Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Information Sciences 150 (2003) 153–164

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

AgentGateway: A communication tool for multi-agent systems

Jason Jen-Yen Chen [a,*], Shih-Wei Su [b]

[a] *Department of Computer Science and Information Engineering, National Central University, Jung-Li, Taiwan*

[b] *Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsin-Chu, Taiwan*

Received 30 December 2001; received in revised form 17 March 2002; accepted 4 July 2002

Abstract

With the rapid growth of multi-agent systems (MAS), there is a pressing need to communicate between different MAS. Various MAS communication standards have been proposed. However, MAS is usually designed to meet special need, thus making it difficult to follow a standard. This paper presents a tool called AgentGateway trying to solve this problem using a simple, effective approach. The tool translates messages from one MAS to intermediate XML-based messages, which then are translated to messages for another MAS. AgentGateway is scalable, meaning that new MAS can easily join it. Furthermore, it provides transparent and reliable communication. A prototype is developed that shows communication between two MAS, namely, Java Agent Template Lite (JATLite) and OAA, can be done using this approach.

Keywords: Software agent; Multi-agent system; Agent communication language

1. Introduction

Agent is a new and powerful software abstraction with: (1) autonomy, that is, agent controls its behavior and internal states, (2) social ability, meaning

* Corresponding author. Tel.: +886-3-4227151x4515; fax: +886-3-4222681.
E-mail addresses: jychen@csie.ncu.edu.tw (J.J.-Y. Chen), swsuh@csie.nctu.edu.tw (S.-W. Su).

0020-0255/02/$ - see front matter
PII: S0020-0255(02)00374-2

that through agent communication language (ACL), agent is able to collaborate with other agents, and (3) reactivity, meaning that agent senses environment, including other agents' states, and reacts accordingly. In one sense, autonomy results from reactivity. The rationale is that agent behaves autonomously by reacting to its environment, rather than behaves passively by receiving messages as in object abstraction. As for ACL, commonly seen ones include Knowledge Query and Manipulation Language (KQML) [2], Foundation for Intelligent Physical Agents (FIPA) [3], and so on.

Let us explore the power of agent from software engineering perspective. Large and complex distributed systems are always bothered by interoperation problem, because heterogeneous components in those systems involve different developers, different languages, and different interfaces. Let us envision that each component can be developed as an agent, and all the agents can communicate through some ACL, the system can then be decomposed into relatively autonomous agents [1].

A software system composed of multiple agents is called a multi-agent system (MAS). From distributed problem solving viewpoint, MAS is regarded as:

> A MAS is a loosely coupled network of problem-solver entities that work together to find answers to problems that are beyond the individual capability or knowledge of each entity [4].

A more general meaning of MAS is that it is composed of autonomous components (agents) such that [5]:

- each agent has incomplete capability to solve a problem;
- there is no global system control;
- data is decentralized; and,
- computation is asynchronous.

As just mentioned, each agent in MAS has its own capability and knowledge. Collectively, agents collaborate to solve a problem. When a new agent joins the MAS, the agent will declare its needs and capabilities. Hopefully, it will find the agent providing the service it needs through a system program called middle agent [6]. Middle agent can assist agent to advertise, find, use, manage, and update agent services and information. There are many kinds of middle agents such as Facilitators [7], Mediators [8], Brokers [9], Matchmakers [10], Yellow pages [9], and Blackboards [11]. They are so roughly defined that sometimes it is difficult to distinguish one from another.

Recently, MAS by different developers has increased rapidly. Thus, a mechanism is definitely needed to make agents of different MAS interoperable. Several independent industrial or research groups have begun to pursue the standardization of multi-agent technology. However, because that MAS is often designed to meet special need and requirement, a MAS usually cannot

conform to any standard whatsoever. To solve that, this paper presents a simple and effective tool called AgentGateway. The tool enables agents to access services and information provided by agents of different MAS. Note that communication within one MAS is not the issue to be addressed here.

Design considerations of AgentGateway are:

1. AgentGateway should be scalable so that a new MAS can easily join to share information and capabilities. To facilitate that, there should be an intermediate agent communication language that different ACL can be translated to.
2. Agent communication between different MAS should be transparent. In other words, an agent can communicate with agents in different MAS without knowing the existence of other MAS. By doing so, MAS need not be modified for the purpose of communication.
3. AgentGateway should be reliable. AgentGateway is responsible for agent communication between different MAS, and all replies and queries will be transmitted through it. Thus, AgentGateway should be robust enough to ensure that messages will not get lost in transmission.

