Efficient Agent Communication in Slow Wireless Networks

HEIKKI HELIN, MIKKO LAUKKANEN
Sonera Corporation, Corporate R&D
P.O.Box 970, FIN-00051 Sonera
FINLAND
{ }

Abstract: – We discuss agent communication in wireless environment according to FIPA's communication model. In wireless environments, the agent communication should be tailored in order to provide an efficient use of scarce and fluctuating data communication resources. In some cases, efficiency is not so important an aspect as, for example, reliability. Nevertheless, the communication solutions used in the modern distributed systems or the agent-based systems seldom fulfil these requirements.

Key-Words: – Agent communication, Wireless environment, FIPA

1 Introduction

In multi-agent systems, communication is an essential component. To exchange their knowledge, agents should be able to communicate with each other. In the lower layers of communication, the agent communication does not necessarily differ from the communication in traditional distributed systems. In fact, the same transport protocols and messaging techniques as in modern distributed systems should be used. From the lower-layer's point of view, agents are just sending data. What makes the software agent communication different from the one in the traditional distributed systems is the use of the agent communication languages (ACLs). In this paper we are not interested in why agents want to communicate, but assume that they will communicate, using some ACL, and that at least part of the communication path is implemented using wireless technology.

In wireless environments, the agent communication should be tailored in order to provide an efficient use of scarce and fluctuating data communication resources. In some cases, efficiency is not so important an aspect as, for example, reliability. Nevertheless, the communication solutions used in the modern distributed systems or the agent-based systems seldom fulfil these requirements. In this paper we analyze the FIPA[1] communication model when employed in the wireless environment. Especially we concentrate on an efficient ACL communication and analyze the various options for concrete encoding of ACL messages that FIPA has specified.

Figure 1 depicts a layered model of the FIPA agent communication. In wireless environments, the agents need to communicate efficiently and the communication should be reliable. Therefore, the communication stack given in Figure 1 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 a high-latency communication path is used. It is important to notice, that "minimal" here does not mean the absolute minimal value; sometimes it is better to use more round-trips to achieve a better end-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. This means that usually a binary encoding should be used instead of an ASCII-based. The MTP should be able to transfer messages over a wireless link reliably and efficiently. The selection of a message transport protocol also typically affects the selection of a transport protocol. For example, if the transport protocol is reliable also in wireless environments, the implementation of the MTP can be much simpler. Typically, however, this is not the case, and therefore reliability should be implemented into the MTP.

The rest of this paper is organized as follows. Section 2 discusses lower-level transport issues concentrating on reliability. In Section 3 we analyze

[1]Foundation for Intelligent Physical Agents; `http://www.fipa.org`.

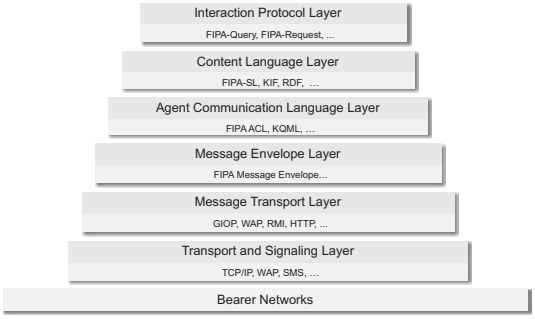

Fig. 1: A layered model of agent communication

FIPA-ACL messaging and provide results of extensive study on different concrete FIPA-ACL encodings. Finally, Section 4 concludes the paper.

2 Transport and Envelope Layers

The message exchange over a network is needed when communicating agents are located on different hosts in the network. Low-level issues related to transferring data over the network is not itself an agent-related problem. However, without efficient and reliable delivery of data, implementing efficient and reliable agent communication—especially in wireless environments—is impossible.

TCP is perhaps the most widely used transport protocol, in agent systems as well. In most cases, TCP is used indirectly. For example, the IIOP and HTTP protocols used in FIPA use TCP as the transport protocol It is well known that TCP performs poorly in slow wireless networks [5], and thus gives a poor basis for MTP employing it. In addition, there are other problems related to IIOP when used in the wireless environment (see for example [4]).

FIPA has specified that WAP can be used as a baseline MTP in the wireless environment. Should the WAP be used, it is up to the implementation and the environment, which protocol is used as the transport protocol. Further, depending on the MTP, proprietary transport protocols may be possible. For example, there are several transport protocols for replacing TCP in the wireless environment. While these protocols typically provide efficient and reliable data transfer, using proprietary protocols may deteriorate the interoperability.

