results are also very similar. A difference to FIPA-Query case is that in this experiment we also used P-HTTP MTP. As can be seen, the results P-HTTP are quite similar to those of HTTP. This was expected, as the only difference between these two MTPs is that HTTP needs to open two TCP sockets for replies where as P-HTTP needs only one. The results of FIPA-Request interaction using other connection speeds are similar to those of 28,800bps.

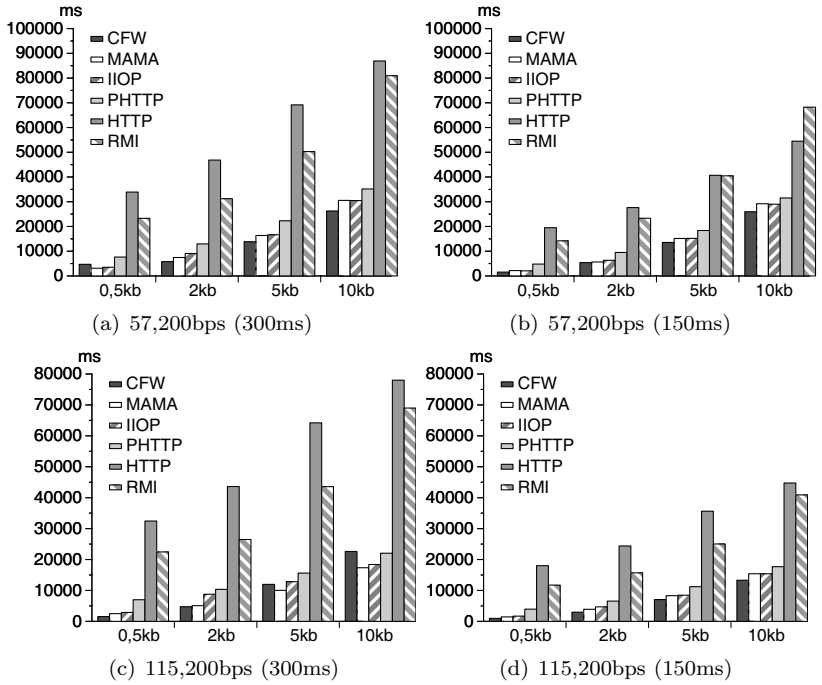
3.3. Subscription Case

Figure 6 compares MTPs using the subscription interaction protocol using connection speeds 57,600bps and 115,200bps. In these results the effects of opening a TCP socket for each message can be seen clearly. The performance of HTTP and RMI is significantly worse than that of CFW, MAMAv2, IIOP, and P-HTTP. The performance of P-HTTP is slightly worse than that of MAMAv2 and IIOP. This was expected since the actual payload is larger in P-HTTP because of HTTP headers. Furthermore, because of this, the relative difference is bigger when the



(a) 28,800bps (300ms) (b) 28,800bps (150ms)

FIGURE 5. Comparison between MTPs using request interaction with 28,800bps connection speed



(a) 57,200bps (300ms) (b) 57,200bps (150ms)

(c) 115,200bps (300ms) (d) 115,200bps (150ms)

FIGURE 6. Comparison between MTPs using subscription interaction with 57,600bps and 115,200bps connection speeds

actual payload is smaller. For example, in 28,800bps connection speed with 300ms delay, the P-HTTP is about 2.2 times slower than MAMAv2 when the payload is 0.5kb, but only about 1.1 timer slower when the payload is 10kb. The actual time

difference in both cases is about the same—about five seconds, which is due to more inefficient TCP socket usage in P-HTTP.

When there are more messages that just one or two in simple interactions between the mobile node and the fixed network, the difference between CFW, MAMAv2, IIOP, and P-HTTP is insignificant. P-HTTP performs slightly worse the MAMAv2 and IIOP, mainly because of additional payload caused by HTTP headers and message envelope and ACL part separation mechanisms. On the other hand, the implementation of P-HTTP is much simpler than the other two protocols. Furthermore, the P-HTTP can be improved by carefully selecting which HTTP headers should be included in the message. If the communication is only between a mobile node and selected access node at the fixed network, not all HTTP headers mandated by FIPA specification [9] are necessary. Additionally, in such environments, the performance of P-HTTP can be improved by using the same TCP socket for messaging in both directions. Using HTTP or RMI is clearly not an option in such wireless environments we tested in this experiment, because of the bad performance.

4. Message Envelope Layer

In this section, we present a performance evaluation of message envelope encoding, and give a short analysis of the results. The selected encoding schemes are analyzed in number of output bytes needed for transmitting the message envelope.

