Performance Comparison of some Message Transport Protocol Implementations for Agent Communication

Phuong T. Nguyen
Dept. of Computer Science
University of Jena, Germany
phuong-thanh.nguyen@uni-jena.de

Volkmar Schau
Dept. of Computer Science
University of Jena, Germany
volkmar.schau@uni-jena.de

Wilhelm R. Rossak
Dept. of Computer Science
University of Jena, Germany
wilhelm.rossak@uni-jena.de

***Abstract*—Like in any distributed systems, communication in Multi-Agent System (MAS) plays an important role as it helps agents sharing knowledge and facilitating cooperation. Communication in MAS may present at various forms; it can be exchanging of comprehensible messages, or requesting to perform an action, or negotiating cooperation. Agent communication has common properties like in any other distributed systems. But it is the use of ACL messages that differentiates agent communication from that in other distributed systems. Based on the layered model for agent communication, there have been many studies done in order to optimize communication performance between agents in MAS.**

In this paper, we present an evaluation of some typical Message Transport Protocol (MTP) implementations for agent communication. The performance tests are conducted on the JADE Agent Platform.

***Keywords*-Agent Communication; Message Transport Protocol; CORBA; JADE; Performance;**

I. Introduction

In the industry of intelligent agents, the Foundation for Intelligent Physical Agents (FIPA) appears to be the most prominent international organization as it is specialized in developing specifications supporting interoperability among agents and agent-based applications. In specifications for Agent Communication, FIPA has defined Agent Communication Language (ACL) messages, message exchange protocols, speech act theory-based communicative acts and content language representations.

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 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 has defined 22 types of messages which are called performative.

In this paper, we persent a performance evaluation of some Message Transport Protocol (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 II gives an introduction to the layered model in agent communication. Section III briefly describes the use of Hypertext Transfer Protocol as Transport Protocol in agent communication. Section IV introduces CORBA and its application in agent communication. Section V presents the results of this work. Section VI concludes the paper.

II. Layered Model for Agent Communication

In system design, abstractions methods play an important role 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 in the original [2]. For agent communication, an abstract model had been introduced in [11] 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 [11].

A. The Transport and Signalling Layer

The Transport and Signalling Layer accounts for an important contribution to communication performance in every distributed systems, including agent systems as it is responsible for low-level of transferring data over a network. This layer is expected to provide an efficient and reliable data transport service. Two typical transport protocols are TCP and UDP.

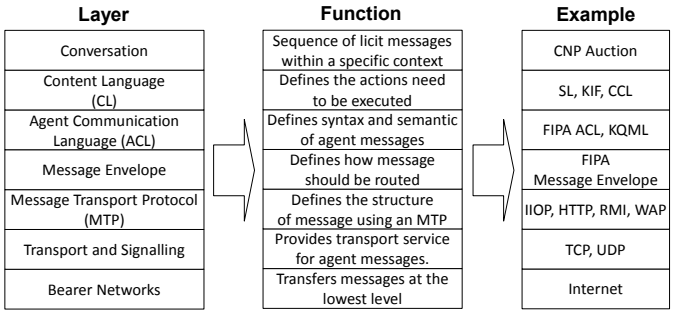

Figure 1. Layered Architecture for Agent Communication [11]

B. The Message Transport Protocol Layer

The 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. So far, FIPA has defined three MTPs: MTP based on IIOP, MTP based on WAP, and MTP based on HTTP.

C. 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 take care ACL messages' content. Three message envelope syntaxes have been defined in FIPA specifications. The first is "built-in" in the message transport protocol for IIOP MTP. The second is the XML-based message envelope syntax. The last is bit-efficient encoding.

D. 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 a very similar syntax and semantics, and they are both based on the speech act theory [21].

FIPA-ACL is designed specifically to describe and facilitate the communication process between agents. An ACL can be considered as a collection of message types, each corresponds to a pre-defined meaning. ACL does not deal with the physical exchange over the network, but with the specification of the content of the exchange. Hence, an ACL can differentiate between various types of messages: a query, an interest for a particular content, or an assertion. So far, in Agent Communication specifications, FIPA has defined 22 types of messages which are called performative.

E. The Content Language Layer

The ACL messages are used to represent the syntax and semantic of the communication, but the actions that need to be done by the receiver are specified by the content language. The Content Language layer is used to express the content of a communication between agents. Some examples of content language are: FIPA-SL, FIPA-CCL, FIPA-KIF.

F. 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 [2],[15]. FIPA has defined a set of interaction protocols which consists of 22 different types of messages.

