Performance Comparison of some Message Transport Protocol Implementations for Agent Community Communication

Phuong T. Nguyen, Volkmar Schau and Wilhelm R. Rossak

Friedrich Schiller University Jena, Germany
phuong-thanh.nguyen|volkmar.schau|wilhelm.rossak@uni-jena.de

Abstract: Like in any distributed community systems, communication in Multi-Agent System (MAS) plays an important role. Communication helps agents share knowledge and cooperate. Agent communication may present at various forms; it can be the exchange of comprehensible messages, a request to perform an action, or cooperation negotiation. Agent communication has common properties like in any other distributed community systems, but it is the use of Agent Communication Language (ACL) that differentiates agent communication from that in other distributed community systems. In this paper, we present an evaluation of some typical Message Transport Protocol (MTP) implementations for agent communication. In order to analyze performance and stability of these MTPs, the tests have been conducted in conjunction with three existing ACL message encoding schemes.

1 Introduction

Communication between agents is one of the most important features of MAS. In general, agent communication is communication between software entities and therefore, it has common properties with communication in distributed community systems. What makes agent communication different from other common communication techniques is the use of ACL Message in exchanging information. ACL is designed specifically to describe and facilitate the communication process between agents. It does not deal with the physical exchange over the network, but with the specification of the content of the exchange. A specific ACL can be considered as a collection of message types, each type of message has a pre-defined meaning, for example: a query, an interest for a particular content, or an assertion. So far, in Agent Communication specifications, FIPA [1] has defined 22 types of messages which are called performative.

The use of ACL in MAS accounts for the difference between communication in MAS and that in conventional distributed systems. An ACL message consists of different components, each of them can be represented in different ways. Since an MAS can use different transport protocols to transfer ACL messages, there is a link between ACL and Message Transport Protocol (MTP).

[1] The Foundation for Intelligent Physical Agents (http://www.fipa.org/)

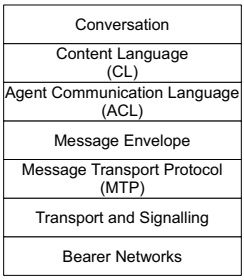

Figure 1: Layered Architecture for Agent Communication [Hel03]

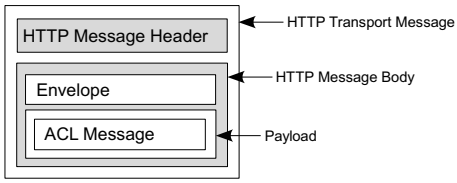

Figure 2: Encapsulation of ACL Message by HTTP Transport

In this paper, we persent a performance evaluation of some MTP implementations for agent communication. We consider to analyze the following MTPs: ORBacus based MTP, OpenORB based MTP, Sun ORB based MTP, and HTTP based MTP. The paper is organized as follows: Section 2 gives an introduction to the layered model in agent communication. Section 3 describes the use of Hypertext Transfer Protocol (HTTP) and CORBA [2] as MTP in agent communication. Section 4 draws an overview of ACL message encoding techniques. Section 5 presents the results of this work. Finally, Section 6 concludes the paper.

2 Layered Model for Agent Community Communication

Abstraction methods play an important role in system design as they can be used to represent the structure of the system in a new space so that reasoning in the new model is simpler than that in the original [Cal02]. For agent communication, an abstract model had been introduced in [Hel03] as illustrated in Figure 1. This layered model decouples low-level data transport issues from high-level semantics aspects. The two layers Transport and Signalling and Bearer Networks are considered as low-level layer and take care of routing and delivering agent messages through networks. The rest is high-level semantic interoperability layers; they parse and interpret messages. The aim of the layering process is to alleviate the optimization of agent communication in different levels, independently [Hel03].

- The Transport and Signalling Layer: In every distributed community systems including agent systems, the Transport and Signalling Layer accounts for an important contribution to communication performance. It is responsible for low-level of transferring data over a network.
- The Message Transport Protocol Layer: This layer defines the structure of messages using a transport protocol. It delivers the messages to the destination agent in the correct order and informs the sender when communication error occurs.