We selected five different encoding schemes for the evaluation: IDL [10], Bit-efficient [7], XML [8], Binary-XML, and serialized Java object. In the FIPA-IIOP MTP [10], the message envelope is encoded to the GIOP message and all field codes are binary data. Therefore, this encoding is expected to be quite efficient in the number of bytes it produces. The syntax of the bit-efficient envelope is similar to that of FIPA-ACL [4]. This allows implementations to use (at least partially) same parser for envelopes as for ACL messages. XML DTD for message envelope is defined in [8]. This encoding scheme is expected to be highly verbose. Given the verbose syntax of XML, several binary-XML encoding schemes have been developed. For this evaluation, we choose the one provided by the WAP Forum [36]. This encoding allows two different ways to encode the message. Firstly, binary-XML can be used with or without special encoding tokens. We will evaluate both of these options, although neither of them is a FIPA standard. The last encoding scheme, serialized Java object, is not a FIPA standard, but it is widely used in intra-platform communication. The actual object we will use in this experiment is Jade's Envelope class [1].

All the experiments are conducted in a Linux environment using the Jade agent platform (version 2.5) [1] with JDK1.3. Here we do not analyze other aspects of concrete message envelope syntaxes, such as construction or parsing time of a message envelope. The main reason for this is that most of the encoding options used in this evaluation are experimental software, and therefore we believe that

Table 1. Comparison of FIPA message envelope transport encoding options

	Bit-Efficient	IIOP	bXML (w /codes)	bXML (plain)	XML	Object
Case 1	33	153 (464%)	90 (273%)	205 (621%)	346 (1048%)	1421 (4306%)
Case 2	179	337 (188%)	262 (146%)	473 (264%)	671 (375%)	1694 (946%)
Case 3	694	973 (140%)	843 (121%)	1154 (166%)	1844 (266%)	2790 (402%)

the results would not been comparable. We analyze construction and parsing times of ACL messages in the next section. As the encoding options for ACL are similar to those of message envelopes, similar results are expected.

We chose three different message envelopes for this experiment. The first case can be considered as a minimal message envelope. Additionally, the field values are minimal, that is, only a one byte in the most cases. This, obviously, is not a realistic message envelope, but was chosen to demonstrate the relative difference of the additional overhead caused by different encoding schemes. The second one is the same as the first one, but the field values are more realistic. The last case covers all the aspects of the message envelope.

Table 1 gives the number of bytes of message envelope transport syntaxes in three cases using all the selected encoding options. The bit-efficient message envelope is the most compact in all cases. This was expected. However, when the message envelope size increases, the relative difference decreases, especially when compared to IDL and binary XML. The reason for this is, that in the case of big message envelope, the ratio between additional overhead and the message envelope information content (i.e., the field values) increases. None of the selected encoding schemes handles the information content efficiently. As was also expected, the XML encoding and Java object serialization produces big message sizes. But, again, the relative difference decreases when the message envelope size increases. However, for example, in the case of Object serialization, the output size is still about four times bigger than the output size of bit-efficient encoding even in the case of big envelope. The binary-XML encoding shows its power in the case where the information content is small. In these cases using predefined tokens, the output size is much smaller than using the same encoding without predefined tokens. Also, using binary-XML with predefined the output is smaller than in the case of IDL encoding. Without predefined tokens, the output size of binary-XML is bigger than using IDL encoding.

5. Agent Communication Layer

In this section, we analyze the performance of different ACL encoding options. We selected four test cases to find out the performance of different encoding options. These test cases are the same as in message envelope experiment. Then, we selected five alternative methods to encode FIPA-ACL messages, which we compare against the bit-efficient encoding, totaling six different methods for ACL encoding. Firstly, the string-based FIPA-ACL encoding is measured. Especially we use the string encoding as provided by the Jade agent platform. Secondly, a standard XML-based FIPA-ACL encoding is measured. Thirdly, the binary-XML encoding is measured. As with the message envelope experiment, we use binary-XML both with special encoding tokens and without them. Fourthly, the standard Java serialization to output Jade's `ACLMessage` class is measured. Although this method is not a FIPA-compliant way to exchange ACL messages, it is, for example, used in Jade's internal communication when the agents are located on different hosts but belong to the same agent platform (that is, when Java RMI is used). Lastly, as the string-based messages are text information, we also analyze the deflate compression algorithm to compress the ACL messages. The implementation of this algorithm is the one included in JDK1.3 (`DeflaterOutputStream`). Notice that this encoding is not a FIPA-compliant solution. In this case, we also analyzed two different cases. In the first case, the message to compressed in encoded using the string encoding and in the second case using the XML encoding. In both cases, the message stream is reset after each message, which means that the same code table cannot be used in subsequent message. In the measurements, we use Intel Pentium II (366Mhz) laptop (Toshiba Portégé 7020CT) with 128Mb of main memory, Linux 2.2.14, JDK1.3 and Jade 2.3 Agent Platform.