III. HTTP based Message Transport Protocol

The transport of message between agents using the Hypertext Transfer Protocol (HTTP) has been specified by FIPA in [10]. 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 in case 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 [7], String ACL Encoding [8], and XML ACL Encoding [9].
- 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.

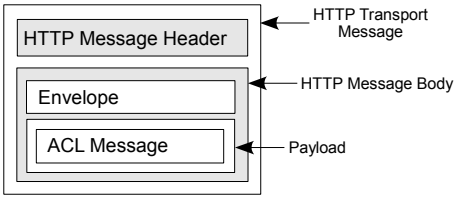

Figure 2. Encapsulation of ACL Message by Transport HTTP

IV. CORBA based Message Transport Protocol

A. Overview of CORBA

In distributed object applications, all entities are modeled as objects and application is a set of cooperating objects. An

important issue in these applications is how to optimize the communication between objects. The nature of distributed applications, such as: data used by the applications, the computation, and the users are all distributed over networks, makes them more difficult to be implemented and deployed effectively. 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. The co-locating of objects helps software developers to handle objects more efficient. In addition, error handling, security, thread management can also be improved [3].

The Common ORB Architecture (CORBA) was introduced in order to provide a standard computing infrastructure for distributed object-oriented applications. CORBA allows a distributed, heterogeneous collection of objects to interoperate. It enables interoperability between applications which are written in multiple programming languages and due to run on different machines across network. It is independent of platform, programming language and therefore it provides a flexible infrastructure for distributed applications.

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." 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 [14],[23].

The core of CORBA architecture is the distributed service Object Request Broker (ORB) which is responsible for [12],[23]:

- managing object location,
- implementing request to remote objects,
- locating objects in the network,
- issuing requests to the object,
- waiting for the output values, and
- sending back the results to the client.

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) is a constituent part of the ORB, it is responsible for managing server-side resources for scalability. POA takes care of deactivating objects' servants when no task is dedicated to them, and activating 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. Client invokes methods which have been exposed by object and POA maps the object to the appropriate servant [17].

The communication between ORBs takes place through the use of a network protocol called General Inter-ORB Protocol (GIOP). 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.

B. Agent Communication using CORBA

In MAS, agents work on multiple platforms and operating systems, they send and receive messages through heterogeneous networks. In the view of CORBA, sender and receiver agents are distributed object themselves, and sending messages between agents is just method invocation on remote objects taking place. Consequently, CORBA architecture is considered to be an appropriate communication infrastructure for Multi-Agent Systems.

We take an example of agent 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 communication architecture is illustrated in Figure 3 [14],[23].

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.

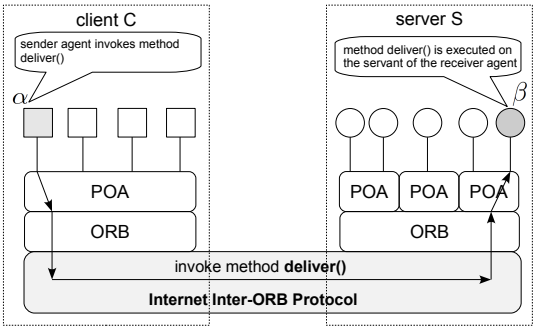

Figure 3. Agent Communication using CORBA

V. Performance Evaluation of MTPs

In this section, we evaluate performance 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 perform tests in conjunction with three ACL Message Encoding schemes that have been defined by FIPA: Bit-Efficient Coding [7], String ACL Encoding [8], and XML ACL Encoding [9].

A. JADE Agent Communication

As JADE (Java Agent DEvelopment Framework) has been used widely in research environment, we use it as the testing platform for our performance experiments.

JADE is a framework for developing multi-agent systems developed at Telecom Italia Lab (TILAB) [6]. JADE is compliant with FIPA97 specification [22] and supports agent communication in different levels. Communication messages in JADE are represented according to FIPA ACL specifications [1].

Depending on the location of communicating agents, JADE supports three types of communication [1]:

- Communication between agents in the same agent container: The ACLMessage is simply cloned by using Java events.
- Communication between agents in different containers of the same platform: Message is serialized at sender side, a Java RMI is called to transfer the message, and finally the message is unserialized at the receiver side.
- Communication between agents in different platforms: Currently, JADE supports HTTP and IIOP as inter-platform communication.