An important function of the MTP is to provide reliable message delivery. Neither IIOP nor HTTP provide sufficient reliability, although at-least-once message delivery semantics can be implemented fairly easily. However, this functionality is not specified by FIPA, and therefore it is not a generic solution. WAP is the only MTP specified by FIPA that provides the system with sufficient reliability. However, it is important to note that OMG is currently developing an environment specific GIOP mapping for wireless environments [6]. This protocol provides reliable delivery of IIOP messages in wireless environments.

The purpose of the envelope layer is to enable transport protocol-independent message handling, for example routing of messages without touching the ACL. FIPA has defined three concrete message envelope syntaxes [3]. In the IIOP MTP, the message envelope is “built-in” in the MTP, that is, the IDL interface defines the structure of the message envelope. The XML-based syntax for the message envelope is designed to complement the XML-based syntax for FIPA-ACL and HTTP MTP. The third syntax, bit-efficient, is meant for the wireless environment.

Figure 2 gives the number of bytes of message envelope transport syntaxes in two cases. The first case can be considered as a minimal message envelope, that is, it contains only the mandatory fields of a message envelope. The second one is perhaps a more typical message envelope. As the XML syntax is based on ASCII strings, the number of bytes in the second case is as large as two kilobytes. However, the dominating factor in both cases is the actual content of the message envelope, that is, the strings used as agent-identifiers, addresses, etc. Even the bit-efficient message envelope encoding handles these inefficiently, as it decodes them as strings. Therefore, if the content of an envelope is huge, the selection of transport syntax does not matter. However, we believe that the content of a typical envelope is fairly small. Here we do not analyze other aspects of concrete message envelope syntaxes, such as construction or parsing time of a message envelope. In the next section, we will analyze the bit-efficient ACL more thoroughly; similar results of a bit-efficient message envelope can be expected.

3 ACL Layer

The agent communication language layer defines the outer language used in communication. Here we

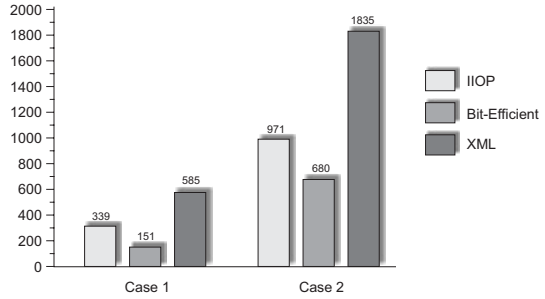

Fig. 2: Number of bytes in the message envelope using standard FIPA encoding schemes

concentrate on one particular ACL: FIPA-ACL. For FIPA-ACL three standard encoding schemes have been specified [2]: String-based, XML, and bit-efficient. Here we compare bit-efficient FIPA-ACL to other standard encoding schemes as well as to some non-standard solutions.

In bit-efficient FIPA-ACL there are two primary ways to reduce the transfer volume over the wireless link: tokenized syntax and use of dynamic code table. First, FIPA-ACL messages are encoded efficiently by using one octet codes for predefined message parameters and other common parts of messages. This is a significant improvement compared to a simple string-based coding, as it in most cases reduces additional overhead to half of the original. By additional overhead we mean the difference between the encoding size and the size of minimum-cost encoding. Furthermore, this improvement is easy to implement and is generally faster to parse than the string-based coding—comparing bytes is much faster than comparing strings. Although the tokenized syntax gives a significant improvement compared to string-based encodings, the true power of bit-efficient FIPA-ACL lies in the use of dynamic code table. In this encoding scheme similar parts of subsequent messages are not transmitted multiple times over the communication channel, but subsequent occurrences are replaced by short codes. Intelligent caching, however, requires a tight coupling between communicating peers, and thus is inapplicable in some situations. Furthermore, implementation of intelligent caching requires more memory and processing power than a simple data reduction scheme.

For the analysis, we selected four test cases in which we analyze the size of the output and both the time it takes to construct a message and the time it takes to parse a message. In the first case, we use fipa-request interaction protocol in which the initiator sends one message (REQUEST) and the participant two (AGREE and INFORM). In the second case, we use fipa-query interaction protocol. In this case, two messages are sent—one by both the initiator (QUERY-REF) and the participant (INFORM). In the third case, fipa-subscribe interaction protocol is used. The initiator sends the SUBSCRIBE message and the participant replies with 10 INFORM messages. The last case is combination of the first three.

Four alternative methods—string-based, XML, serialized Java object, and compressed string—is selected to encode FIPA-ACL messages which are compared against the bit-efficient encoding. The last two schemes are not standard methods to encode messages. However, serialized Java objects are used, for example, in Jade's internal communication when the agents are located on different hosts but belong to same agent platform (i.e., when Java RMI is used). Further, as string-based messages are text information, using text compression is simplest way to encode them efficiently, and therefore we wanted to analyze it as well.