[2] Common Object Request Broker Architecture

- The Message Envelope Layer: With the introduction of this layer, messages can be sent between agent platforms regardless of their content. The ACC which transfers the messages does not have to care ACL messages' content.
- The Agent Communication Language Layer: This layer specifies the syntax and semantics of agent messages. FIPA-ACL and Knowledge Query and Manipulation Language (KQML) are the most widely used communication languages for inter-agent communication. The two languages share very similar syntax and semantics, and they are both based on the speech act theory [Spe04]. In FIPA-ACL specification, ACL can be considered as a collection of message types, each corresponding to a pre-defined meaning. An ACL can differentiate between various types of messages: a query, an interest for a particular content, or an assertion.
- The Content Language Layer: The ACL messages are used to represent the syntax and semantic of the communication and the content languages specify the actions that need to be done by the receiver. The Content Language layer is used to express the content of a communication between agents.
- The Conversation Layer: The process of exchanging messages between coordinated parties like agents typically falls into common patterns. At any point of the conversation, a certain type of messages is expected. The patterns of message exchange are called interaction protocol and they represent the highest abstraction component in the agent communication stack [Cal02],[Bon02].

3 Message Transport Protocols

3.1 HTTP based MTPs

The Message Transport Protocol between agents using the Hypertext Transfer Protocol (HTTP) has been specified by FIPA in [FIP02d]. In order to send an ACL Message to the receiver, the sender encapsulates the ACL Message in an HTTP POST request and then submits the request to the receiver. At the destination, after receiving the request, the receiver acknowledges by sending an HTTP response 200 back to the sender. An HTTP request used in context of agent communication is illustrated in Figure 2 and consists of the following components:

- ACL Message: The original ACL Message that needs to be sent using HTTP.
- Payload: The payload is the ACL Message itself but represented in another form. So far, FIPA has defined three encoding schemes for ACL Message: Bit-efficient encoding [FIP02a], String ACL Encoding [FIP02b], and XML ACL Encoding [FIP02c]. An overview of these techniques is given in section 4.
- Envelope: The envelope consists of the transport address of the destination and its incoming port.

- HTTP Message Body: This message component is made of the envelope and the payload.
- HTTP Message Header: This component contains necessary parameters for HTTP: Content-Type, Host, Cache-Control, MIME-Version.

3.2 CORBA based MTP

3.2.1 Overview of CORBA

The Common ORB Architecture (CORBA) was introduced in order to provide a standard computing infrastructure for distributed object-oriented applications. It has been shown that in object-oriented distributed applications, objects communicate faster if they are co-located in the same process than if they interact across process or machine boundaries. CORBA allows a distributed, heterogeneous collection of objects to interoperate. It enables interoperability between applications which are written in multiple programming languages and expected to run on different machines across network. CORBA is independent of platform, programming language and therefore it provides a flexible infrastructure for distributed applications. In CORBA architecture, CORBA clients make calls to objects instead of making calls to a server process. At the programming view, remote calls look similar to local calls [Mah00],[Vin00]. It should be noted that, in CORBA architecture, the names ”server” and ”client” are used interchangeably, based on the context of use. Client is the entity that sends request to other entity which is called server to invoke methods. Their role can be reserved as ”server” invokes methods on ”client.” A typical CORBA architecture consists of the following components:

- The Object Request Broker (ORB) [IBM07],[Vin00]. A CORBA object is viewed from the client side as an entity that provides defined operations which are always available to the client side. The object works as the interface for remote invocation but its actual implementation is an entity called the servant. The servant is the executing CPU and memory resource that performs object's operation, it contains methods for handling remote method invocation. The servant is visible only to the server. Every object on the server which wants to have its methods being invoked by other objects, has to register itself with the ORB. After registration, each object is assigned an unique identifier called the Interoperable Object Reference (IOR). Normally, an IOR is made of the remote host's IP address, the port number of the CORBA server, a string defining the class of the remote object on which the methods will be invoked, and an object key which works as identifier for the servant.
- The Portable Object Adapter (POA) in ORB is responsible for managing server-side resources for scalability. POA deactivates objects' servants when no task is dedicated to them, and activates them again when they are in need. ORB uses object key to identify the right POA. An object ID which is a part of the object key is used by the POA to map the target object to its servant. Clients invoke methods which

have been exposed by object and POA maps the object to the appropriate servant [OMG10].