5.1. Results Encoded Output Size

Table 2 shows the results of the output size measurement in bytes. In this case, we use the bit-efficient FIPA-ACL without a dynamic code table. We will analyze the effects of using the code table later. As can be seen in Table 2, the bit-efficient encoding gives the smallest output in all cases, as was expected. However, the difference between binary-XML with special tokens and the bit-efficient encoding is insignificant. Also, the difference between the deflate encoding and the bit-efficient is small. But, neither the deflate encoding nor the binary-XML are a FIPA-compliant solutions, and therefore cannot be used in a general case. The XML encoding output size is about twice as big as the string-based encoding. This was also expected. The serialized `ACLMessage` output size is notably big. This is because the Java serialization outputs the class description to each `ObjectOutputStream` to which the serialized objects are written. However, the class description is output only once to each stream, that is, if two or more objects are written to the same stream, the class description is written only once. In our measurements, we use a different stream for each message, and therefore several class descriptions are needed. While this may seem unfair, it is actually the most common case. For

TABLE 2. The output size in bytes

		Case 1	Case 2	Case 3	Case 4
Bit-efficient	Send	175	161	167	503
	Recv	371	168	1800	2339
bXML	Send	195 (111%)	182 (113%)	188 (113%)	565 (112%)
	Recv	409 (110%)	188 (188%)	2000 (111%)	2597 (111%)
bXML(plain)	Send	353 (202%)	342 (212%)	348 (208%)	1043 (207%)
	Recv	722 (195%)	345 (205%)	3570 (198%)	4637 (198%)
XML	Send	638 (365%)	626 (383%)	632 (378%)	1896 (377%)
	Recv	1294 (348%)	630 (375%)	6420 (357%)	8344 (357%)
String	Send	351 (212%)	339 (211%)	345 (207%)	1035 (206%)
	Recv	720 (194%)	343 (204%)	3550 (197%)	4613 (197%)
String (cmpr)	Send	204 (117%)	203 (126%)	211 (126%)	618 (122%)
	Recv	422 (113%)	208 (124%)	2165 (120%)	2795 (120%)
ACL Object	Send	1408 (805%)	1380 (857%)	1392 (834%)	4172 (829%)
	Recv	2854 (769%)	1394 (830%)	14144 (786%)	18384 (786%)

TABLE 3. Time to construct the messages (in milliseconds)

Bit-efficient		String		String (deflate)		ACL object	
Send	Recv	Send	Recv	Send	Recv	Send	Recv
Case 1							
3.24	4.42	4.22 (130%)	6.30 (143%)	9.40 (290%)	12.64 (286%)	107.62 (3321%)	117.92 (2668%)
Case 2							
3.16	3.35	4.14 (131%)	4.30 (128%)	9.28 (294%)	9.88 (295%)	106.76 (3379%)	106.12 (3168%)
Case 3							
3.32	11.68	4.32 (130%)	21.46 (184%)	9.24 (278%)	38.30 (328%)	106.36 (3204%)	149.82 (1283%)
Case 4							
5.20	14.56	7.94 (152%)	26.98 (199%)	15.68 (301%)	47.86 (329%)	115.64 (2223%)	163.24 (1121%)

example, when using Java RMI, a separate `ObjectOutputStream` has to be created for each invocation.

5.2. Constructing and Parsing Messages

In the second measurement, we analyze how long it takes to construct the output for different encoding schemes. The XML-based encoding schemes are left out in these measurements, as the parsers for these are too experimental for the results being comparable to other encoding schemes. Table 3 provides the results of these measurements. Each test is repeated 50 times and results (averages) are given in milliseconds. In these measurements we first create a Jade `ACLMessage` object of a

Table 4. Time to parse the messages (in milliseconds)

Bit-efficient		String		String (deflate)		ACL object	
Send	Recv	Send	Recv	Send	Recv	Send	Recv
Case 1							
16.14	18.32	24.70 (153%)	31.08 (170%)	27.48 (170%)	40.48 (220%)	144.88 (898%)	151.58 (827%)
Case 2							
16.04	15.60	24.66 (154%)	24.84 (159%)	27.74 (173%)	27.60 (177%)	143.68 (896%)	144.38 (926%)
Case 3							
15.42	41.24	24.88 (161%)	93.08 (226%)	27.68 (180%)	186.76 (453%)	144.02 (934%)	211.22 (512%)
Case 4							
20.86	49.40	36.36 (174%)	125.98 (255%)	52.72 (252%)	262.98 (532%)	158.52 (759%)	233.28 (472%)

FIPA-ACL message and then generate the encoded output of the message from this object. The time to create the `ACLMessage` object is not calculated in the results. As can be seen in Table 3, the bit-efficient encoding is the fastest in all cases, but the difference to the string-based encoding is insignificant. This was expected, since creating a string-based FIPA-ACL message is just outputting strings; there is little to optimize. The low performance of the deflate algorithm is due to a fact that after uncompressing the message, it still have to be parsed in order to create a Java object. This phase is included in the process of parsing the bit-efficient encoding. Furthermore, the deflate algorithm gives a slightly larger output than the bit-efficient encoding scheme (see Table 2). Creating serialized objects is also surprisingly slow. The reason for this is that creating a new `ObjectOutputStream` is a slow operation.