B. OpenORB based MTP

Besides integrated MTPs, other MTPs have been also implemented for JADE using its interface [4],[13],[16]. The implementation for a new MTP in JADE is quite straightforward as JADE has been well organized so that MTPs can be integrated as ”plug and play” protocol. 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 [13],[18],[20]. The Sun ORB based MTP implementation has already been integrated in JADE source code. The add-on ORBacus based Message Transport Protocol implementation can also be found at JADE website [13],[20].

We implemented new CORBA based MTP using OpenORB as add-ons for JADE. OpenORB is a free Java open source CORBA implementation with a BSD-like license, it has been developed by 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 [19].

C. Test Scenario

We set up our performance evaluation tests based on the benchmark for performance and scalability of JADE agent platform which is described in [5] and depicted in Figure 4.

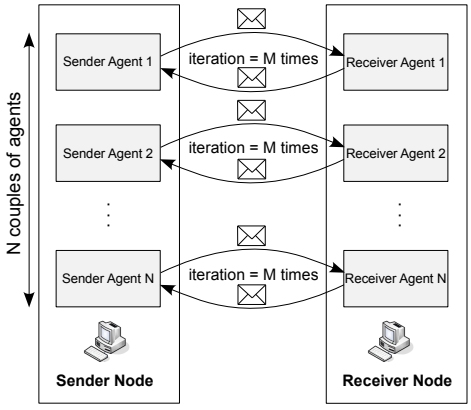

Figure 4. Test Scenario [5]

Originally, the test aims to measure performance and scalability of JADE. However, it can also be used to measure performance efficiency of a new MTP. In this test, the average round trip time (avgRTT) is calculated as the time needed for sending a message from a sender agent to a receiver agent and then back to the sender. The message is an ACL INFORM performative and has the content consisting of the string ”CONTENT” with seven letters. The measurement unit is millisecond (ms).

The average round trip time (avgRTT) is calculated as follows [5]:

$$avgRTT\_per\_couple = \frac{1}{M}\sum_{i=1}^{M} time(i) \qquad (1)$$

$$avgRTT = \frac{1}{N}\sum_{j=1}^{N} avgRTT\_per\_couple(j) \qquad (2)$$

In which $time(i)$ is the measured time at i^{th} iteration when a couple of agents exchanges one message.

In our experiments, we fixed 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 with different message encoding schemes.

Java source code for the benchmark and three ACL Message Encoding techniques are available for downloading at JADE Website [22]. The test configuration is listed in Table I.

Table I
HARDWARE AND SOFTWARE CONFIGURATION FOR THE TEST SCENARIO

	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

D. Results

Three performance tests have been conducted independently based on the test scenario described above. The performance diagrams for MTPs with different ACL Message Encoding schemes are shown in Figure 5, Figure 6, and Figure 7. It can be seen that, HTTP based MTP has the best performance among other protocols for all encoding schemes. Among CORBA based MTPs, OpenORB based MTP outperforms the others.