- The General Inter-ORB Protocol (GIOP) is the network protocol for communication between ORBs. The GIOP version which works over the Internet is Internet Inter-ORB Protocol (IIOP). IIOP enables interoperability between objects using CORBA products from multiple vendors.

3.2.2 Agent Community Communication using CORBA

In MAS, agents work on multiple platforms and operating systems, they send and receive messages through heterogeneous networks. In CORBA's view, sender and receiver agents are distributed objects. Sending messages between agents means method invocation on these remote objects takes place. CORBA architecture is considered to be an appropriate communication infrastructure for Multi-Agent Systems. We take an example of agent community communication using CORBA, agent α on client C sends a message to agent β on remote server S. Agent β with method *deliver()* is an object and has been registered with server-side ORB. It means it's servant is ready to be invoked. Agent α on the client-side obtains the IOR of the servant of the agent β from the appropriate ORB, it then invokes method *deliver()* by sending the request using the IOR. The request with all parameters of function *deliver()* is transferred through IIOP to the server-side ORB. The server-side ORB then dispatches the received request to the POA that holds the target object. An application may have multiple POAs, so ORB uses one part of the IOR, called the object key to identify the appropriate POA. The POA in turn uses the object ID which is extracted from the object key to map the target object to its appropriate servant and feeds the request to the servant. Finally, the method is executed on the servant and the message is delivered to the target agent. The parameters of the function *deliver()* are only the message envelope and the payload. Unlike HTTP based MTP, there is no need to add additional transport information for CORBA based transport protocol. Therefore, compared to HTTP based MTP, CORBA based MTP consumes less overhead for encoding messages.

4 ACL Message Encoding

The Agent Communication Language Layer in the layered model of agent community communication specifies the syntax and semantics of agent messages. For the purpose of optimization based on network environments, ACL messages can be represented in different formats. FIPA has defined three standard encoding schemes for ACL messages, they are: Bit-efficient encoding [FIP02a], String ACL encoding [FIP02b], and XML ACL encoding [FIP02c]. The following section gives a brief introduction to these encoding techniques.

Bit-efficient encoding technique encodes message by replacing fields of ACL messages with a combination of pre-defined hexadecimal byte. This technique can reduce the size of ACL messages considerably. Consequently, bit-efficient encoding scheme is suitable

for transferring messages in wireless environment. *String ACL encoding* uses string to represent message fields. This encoding technique is not optimal as performing operations on strings consumes more time than other types of data, for example byte. *XML ACL encoding* is based on XML [3] which is used to represent and exchange data over the Internet. The ACL message representation in XML has been specified in [FIP02c]. Unlike the previous techniques, ACL representation in XML produces more overhead to messages by adding XML tags; however, it is preferred because of the following reasons [Che08], [Lab99]:

- XML is easy to manipulate; it enables users to format their own document structure.
- XML can be used to represent different types of information in human-readable format.
- XML is an open standard and is used by various software systems. Encoding ACL messages using XML allows MASs to be more open to other systems. As a result, the interoperability between heterogeneous systems can be improved.
- As XML has been used widely in web services, the deployment of XML in ACL messages facilitates collaboration between MASs and web services.

5 Performance Evaluation of MTPs

We perform evaluation for the following MTPs: SunORB based MTP, ORBacus based MTP, OpenORB based MTP, and HTTP based MTP. In order to analyze performance of these MTPs, we execute tests in conjunction with the three ACL message encoding schemes, which have been described in Section 5. The aim of this combination is to check the stability of each MTP towards different message encoding schemes. Referring to the layered model in section 2, the evaluation is related to two layers: The Message Transport Protocol Layer and the Agent Communication Language Layer. JADE (Java Agent DEvelopment Framework) is a framework for developing multi-agent systems developed at TILAB [4][Rim03]. JADE is compliant with FIPA97 specification [TIL10] and supports agent communication of different levels. Communication messages in JADE are represented according to FIPA ACL specifications [Rim01]. As JADE has been used widely in research community, we opted for it as the testing platform for our performance experiments.