All the experiments are conducted in the Linux environment using the Jade agent platform [1] (JDK1.3), hardware platform being Intel 366Mhz procecessor and 128 Mb of memory. Our bit-efficient FIPA-ACL implementation used in the measurements is a full implementation of FIPA's standard. It is a 100% Java implementation, and can be used both with Jade [1] and FIPA-OS [7]. The platform is selected at in compile time; thus, there is no additional "multi-platform" overhead at runtime.

3.1 Analysis

Table 1 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, the bit-efficient encoding gives the smallest output in all cases, as was expected. However, the difference between compressed string encoding and bit-efficient encoding is insignificant. The XML encoding output size is about twice as big as the string-based encoding. This also was ex-

Table 1: The output size in bytes

	Bit-efficient		XML		String		String (cmpr)		ACL object	
	send	recv	send	recv	send	recv	send	recv	send	recv
Case 1	175	371	638 (365%)	1294 (348%)	351 (212%)	720 (194%)	204 (117%)	422 (113%)	1408 (805%)	2854 (769%)
Case 2	161	168	626 (383%)	630 (375%)	339 (211%)	343 (204%)	203 (126%)	208 (124%)	1380 (857%)	1394 (830%)
Case 3	167	1800	632 (378%)	6420 (357%)	345 (207%)	3550 (197%)	211 (126%)	2165 (120%)	1392 (834%)	14144 (786%)
Case 4	503	2339	1896 (377%)	8344 (357%)	1035 (206%)	4613 (197%)	618 (122%)	2795 (120%)	4172 (829%)	18384 (786%)

pected. The serialized `ACLMessage` output size is notably big. This is due to the fact that the Java serialization outputs the class descriptions to each `ObjectOutputStream` to which the serialized objects are written. This could be optimized by using one stream for several objects. However, this is not the way how for example Java RMI uses streams.

In the second test, we analyze how long does it take to construct the output for different encodings[2]. Table 2 provides the results of these measurements. Each test is repeated 50 times and results (averages) are given in milliseconds. In these tests we first create a Jade `ACLMessage` object of an 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 final results. The bit-efficient encoding is the fastest in all cases, but the difference to string-based encoding is insignificant. This was expected, since creating string-based FIPA-ACL message is basically just outputting strings; there is not much to optimize. Surprising in these results is the low performance of the zip algorithm. The algorithm we employ in our test is implemented using native code instead of Java, and therefore it should be faster. Furthermore, the compression algorithm gives a slightly larger output than the bit-efficient encoding scheme. Creating serialized objects is also surprisingly slow. The reason for this is that creating a new `ObjectOutputStream` is a very slow operation.

In the third test, we measure the parsing time of an encoded message, that is, how long does it take to create a Jade `ACLMessage` object from an encoded stream. In all cases, the data is first read into memory buffer, so the actual reading time is not included in the results. Table 2 gives the results of this test. Again, the bit-efficient encoding is the fastest, and this time it is much faster than any other encoding scheme we measured. The main reasons for this are that (1) very few string comparisons are needed to parse the message and that (2) the bit-efficient FIPA-ACL implementation, instead of allocating new memory, tries to reuse already allocated memory whenever possible.

[2]The XML encoding is left out in these tests, as the current version of Jade does not support it.

3.2 Effects of Dynamic Code Table

In all the tests analyzed above we used the bit-efficient FIPA-ACL encoding without the dynamic code table. Before the tests, we believed that using the dynamic code table should give better compression ratio, but the code table management might slow down both constructing output and parsing input as the code table is implemented in Java. However, as the result will show, the code table management slows down neither message constructing time nor parsing time.

First, we analyze the size of the encoded message. As can be seen in Table 3, using 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 lerger 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 gives a more compact output.

Table 2: Time to construct and parse the messages (milliseconds)

	Bit-efficient		String		String (cmpr)		ACL object	
	send	recv	send	recv	send	recv	send	recv
Message construction								
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%)
Message parsing								
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%)

Table 3: 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	249	175	257	175	257	175	257
Case 2	161	168	161	168	161	168	161	168	161	168
Case 3	167	1800	167	792	167	864	167	864	167	864
Case 4	503	2339	354	1063	364	1152	364	1152	364	1152

Next we analyze how long it takes to construct the encoded output using different cache sizes. Table 4 shows the results of this test. A coding scheme without a code table is the fastest when there is 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 from the code table, which takes some time. However, when there are several messages and the encoder actually finds something from 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 from the code table, it only has to output the corresponding index to the encoded message. Similar results are also achieved when the parsing time is measured (see Table 4). The difference is less significant that in constructing messages. The reason for this is that the code table lookups are much faster when decoding the message.