In the third measurement, we measure the parsing time of an encoded message, that is, how long it takes to create a Jade `ACLMessage` object from an encoded stream. In all cases, the data is first read into a memory buffer, and the time needed for this is excluded in the results. Table 4 gives the results of this measurement. Again, the bit-efficient encoding is the fastest. Further, in this measurement it is much faster than any other encoding scheme we measured. The main reasons for this are that (1) a very few string comparisons are needed to parse the message and that (2) our bit-efficient FIPA-ACL implementation, instead of allocating new memory, tries to reuse already allocated memory whenever possible. The method is an efficient method for optimizing Java programs.

5.3. Effects of Dynamic Code Table in Bit-efficient ACL

In all the cases analyzed above we used the bit-efficient FIPA-ACL encoding without the dynamic code table. Before the measurements, we believed that using the dynamic code table should give a better compression ratio, but the code table

TABLE 5. Number of bytes using different cache sizes

No cache		2^8		2^9		2^{10}		2^{15}	
Send	Recv	Send	Recv	Send	Recv	Send	Recv	Send	Recv
Case 1									
175	371	175 (100%)	249 (67%)	175 (100%)	257 (69%)	175 (100%)	257 (69%)	175 (100%)	257 (69%)
Case 2									
161	168	161 (100%)	168 (100%)	161 (100%)	168 (100%)	161 (100%)	168 (100%)	161 (100%)	168 (100%)
Case 3									
167	1800	167 (100%)	792 (44%)	167 (100%)	864 (48%)	167 (100%)	864 (48%)	167 (100%)	864 (48%)
Case 4									
503	2339	354 (70%)	1063 (45%)	364 (72%)	1152 (49%)	364 (72%)	1152 (49%)	364 (72%)	1152 (49%)

management might slow down both constructing the output and parsing the input as the code table is implemented in Java. However, as the result will show, the code table management slows down neither constructing time nor parsing time.

First, we analyze the size of the encoded message. As can be seen in Table 5, using the code table provides a more compact output, but only if there are enough messages to encode. This can be seen especially in the Case 4, where the coding scheme without the code table provides 2339 bytes of output in incoming traffic, while using the code table provides 1063 bytes of output. Using the code table with a larger size than 2^8 gives a slightly larger output, because of the two-byte cache indexes. However, when encoding a large number of messages, it is expected that using a larger code table give a more compact output.

Next we analyze how long it takes to construct the encoded output using different cache sizes. Table 6 shows the results of this measurement. A coding scheme without a code table is fastest when having only one or at most a few messages. This was expected, since when the code table is used, the encoder tries to find every string in the code table, which takes some time. However, when there are several messages and the encoder actually finds something in the code table, the process of constructing messages becomes faster. The reason for this is that when the encoder should output a string to the encoded message, it must copy it there, while if the string is found in the code table, it only has to output the corresponding index to the encoded message (one or two byte(s)). Similar results are also achieved when the parsing time is measured (see Table 7). The difference, however, is less significant than in constructing messages. The reason for this is that the code table lookups are much faster when decoding the message.

TABLE 6. Time to create messages using different cache sizes (in milliseconds)

No cache		2^8		2^9		2^{10}		2^{15}	
Send	Recv	Send	Recv	Send	Recv	Send	Recv	Send	Recv
Case 1									
3.40	4.40	4.12 (121%)	4.96 (112%)	4.22 (124%)	4.98 (113%)	4.28 (125%)	5.10 (116%)	4.20 (124%)	5.24 (119%)
Case 2									
3.30	3.36	4.10 (124%)	4.24 (126%)	4.14 (125%)	4.18 (124%)	4.12 (125%)	4.14 (123%)	4.08 (124%)	4.24 (126%)
Case 3									
3.28	11.78	4.24 (129%)	9.92 (84%)	4.18 (127%)	9.96 (85%)	4.20 (128%)	9.98 (85%)	4.12 (126%)	10.00 (85%)
Case 4									
5.16	14.54	5.88 (114%)	12.06 (83%)	5.98 (116%)	12.16 (84%)	6.00 (116%)	12.18 (84%)	5.84 (113%)	12.14 (83%)

TABLE 7. Time to parse messages using different cache sizes (in milliseconds)

No cache		2^8		2^9		2^{10}		2^{15}	
Send	Recv	Send	Recv	Send	Recv	Send	Recv	Send	Recv
Case 1									
13.78	15.14	13.84 (100%)	14.68 (97%)	13.90 (101%)	14.76 (97%)	13.88 (101%)	14.64 (97%)	13.94 (101%)	14.74 (97%)
Case 2									
13.80	13.88	13.88 (101%)	13.94 (100%)	13.94 (101%)	13.88 (100%)	13.90 (101%)	13.90 (100%)	13.82 (100%)	13.90 (100%)
Case 3									
13.80	25.12	13.92 (101%)	19.26 (71%)	14.00 (101%)	19.64 (78%)	13.90 (101%)	19.78 (79%)	13.92 (101%)	19.76 (79%)
Case 4									
16.14	28.88	15.56 (96%)	21.68 (75%)	15.70 (97%)	22.06 (76%)	15.72 (97%)	22.26 (77%)	15.74 (98%)	22.16 (77%)