5.1 OpenORB based MTP

Besides integrated MTPs, other MTPs have been also implemented for JADE using its interface [Lyo03],[TIL00],[Kle09]. The implementation for a new MTP in JADE is quite

[3] eXtension Markup Language
[4] Telecom Italia Lab

straightforward as JADE has been well structured so that MTPs can be integrated as "plug and play" protocols. Developers need to implement some Java abstract classes and interfaces. To the best of our knowledge, so far two CORBA based implementations have been already implemented as the Message Transport Protocol for JADE, they are JDK's built-in ORB (Sun ORB) and ORBacus [TIL00],[OOC00],[Rim00]. We implemented a new CORBA based MTP using OpenORB as an add-on for JADE. OpenORB is a free Java open source CORBA implementation with a BSD-like license, it has been developed by the Distributed Object Group with the original name JavaORB. OpenORB combines all CORBA features with implementation specific extensions, and can be customized to meet applications' requirements. It complies with JDK 1.2 and 1.3, and CORBA 2.4.2 [Ope05].

5.2 Test Setup

Our performance evaluation tests are setup based on the benchmark for performance and scalability of JADE agent platform which is described in [ea02]. Originally, the benchmark aims to measure performance and scalability of JADE. However, it can also be used to measure efficiency of an MTP. In this test, two agent platforms exchange ACL messages and the performance is measured as the average round trip time ($avgRTT$). The average round trip time is calculated as the time needed for sending an ACL message from a sender agent to a receiver agent and then back to the sender. The sending is repeated for M times. The scalability is measured as the performance changes by adding N couples of agents to both platforms. The measurement unit is millisecond (ms). In our experiments, we set the iteration number M to 1000 times for every test configuration. By varying the number of couples of agents N=$\{$10, 50, 100, 200, 300, 500, 1000$\}$ and sketching towards the average round trip time, we obtained performance diagrams for MTPs. Java source codes for the benchmark, ORBacus based MTP, and the three ACL message encoding techniques are available for downloading at JADE Website [TIL10]. The test configuration is listed in Table 1.

	Sender Node	**Receiver Node**
Processor	AMD Athlon 64 2.21GHz	Intel 1.7GHz
RAM	2GB	1GB
Operating System	Windows XP SP3	Windows XP SP3
JADE	version 4.0	version 4.0
Java & Sun ORB	Sun JDK 1.6	Sun JDK 1.6
ORBacus	version 4.0.3	version 4.0.3
OpenORB	version 1.4.0	version 1.4.0
HTTP	version 1.1	version 1.1
Network	1000Mbps	1000Mbps

Table 1: Hardware and Software configuration for the test scenario

5.3 Test Scenario

- Scenario 1: The message is an ACL INFORM performative with the content consisting of an ASCII string with seven bytes.
- Scenario 2: The message is the same as in Scenario 1, except that its content size has been increased to 2150 bytes.

5.4 Experiment Results

Six independent performance tests have been conducted based on the scenario described above. Performance diagrams for MTPs with different ACL message encoding schemes for Scenario 1 are shown in Figure 4, Figure 3, and Figure 5. Test results for Scenario 2 are displayed in Figure 7, Figure 6, and Figure 8. In general, HTTP based MTP has the best performance compared to all protocols, in every encoding schemes. Among CORBA based MTPs, OpenORB based MTP outperforms the others. From the Figure 4, Figure 3, and Figure 5, it can be seen that, when the payload is small, HTTP based MTP has outstanding performance; OpenORB based MTP has clearly better performance than its CORBA colleagues, and Sun ORB based MTP has the worst performance. When the payload size increases as in Figure 7, Figure 6, and Figure 8, the performance difference becomes smaller. Regarding the ACL message encoding schemes, in all test configurations, bit-efficient has higher efficiency than the others.

In order to investigate the robust of each transport protocol towards the message encoding, we performed some statistical analyses on the test resutls. We analyzed how sensitive a transport protocol to message size is. For every message encoding scheme, we considered each MTP by dividing the $avgRTT$ value for the small payload (Scenario 1) by the $avgRTT$ value for the big payload (Scenario 2).

$$\varepsilon = \frac{avgRTT^{N}_{Scenario1}}{avgRTT^{N}_{Scenario2}} \tag{1}$$

In which N is the number of couples of agents. By averaging for all number of couples of agents N=$\{$10, 50, 100, 200, 300, 500, 1000$\}$, we got the meant value:

$$\delta(MTP, Encoding) = average(\varepsilon) \tag{2}$$

It means, the bigger δ is, the more sensitive the protocol to message size is. Three cases are explained as follows:

- $\delta = 1$: The increase in message size has no effect on transfer time.
- $\delta > 1$: The protocol takes more time to transfer small messages than to transfer big messages.