Besides, AgentGateway should register agent capabilities to other MAS so that agents can access capabilities of other MAS. Furthermore, because of open architecture of MAS, agent can join or leave MAS at any time. Therefore, the registration should be done in real-time manner.

This paper is organized as follows. Section 2 introduces MAS communication standards and related work. Architecture of this tool, including the XML-based ACL, and some considerations regarding implementation are depicted in Section 3. Then, an example in Section 4 illustrates the communication between two different MAS in details. Finally, benefits of this tool are concluded in Section 5.

2. Related work

Recently, several research groups started to standardize MAS. They include Object Management Group (OMG), the FIPA, and the Knowledgeable Agent-oriented System (KAoS) group.

OMG proposed a reference model [12], which describes an agent environment as composed of agents and agencies (the environment where agents live). In the model, agents are described based on their capabilities, types of interaction supported, and mobility. Agents are capable of communicating with other agents and agencies through pre-defined policies of interaction. An agency should support concurrent execution, security, and mobility of its agents.

FIPA proposes another reference model [13] which specifies the standard environment in which an agent can live and work, including an agent platform

that specifies agents deployment and interaction, and an agent management system that describes agents' creation, deletion, suspension, and resumption.

KAoS [10] describes the implementation of agent, and the planning of agent-to-agent communication protocol using agent conversation policies. In addition, KAoS provides a mechanism for KAoS agents and non-KAoS agents to communicate with each other, and an agent structure to manage beliefs, desires, intensions, and capabilities of agents.

However, as mentioned earlier, a MAS is designed to meet special need and requirement. Thus, design of MAS usually cannot conform to any standard. Consequently, such MAS will not be able to communicate with other MAS through a standard. To solve that, we try taking another approach of non-standardized agent communication. RETSINA–OAA InterOperator (ROI) [14] is a good example using this approach. ROI is an interoperate agent between two MAS, namely, RETSINA [15] and Open Agent Architecture (OAA) [16].

This tool called AgentGateway uses the approach similar to that of ROI. However, the two differ in three regards as below:

(1) *Scalability:* ROI focuses only on two specific MAS, namely, RETSINA and OAA, which lacks flexibility to expand. ROI should be redesigned if it is needed to communicate with MAS other than the two. On the other hand, AgentGateway is scalable, meaning that different communication interfaces can be designed to handle different MAS. In addition, AgentGateway uses XML-based intermediate ACL. Since XML is simple and well formed, it is easy for agents to understand the meaning and relationship of message constructed by XML. This facilitates designing different interfaces for various MAS.

(2) *Transparency:* The first two of three layers of ROI are: (1) MAS-specific Agent Session Layer that maintains agent communication session and protocol, (2) MAS-specific API Layer that provides different libraries of MAS for the development of particular system, such as Communicator of RETSINA and Agent Library of OAA. Thus, Agents in RETSINA and OAA can communicate without being aware of the existence of each other. ROI is treated as an agent in both RETSINA and OAA. Thus, architecture of either one need not be modified. Similarly, AgentGateway follows the communication protocol of MAS, and uses library API provided by different MAS to maintain system boundary of different agent systems.

AgentGateway thus has the advantage of transparency, because it hides the detailed process of messages translation and transmission to make agents believe as if they were communicating only with AgentGateway.

(3) *Reliability:* AgentGateway keeps agent communication messages and MAS service records in the database. When a message gets lost or AgentGateway crashes, the communication message can be resent based on those data. The query table in ROI does similar things. There are three tables in the top layer of ROI, namely: (1) OAA–RETSINA Query Table, (2) RETSINA–OAA Query Table (ROQT), and (3) Persistent Query Table. The first two

tables record communication messages, respectively, from OAA to RETSINA and from RETSINA to OAA. The third table is simply a variant of the second table. It is used when there are multiple facilitators in OAA. One difference between ROI and AgentGateway is that the latter records the messages to be used later, while the former does not seem to record them.