6. Content Language Layer

A content language is used to express the actual content of a communication between agents. Each language specified in the FIPA-CLL [14] has only one concrete transport encoding syntax. Further, in each case either a string s-expression or XML is used. Given this, they are not in general suitable for environments where slow wireless links are involved. Obviously, having an efficient encoding of the message envelope and the FIPA-ACL does not help much, if the actual message content is expressed using a verbose encoding.

In following, we explore different options for encoding FIPA-SL and FIPA-CCL. We believe that the results can be generalized to any similar content language. In this experiment, we evaluate only the output size of different encoding options. Other features, such as the parsing time, are excluded because lack of mature enough implementations.

6.1. The Case of FIPA-SL

For FIPA-SL content language [23] evaluation we choose four different encoding schemes for FIPA-SL. Firstly, the standard s-expression is evaluated. This encoding option is the only one specified by FIPA. Secondly, we use deflate algorithm to encode the s-expression syntax. As the s-expression syntax is string, we believe that this option can give good results. Thirdly, we use XML encoding. Lastly, we use binary-XML. As with the message envelope encoding and FIPA-ACL encoding, we use binary-XML both with and without special encoding tokens.

For the experiment, we choose three FIPA-SL expressions, that is, message contents. The first two are simple and typical messages used with communication with the AMS. The third expression is a somewhat more complicated and contains more data than the other expression.

Table 8 shows the results of the output size measurements in bytes. The s-expression encoding and binary-XML with special tokens gives a similar performance; binary-XML being slightly better. Although s-expression encoding is plain text encoding, it does not contain that much additional overhead. On the other hand, the source format, that is XML, for binary-XML is so verbose, that even the binary version cannot produce small output. The output of the XML encoding is the largest, as was expected. The deflate algorithm gives better output if the message to encode is large enough. But even if the message is small, the deflate algorithm is only a slight worse than binary-XML and s-expression encoding schemes. Given this, it seems that using the deflate algorithm to encode the message content is the best solution when sending messages over a (slow) wireless link. Obviously, the deflate algorithm needs more processing power than the other options, because after the message content is decompressed, it still have to be parsed. Therefore, if the processing power is limited, the s-expression seems to be the best solution, assuming that the message content is relatively small.

6.2. The Case of FIPA-CCL

For FIPA-CCL content language [12] evaluation, we choose the same options as with the case of FIPA-SL. However, the s-expression encoding is excluded, as there is no s-expression syntax for FIPA-CCL. Obviously, we could define such syntax, but there is no real reason for doing so. Therefore, the encoding options we will use in the FIPA-CCL experiment are XML, binary-XML (with and without special encoding tokens), and deflated XML.

For the experiment, we choose three FIPA-CCL expressions, which are not semantically the same as in the case of FIPA-SL experiment. We could not use the same expression as in the case of FIPA-SL experiment, because these two

TABLE 8. Comparison of selected FIPA-SL encoding options in number of bytes

	S-Expression	bXML	bXML (plain)	Deflate	XML
Example 1	222	172 (77%)	303 (136%)	224 (101%)	558 (251%)
Example 2	229	177 (77%)	314 (137%)	232 (101%)	572 (250%)
Example 3	682	661 (97%)	865 (127%)	378 (55%)	2275 (334%)
Total	1133	1010 (89%)	1482 (131%)	456 (40%)	3405 (301%)

TABLE 9. Comparison of selected FIPA-CCL encoding options in number of bytes

	bXML	bXML(plain)	Deflate	XML
Example 1	335	548 (164%)	297 (89%)	885 (264%)
Example 2	433	676 (156%)	340 (79%)	1125 (260%)
Example 3	418	679 (162%)	336 (80%)	1122 (268%)
Total	1186	1903 (160%)	486 (41%)	3132 (264%)

content languages are developed for different purposes. The results of the output size measurements in bytes of the FIPA-CCL experiment are given in Table 9. The results are similar to those of the FIPA-SL experiment. The deflate algorithm and binary-XML with special tokens gives similar output. The binary-XML without special tokens is slightly worse and plain XML encoding is much worse than any other option.

7. Conversation Layer

Ongoing conversations between agents often fall into typical patterns, which can be described as a series of states linked by transitions. Given the certain state of a conversation, the participants can send and/or expect only certain messages. These patterns of message exchange are called interaction protocols [17]. FIPA has defined several interaction protocols, including simple ones such as FIPA Request [22] and FIPA Query [21], and more complicated ones such as FIPA Contract Net [18] and FIPA Auction English [16].