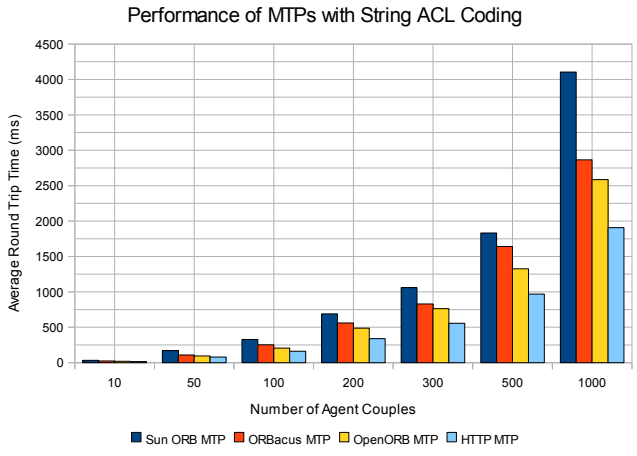

Figure 5. Performance Comparison for MTPs in case of String ACL Coding

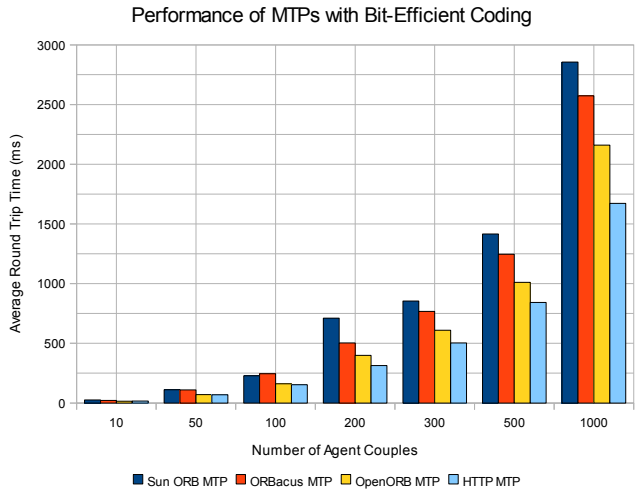

Figure 6. Performance Comparison for MTPs in case of Bit-efficient Encoding

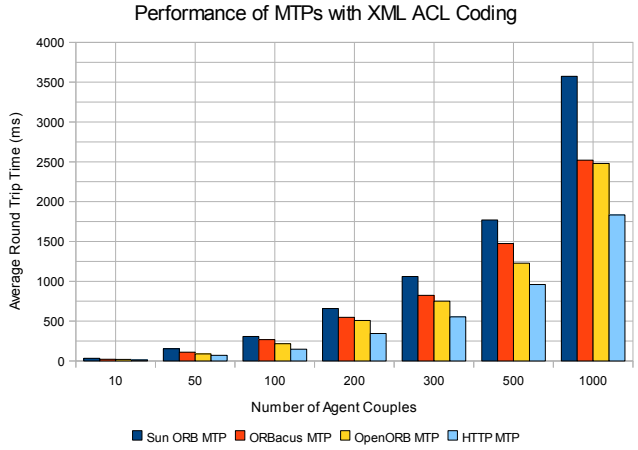

Figure 7. Performance Comparison for MTPs in case of XML Encoding

VI. CONCLUSION

In this paper, we introduced a performance evaluation of some existing Message Transport Protocol implementations for agent 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 works fastest as it needs less time to transfer the same amount of ACL messages.

ACKNOWLEDGMENT

Phuong T. Nguyen gratefully acknowledges the financial supports from the Vietnamese Overseas Scholarship Program and DAAD.

REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. 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, London, UK, 2001. Springer-Verlag.

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

[3] A. Chaffee and B. Martin. Introduction to CORBA. *SUN Tutorials and Online Training*, 1999.

[4] E. Curry, D. Chambers, and G. Lyons. A JMS Message Transport Protocol for the JADE platform. In *IAT '03: Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, page 596, Washington, DC, USA, 2003. IEEE Computer Society.

[5] E. C. et al. Scalability and Performance of JADE Message Transport System, 2002.

[6] A. P. F.Bellifemine, G. Caire and G. Rimassa. JADE: A White Paper. 3:6–19, 2003. http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf.

[7] FIPA. FIPA ACL Message Representation in Bit-Efficient Specification, 2002.

[8] FIPA. FIPA ACL Message Representation in String Specification, 2002.

[9] FIPA. FIPA ACL Message Representation in XML Specification, 2002.

[10] FIPA. FIPA Agent Message Transport Protocol for HTTP Specification, 2002.

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

[12] IBM. Object Request Broker. http://www-01.ibm.com/support/docview.wss?uid=swg27009777aid=1.

[13] T. I. Lab. ORBacus MTP Implementation source code. 2000. http://jade.tilab.com/dl.php?file=ORBacusMTPAddOn-3.3.zip.

[14] Q. H. Mahmoud. *Distributed Programming with Java*. Manning Publications Co., Greenwich, CT, USA, 2000. http://java.sun.com/developer/Books/corba.

[15] M. B. Maja Stula, Darko Stipanicev. Feasible agent communication architecture. In *International Conference on Software, Telecommunications and Computer Networks*, 2002.

[16] A. Micsik, P. Pallinger, and A. Klein. SOAP based Message Transport for the Jade Multiagent Platform. 2009.

[17] OMG. ORB Basics. *Object Management Group*, 2010. http://www.omg.org/gettingstarted/orb$_b asics.htm$.

[18] OOC. ORBacus trader. ORBacus for C++ and Java. Technical report, OOC, Object Oriented Concepts, Inc., 2000. http://www.ooc.com/ob.

[19] T. C. OpenORB. Community OpenORB. 2005. http://openorb.sourceforge.net/.

[20] G. Rimassa. How to use Orbacus ORB in JADE. 2000.

[21] J. Spenader. Speech Act Theory. 2004.

[22] TILAB. Java Agent DEvelopment Framework. 2010. http://jade.tilab.com.

[23] S. Vinoski. Introduction to CORBA (tutorial session). In *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, page 822, New York, NY, USA, 2000. ACM.