- $\delta < 1$: The protocol takes less time to transfer small messages than to transfer big messages. In this case, when $\delta \approx 1$, the protocol is stable to message size.

The results are shown in Table 2. HTTP based MTP is most sensitive to message size when used in conjunction with String ACL encoding. When ORBacus based MTP is used with Bit-efficient, transmission performance is improved since it takes less time to transfer big messages. Likewise, OpenORB based MTP gains good performance ($\delta = 0.96$) if it combines with Bit-efficient encoding. Sun ORB based MTP is considerably sensitive to message size in all encoding schemes ($\delta = 0.8; 0.72; 0.78$).

	Bit-Efficient	**String ACL**	**XML ACL**
Sun ORB MTP	0.8	0.72	0.78
ORBacus MTP	1.05	0.87	0.89
OpenORB MTP	0.96	0.76	0.83
HTTP MTP	0.91	0.65	0.8

Table 2: MTP Sensitiveness to Messsage Size

6 Conclusion

In this paper, we introduced a performance evaluation of some existing Message Transport Protocol implementations for agent community communication. In every test configuration, HTTP based MTP has the best performance compared to all other MTPs despite of its addtional overhead in transport message. Among CORBA based MTPs, OpenORB based MTP runs fastest as it needs less time to transfer the same amount of ACL messages. Bit-efficient encoding scheme can compress ACL messages and therefore it is suitable for transferring message in low bandwidth network environment.

Acknowledgment

Phuong T. Nguyen gratefully acknowledges the financial supports from the Vietnamese Overseas Scholarship Program and the German Academic Exchange Service (DAAD).

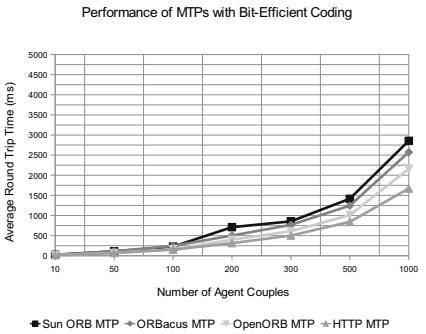

Figure 3: Performance of MTPs in case of Bit-efficient encoding (Scenario 1)

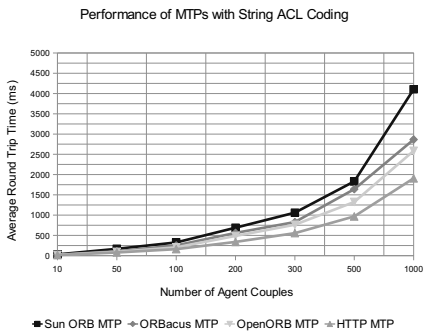

Figure 4: Performance of MTPs in case of String ACL encoding (Scenario 1)

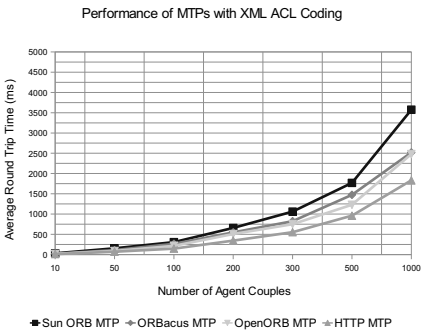

Figure 5: Performance of MTPs in case of XML encoding (Scenario 1)

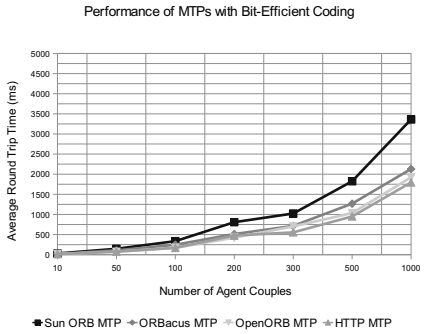

Figure 6: Performance of MTPs in case of Bit-efficient encoding (Scenario 2)