3. AgentGateway architecture

In this section, firstly, XML-based ACL is introduced. Next, tool architecture of AgentGateway is described, followed by its current implementation.

3.1. XML-based agent communication language

Extensible markup language (XML) [17] has quickly become a standard for information exchange over the web. That is exactly the reason why it is chosen in this paper as the message format for MAS communication. Actually, an XML document is an information container, which contains reusable and customizable components that can be used by the agent receiving a message. Also, new message types can be defined by the XML format. Because that XML-based communication message can be understood by agent and that it can be used to easily define new message, better interoperability can thus be expected.

There are three components of this XML-based ACL: (1) parameters of agent communication such as sender, receiver, and so on; (2) the message information; and (3) the actual content of message (see Fig. 1).

Fig. 1 shows a message in XML-based ACL. In Fig. 1, <Message> denotes the message where "id" denotes message id. The whole message contains <COM>, <MSG>, and <CONTENT> three components. <COM> specifies the parameters of communication such as "sender" for sender agent, "receiver" for receiver agent, "from" for original agent, and "to" for destination agent. <ACT> denotes agent's action. <InReplyTo> denotes identifier of the message that triggered this message submission.

<MSG> describes some information about message, including <LANG> for language used by message content (KQML in this example), and <ONTOLOGY> for domain knowledge associated with message (ONTOLOGY-1 in this example). And <CONTENT> denotes actual content of message.

Ontology refers to a set of vocabularies. A vocabulary consists of a word set that describes attributes and actions for an application domain, and how the agent system uses the vocabulary to structure interactions and access services [1]. Note that ontology is not included in either RETSINA or OAA. Thus, it is not addressed in this prototype.

```
<Message id="SFD8427">
    <COM sender="Agent1" receiver="Agent2" from="Agent3" to="Agent4">
        <ACT>reply</ACT>
        <InReplyTo/>SFD8333
    </COM>
    <MSG>
        <LANG>KQML</LANG>
        <ONTOLOGY>ONTOLOGY-1</ONTOLOGY>
    </MSG>
    <CONTENT>"This is a message"</CONTENT>
</Message>
```

Fig. 1. A message in XML-based ACL.

3.2. Tool architecture

AgentGateway is composed of three parts as shown in Fig. 2 (note that agent A belongs to MAS 1, and agent B belongs to MAS 2):

(1) *Communication:* This part contains communication interfaces to handle communication with different MAS. If an MAS needs to communicate with multiple MAS, one specific communication interface should be prepared for each.

(2) *Message:* This part contains translators, Message History, and Agent Service Registry. The translator will translate a message of another ACL into the XML-based ACL. Message History keeps these messages. When a message gets lost or AgentGateway crashes, the message can be resent based on what is kept. The Message History is directional, meaning that messages from MAS 1 to MAS 2 and those from MAS 2 to MAS 1 are kept separately. Agent Service Registry records services provided by the MAS. Both Message and Agent Service Registry can be stored externally in databases.

(3) *Management:* This part contains a web server through which manager can access messages and services in Message History and Agent Service History, respectively.

When using AgentGateway, it will register itself to both MAS, request service lists of both MAS, record those services in Agent Service Registry, and register services of one MAS to another MAS. For example, AgentGateway will register services provided by MAS 1 to MAS 2, and similarly, MAS 2 to MAS 1. By doing so, agent in MAS 1 will regard the services provided by MAS 2 as provided by AgentGateway. Similar thing holds for agent in MAS 2.