The use of interaction protocols eases the agent implementation, especially, when an agent is performing tasks that are irrelevant in achieving its goal, such

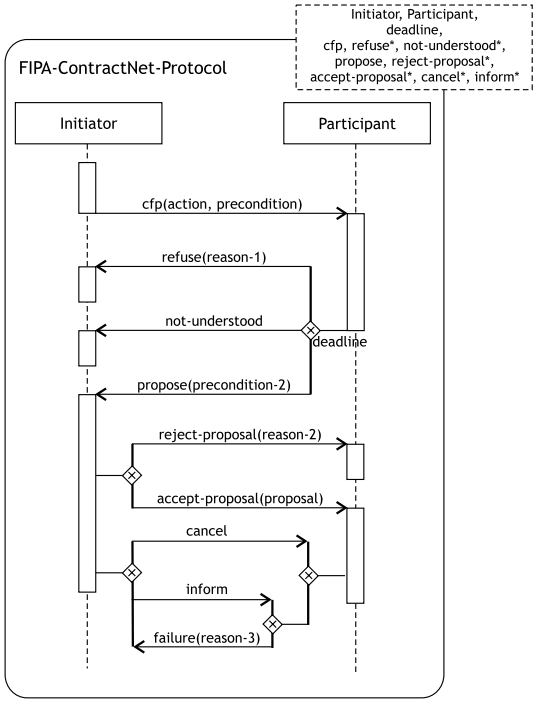


FIGURE 7. FIPA Contract Net interaction protocol

as the registration into a management system or into a directory service. In these cases, by carefully following the interaction protocol, an agent does not have be "smart" to take care of necessary administrative tasks.

The FIPA Contract Net protocol [15] is FIPA's version of the most well-known task sharing protocol called contract net [35]. In the FIPA Contract Net interaction protocol the initiator (contractor) sends a "call for proposals" (CFP) to several participants (contractees) requesting proposals to perform a given action (see Figure 7). The participants send their proposals back to the contractor. From these proposals, the contractor selects the most desired one, and sends an ACCEPT-PROPOSAL message to the sender of the selected proposal and a REJECT-PROPOSAL message to the others. Furthermore, the contractor can define a timeout for how long it will wait for proposals. If it does not receive all proposals in this time, it selects one from the received ones and rejects all subsequent proposals. Finally, the selected contractee sends an INFORM message to the contractor once it has performed the requested action.

Now, let us assume that the contractor resides at the mobile node, and n of the contractees resides in the fixed network. To finish the protocol, about $n * 3$ messages are sent over the wireless link. If we assume that one message contains

2 kilobytes of data and n is 10, more than 60 kilobytes of data is transferred over the link to accomplish the protocol. Having a slow link, such as a GSM data link, it takes more than one minute just to send these messages.

Interaction protocols could be improved for wireless environments. The main idea here is to reduce messages sent over wireless communication path. For example, the contractor can nominate a proxy agent in a fixed network to accomplish the interaction protocol, or using mobile agent technology the contractor can itself migrate to the fixed network and communicate with contractees over the fixed network. In both cases, some additional data is sent over the link. Nominating another agent to accomplish the interaction protocol needs some messages, and if the contractor migrates over the link, its code and state is transferred over the link.

The FIPA Propose protocol [20] was defined with slow wireless links in mind. In a way, it is a version of FIPA Contract Net protocol, where the CFP communicative act as well as the last INFORM communicative act are removed. In this protocol, the initiator agent sends a PROPOSE communicative act to the participant agent proposing that it (the initiator) will perform some action. The participant agent may either accept (ACCEPT-PROPOSAL) or reject (REJECT-PROPOSAL) this proposal.

The interaction protocol in a nomadic environment can be selected based on the current situation. For example, having a low-bandwidth connection, an agent can choose an interaction protocol that requires modest bandwidth, but therefore produces only sub-optimal results. Alternatively, when using more bandwidth is possible, an agent can choose an interaction protocol that requires more bandwidth and thereby produces better results. This selection, however, involves a careful analysis of the protocol; how many round-trips are necessary and how much data is needed. Additionally, some of this analysis must be done at the runtime, as it is impossible in general to predict the way possible opponents act.

8. Related Work

LEAP (Lightweight Extensible Agent Platform) [2] was the first FIPA compliant agent platform running on PDAs and mobile phones. For agent communication in wireless environments, the LEAP platform provides a protocol called JICP [3] for intra-platform communication. This protocol seems to be efficient in number of overhead bytes, but on the other hand, this protocol—even though designed for unreliable wireless communication paths—provides insufficient reliability. For example, messages might get duplicated during an unexpected disconnection, that is, the same message may get delivered to the ultimate destination more than once.

MicroFIPA-OS [32] is an agent development toolkit and platform based on the FIPA-OS toolkit. This system targets at medium to high-end PDA devices that have sufficient resources to execute PersonalJava compatible virtual machine.