As a conclusion, the bit-efficient encoding utilizes the communication path more efficiently than other encodings. This is especially true when the encoding is based on ASCII strings. Furthermore, even if the bit-efficient FIPA-ACL encoding is designed for slow wireless links, it can also be used when high-speed agent communication is needed, as the handling (i.e., parsing) of a tokenized syntax is typically faster than the handling of string-based syntaxes. For example, parsing bit-efficiently encoded messages is more than two times faster than parsing string encoded messages (see Table 2).

4 Conclusions and Future Work

We presented the layered model of the FIPA agent communication, and analyzed thoroughly the bit-efficient encoding of FIPA-ACL messages, and showed that this encoding is not only more space-efficient but also more efficient to deal with. Space-

Table 4: Time to create and parse messages using different cache sizes (milliseconds)

	No cache		2^8		2^9		2^{10}		2^{15}	
	send	recv	send	recv	send	recv	send	recv	send	recv
Message construction										
Case 1	3.24	4.42	3.84	4.40	3.80	4.64	3.82	4.36	4.12	5.04
Case 2	3.16	3.35	3.66	3.76	3.68	3.82	3.72	3.68	4.16	4.12
Case 3	3.32	11.68	3.60	9.40	3.88	9.48	3.86	9.58	4.14	9.92
Case 4	5.20	14.56	5.36	11.74	5.48	11.86	5.40	11.86	5.84	12.24
Message parsing										
Case 1	16.14	18.32	15.70	18.08	15.66	18.18	15.74	18.16	15.50	18.12
Case 2	16.04	15.60	15.52	15.54	15.78	15.62	15.48	15.54	15.56	15.86
Case 3	15.42	41.24	15.54	36.32	15.62	36.78	15.54	36.60	15.54	35.96
Case 4	20.86	49.40	20.44	43.30	20.56	44.06	20.46	43.88	20.36	43.06

efficiency is an important feature especially in wireless environments, but faster handling of messages becomes important when either processing power is limited or a great deal of messages should be handled. Former is true in today's low-end mobile devices and the latter can be expected in the future when agent technology is employed on a large scale. Additionally, we analyzed what is needed from the message transport to support efficient and reliable agent communication in wireless environments. Our implementation of the bit-efficient FIPA-ACL is available as open source, and at the time of writing this paper it is being integrated to both Jade and FIPA-OS agent platforms.

We are currently investigating an efficient encoding for various content languages and models for efficient agent interaction protocols in wireless environments. For content languages, we are considering three options. Firstly, developing a new content language that is suitable for wireless environments. This option is unattractive, as it also requires that "wireless-unaware" agents should support it to communicate with a "wireless-aware" agent. Secondly, we are considering syntax-directed compression for FIPA-SL. Lastly, we are considering using binary-XML to encode various content languages. Some of the FIPA's content languages already have XML encoding syntax, but for example FIPA-SL does not. However, there have been discussions for XML encoding syntax for the FIPA-SL and therefore the last option might be the best way to solve this problem.

The interaction protocol in a wireless 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 more bandwidth can be used, 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 needed 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.

References:

[1] F. Bellifemine, A. Poggi, and G. Rimassa. JADE — A FIPA-Compliant Agent Framework. In *Proc. of the PAAM-99 Conference*, pages 97–108, 1999.

[2] Foundation for Intelligent Physical Agents. *FIPA ACL Message Representations*, 2000. Specification numbers FIPA00069, FIPA00070, FIPA00071.

[3] Foundation for Intelligent Physical Agents. *FIPA Agent Message Transport Envelope Representation Specifications*, 2000. Specification numbers FIPA00073, FIPA00085, FIPA00088.

[4] M. Liljeberg, K. Raatikainen, M. Evans, S. Furnell, N. Maumon, E. Veltkamp, B. Wind, and S. Trigila. Using CORBA to Support Terminal Mobility. In *Proc. of TINA'97 Conference*, pages 56–67. IEEE Computer Society Press, 1998.

[5] G. Montenegro, S. Dawkins, M. Kojo, V. Magret, and N. Vaidya. RFC 2757: Long Thin Networks, Jan. 2000. (Informational).

[6] Object Management Group. Wireless access and terminal mobility in CORBA. OMG Document dtc/2001-06-02, June 2001. OMG Final Adopted Specification.

[7] S. Poslad, P. Buckle, and R. Hadingham. FIPA-OS: The FIPA Agent Platform Available as Open Source. In J. Bradshaw and G. Arnold, editors, *Proc. of the PAAM 2000 Conference*, pages 355–368, Apr. 2000.