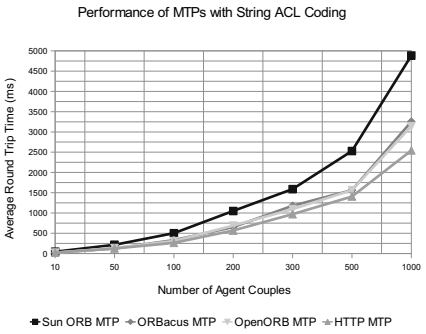

Figure 7: Performance of MTPs in case of String ACL Encoding (Scenario 2)

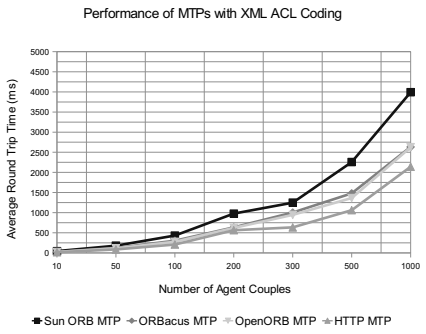

Figure 8: Performance of MTPs in case of XML encoding (Scenario 2)

Bibliography

[Bon02] Stula, M.; Stipanicev, D.; Bonkovic, M. : Feasible Agent Communication Architecture. In *International Conference on Software, Telecommunications and Computer Networks*, 2002.

[Cal02] Calisti, M. : Abstracting Communication in Distributed Agent-based Systems. In *Proceedings of the 16th European Conference on Object-Oriented Programming*, 2002.

[Che08] Chen, B.; Linz, D.; Cheng, H. : XML-based agent communication, migration and computation in mobile agent systems. *Journal of Systems Software*, 81:1364–1376, August 2008.

[ea02] Elisabetta Cortese et al. Scalability and Performance of JADE Message Transport System, 2002.

[FIP02a] FIPA. : FIPA ACL Message Representation in Bit-Efficient Specification. 2002.

[FIP02b] FIPA. : FIPA ACL Message Representation in String Specification. 2002.

[FIP02c] FIPA. : FIPA ACL Message Representation in XML Specification. 2002.

[FIP02d] FIPA. : FIPA Agent Message Transport Protocol for HTTP Specification. 2002.

[Hel03] Helin, H. *: Supporting Nomadic Agent-based Applications in the FIPA Agent Architecture*. PhD thesis, University of Helsinki, Finland, 2003.

[IBM07] IBM. : Object Request Broker. 2007.

[Kle09] Micsik, A.; Pallinger, P.; Klein, A. : SOAP based Message Transport for the Jade Multiagent Platform. 2009.

[Lab99] Benjamin, G.; Labrou, Y. : An Approach to using XML and a Rule-based Content Language with an Agent Communication Language. In *Communication Language, IJCAI-99 Workshop on Agent Communication Languages*. Springer-Verlag, 1999.

[Lyo03] Curry, E.; Chambers, D.; Lyons, G. : A JMS Message Transport Protocol for the JADE Platform. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, page 596. IEEE Computer Society, 2003.

[Mah00] Mahmoud, H. *: Distributed Programming with Java*. Manning Publications Co., 2000.

[OMG10] OMG. : ORB Basics. *Object Management Group*, 2010.

[OOC00] OOC. : ORBacus Trader. ORBacus for C++ and Java. Technical report, OOC, Object Oriented Concepts, Inc., 2000.

[Ope05] OpenORB. : The Community OpenORB. 2005.

[Rim00] Rimassa, G. : How to use Orbacus ORB in JADE. 2000.

[Rim01] Bellifemine, F.; Poggi, A.; Rimassa, G. : Developing Multi-agent Systems with JADE. In *ATAL '00: Proceedings of the 7th International Workshop on Intelligent Agents VII. Agent Theories Architectures and Languages*, pages 89–103. Springer-Verlag, 2001.

[Rim03] Bellifemine, F.; Poggi, A.; Caire, G.; Rimassa, G. : JADE: A White Paper. 3:6–19, 2003.

[Spe04] Spenader, J. : Speech Act Theory. 2004.

[TIL00] TILAB. : ORBacus MTP Implementation source code. 2000.

[TIL10] TILAB. : Java Agent DEvelopment Framework. 2010.

[Vin00] Vinoski, S. : Introduction to CORBA (tutorial session). In *Proceedings of the 22nd International Conference on Software Engineering*, page 822. ACM, 2000.