The MicroFIPA-OS architecture is extensible by plugging in components that either replace or extend the architecture. An example of this kind of contribution is FIPA Nomadic Application Support [19], which provides support for wireless environments, including components for efficient message transport over slow wireless communication paths [31]. For example, The FIPA Nomadic Application Support incorporates the bit-efficient envelope and ACL messages discussed earlier.

Yet another example of providing an agent platform to wireless environments is A-Globe [34]. Unlike LEAP and MicroFIPA-OS, A-Globe is not FIPA compliant agent platform. However, this relaxation gives more freedom to design components for wireless communication and therefore more efficient solution can be made. The obvious drawback is that agent on A-Globe platform cannot directly communicate with agent residing on LEAP or MicroFIPA-OS platforms.

Several other attempts have been developed in order to enable agents in small devices (e.g., PDAs and mobile phones). However, very seldom the properties of wireless communication paths are taken appropriately into account, but these systems rely on communication solutions designed for reliable and fast wireline connections.

Another option for agent communication is to use Web Services standards for delivering ACL messages between agents [28]. However, when considering wireless communication paths, the same problems as with FIPA-style communication will remain (see for example [30]).

9. Conclusions

We performed a performance analysis of agent communication in wireless environments. At the lowest layer—transport and signaling layer—the agent communication should not be different from the communication in other distributed systems; hence we gave only a brief overview of this layer's issues. At the MTP layer, we examined the MTPs specified by FIPA and provided an exhaustive performance evaluation of various protocols. Further, we have designed and implemented a MTP called MAMAv2, which performs well in slow wireless networks. At the message envelope layer and the ACL layer, the most important factor in nomadic environments is efficiency, assuming that the MTP layer provides sufficient reliability as it should. We compared standard message envelope transport encoding options, and concluded that the bit-efficient encoding is the most efficient in number of bytes. The bit-efficient envelope encoding scheme is designed and implemented by us. The XML envelope syntax, as was expected, was the most verbose syntax. Similar comparison was made with ACL transport encoding options, providing similar results. Furthermore, we showed that bit-efficient ACL transport encoding is not only more space-efficient but also more efficient to process. For example, parsing bit-efficiently encoded messages is faster than parsing any other standard transport encoding. Space-efficiency is naturally an important feature in nomadic environments, but faster handling of messages becomes important when either the

processing power is limited or a great deal of messages should be handled. The former is true in today's low-end mobile devices and the latter can be expected to happen in the future when agent technology is employed on a large scale. The bit-efficient ACL encoding scheme is designed and implemented by us, and it is freely available for Jade agent platform. Finally, we performed a similar performance analysis of two content languages, namely FIPA-SL and FIPA-CCL. For these languages, we have designed and implemented binary-XML encoding schemes,

References

[1] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE — A FIPA-compliant agent framework. In *Proceedings of the 4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*, pages 97–108, London, UK, 1999. The Practical Application Company Ltd.

[2] Federico Bergenti, Agostino Poggi, Bernard Burg, and Giovanni Caire. Deploying FIPA-compliant systems on handheld devices. *IEEE Internet Computing*, 5(4):20–25, 2001.

[3] Giovanni Caire, Nicolas Lhuillier, and Giovanni Rimassa. A communication protocol for agents on handheld devices. In *Workshop on Ubiquitous Agents on Embedded, Wearable and Mobile Devices*, Bologna, Italy, July 2002.

[4] Foundation for Intelligent Physical Agents. *FIPA ACL Message Representation in Bit-Efficient Specification*. Geneva, Switzerland, October 2000. Specification number XC00069.

[5] Foundation for Intelligent Physical Agents. *FIPA ACL Message Representation in String Specification*. Geneva, Switzerland, November 2000. Specification number XC00070.

[6] Foundation for Intelligent Physical Agents. *FIPA ACL Message Representation in XML Specification*. Geneva, Switzerland, October 2000. Specification number XC00071.

[7] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Envelope Representation in Bit Efficient Specification*. Geneva, Switzerland, November 2000. Specification number XC00088.

[8] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Envelope Representation in XML Specification*. Geneva, Switzerland, November 2000. Specification number XC00085.

[9] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Protocol for HTTP Specification*. Geneva, Switzerland, October 2000. Specification number XC00084.

[10] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Protocol for IIOP Specification*. Geneva, Switzerland, November 2000. Specification number XC00075.

[11] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Protocol for WAP Specification*. Geneva, Switzerland, October 2000. Specification number XC00076.

[12] Foundation for Intelligent Physical Agents. *FIPA CCL Content Language Specification*. Geneva, Switzerland, October 2000. Specification number XC00009.

[13] Foundation for Intelligent Physical Agents. *FIPA Communicative Act Library Specification*. Geneva, Switzerland, November 2000. Specification number XC00037.

[14] Foundation for Intelligent Physical Agents. *FIPA Content Languages Specification*. Geneva, Switzerland, October 2000. Specification number XC00007.

[15] Foundation for Intelligent Physical Agents. *FIPA Contract Net Interaction Protocol Specification*. Geneva, Switzerland, October 2000. Specification number XC00029.

[16] Foundation for Intelligent Physical Agents. *FIPA English Auction Interaction Protocol Specification*. Geneva, Switzerland, October 2000. Specification number XC00031.

[17] Foundation for Intelligent Physical Agents. *FIPA Interaction Protocol Library Specification*. Geneva, Switzerland, October 2000. Specification number XC00025.

[18] Foundation for Intelligent Physical Agents. *FIPA Iterated Contract Net Interaction Protocol Specification*. Geneva, Switzerland, October 2000. Specification number XC00030.

[19] Foundation for Intelligent Physical Agents. *FIPA Nomadic Application Support Specification*. Geneva, Switzerland, November 2000. Specification number XC00014.

[20] Foundation for Intelligent Physical Agents. *FIPA Propose Interaction Protocol Specification*. Geneva, Switzerland, October 2000. Specification number XC00036.

[21] Foundation for Intelligent Physical Agents. *FIPA Query Interaction Protocol Specification*. Geneva, Switzerland, October 2000. Specification number XC00027.

[22] Foundation for Intelligent Physical Agents. *FIPA Request Interaction Protocol Specification*. Geneva, Switzerland, October 2000. Specification number XC00026.

[23] Foundation for Intelligent Physical Agents. *FIPA SL Content Language Specification*. Geneva, Switzerland, November 2000. Specification number XC00008.

[24] Foundation for Intelligent Physical Agents. *FIPA Messaging Interoperability Service Specification*. Geneva, Switzerland, August 2001. Specification number PC00093.

[25] Rich Fritzson, Tim Finin, Don McKay, and Robin McEntire. KQML — A language and protocol for knowledge and information exchange. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, pages 126–136, Seattle, WA, USA, July 1994.

[26] Heikki Helin. Supporting nomadic agent-based applications in FIPA agent architecture. PhLic. Thesis, Series of Publications C, Number C-2001-63, University of Helsinki, Department of Computer Science, Helsinki, Finland, December 2001.

[27] Heikki Helin and Stefano Campadello. Providing messaging interoperability in FIPA communication architecture. In Krysztof Zieliński, Kurt Geihs, and Aleksander Laurentowski, editors, *New Developments in Distributed Applications and Interoperable System. Proceedings of the Third IFIP TC6/WG6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS'01)*, pages 121–126, Krakow, Poland, September 2001. Kluwer Academic Publishers.

[28] Michael N. Huhns. Agents as web services. *IEEE Internet Computing*, 6(4):93–95, 2002.

[29] Markku Kojo, Andrei Gurtov, Jukka Manner, Pasi Sarolahti, Timo Alanko, and Kimmo Raatikainen. Seawind: A wireless network emulator. In *Proceedings of 11th*

GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems, Aachen, Germany, September 2001.

[30] Mikko Laukkanen and Heikki Helin. Web services in wireless networks—what happened to the performance? In Liang-Jie Zhang, editor, *Proceedings of the International Conference on Web Services (ICWS'03)*, pages 278–284, Las Vegas, USA, June 2003. CSREA Press.

[31] Mikko Laukkanen, Heikki Helin, and Heimo Laamanen. Supporting nomadic agent-based applications in the FIPA agent architecture. In Cristiano Castelfranci and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2002)*, pages 1348–1355, Bologna, Italy, July 2002.

[32] Mikko Laukkanen, Sasu Tarkoma, and Jani Leinonen. FIPA-OS agent platform for small-footprint devices. In John-Jules Meyer and Milind Tambe, editors, *Intelligent Agents VIII, Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 447–460. Springer-Verlag: Heidelberg, Germany, 2002.

[33] G. Montenegro, S. Dawkins, M. Kojo, V. Magret, and N. Vaidya. Long thin networks. Request for Comments 2757, January 2000.

[34] David Šišlák, Milan Rollo, and Michal Pěchouček. *A*-globe: Agent platform with inaccessibility and mobility support. In Matthias Klusch, Sascha Ossowski, Vipul Kashyap, and Rainer Unland, editors, *Cooperative Information Agents VIII*, pages 199–214, 2004.

[35] R. G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[36] Wireless Application Protocol Forum. *Binary XML Content Format Specification*, November 1999. Version 04-Nov-1999.

Information about Software

Software is available on the Internet as

( ) prototype version
(•) full fledged software (freeware), version no.: N/A
( ) full fledged software (for money), version no.:
( ) Demo/trial version
( ) not (yet) available

Internet address:

Description of software: Bit-efficient encoding of ACL messages for Jade
Download address: `http://jade.tilab.com/`

Heikki Helin and Mikko Laukkanen
TeliaSonera Finland
P.O.Box 970
FIN-00051 Sonera
Finland
e-mail: `Heikki.J.Helin@teliasonera.com, Mikko.Laukkanen@teliasonera.com`