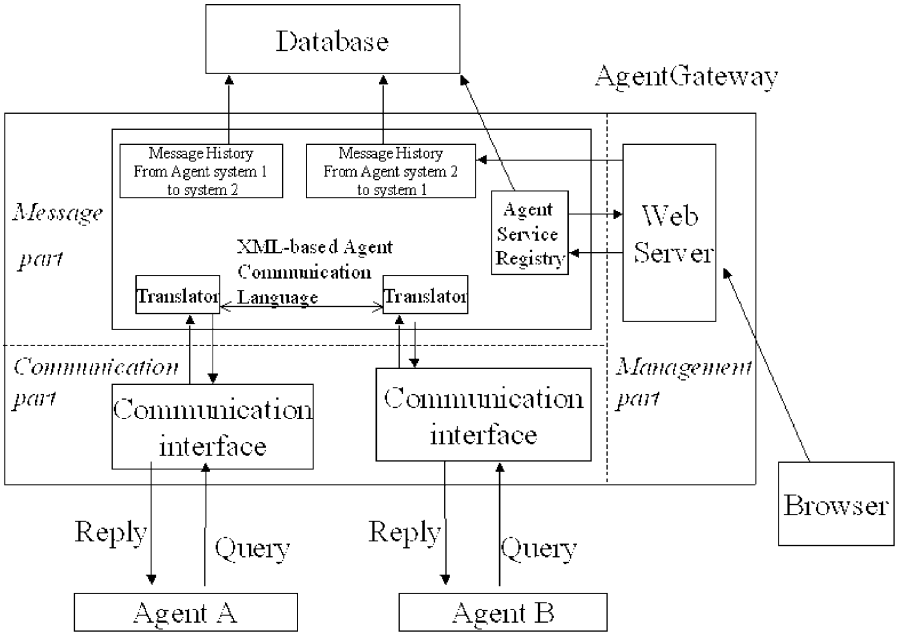

Fig. 2. AgentGateway architecture.

Therefore, when agent A requests a service from AgentGateway. The request will be sent to the communication interface to be translated first to intermediate ACL, then to ACL of MAS 2 by the translator. The result of translation will be sent to agent B where the service is actually provided. After agent B finishes processing the service, the service result is sent back to agent A via the same route.

All the messages in AgentGateway will be recorded in Message History, through which AgentGateway can maintain the state of communication between two different MAS.

Because that new agents join, and old agents leave, an MAS all the time, services provides by an MAS are ever changing. Therefore, AgentGateway will constantly monitor current services provided by each MAS, and adjust its Agent Service Registry accordingly.

3.3. Implementation

The current implementation establishes communication between JATLite [18] and OAA.

JATLite is a MAS developed by Stanford University using Java. It provides a basic infrastructure upon which agents use name and password to register with

Agent Message Router facilitator. It adopts KQML as ACL. Agent communication is built upon open Internet standards, TCP/IP, SMTP, and FTP.

On the other hand, OAA is an MAS developed by SRI. In OAA, the facilitator agent manages all communication among agents. That is, agent communication has to go through the facilitator agent. Agents do not communicate directly. And OAA adopts Prolog-based Inter-agent Communication Language (ICL) as ACL. Next, we like to discuss some implementation considerations.

3.3.1. Architecture considerations

Differences in MAS architectures should be taken into consideration in designing communication interface. For example, in JATLite, when an agent requests a service, it first requests services list from Agent Message Router. From the services list, the agent that provides the service can be located. Then, service request is sent directly to that agent to complete the service. Thus, the requesting agent needs to know the name of the service provider agent recorded in Service Registry in AgentGateway.

On the other hand, in OAA, all communication messages have to go through facilitator. An agent simply sends request to facilitator that will automatically sends the request to the agent that provides the service, if the agent has registered the service into facilitator. And facilitator will send the result back to the requesting agent. AgentGateway regards facilitator as the provider of all the services.

Regarding service monitoring, AgentGateway should constantly query the Message Router of JATLite to get the latest services list. In OAA, a trigger mechanism is provided that performs some action upon some condition. Every agent can install a trigger on itself or other agents. Taking advantage of this mechanism, AgentGateway installs a trigger on OAA facilitator in such a way that when services data in facilitator changes, the trigger will send the latest data to AgentGateway. By doing so, AgentGateway is guaranteed to have the latest services list.

3.3.2. Message translation considerations

The way AgentGateway supports message translation between JATLite and OAA agents is to translate KQML messages by JATLite and Prolog-based messages by OAA to the intermediate XML-based messages.

OAA messages use the following Prolog format:

```
Solvable (Goal, Parameters, Permissions)
```

where "Solvable" is used to advertise capabilities of agents, "Goal" reports what queries the agent should solve, while "Permissions" specify limits to access. For example, the solvable list of a mail agent may look like this:

[solvable (send (mail, ToPerson, Msg), [callback (send_mail)], [ ]), solvable (get_message (MessageNum, Msg), [callback (get_mail)], [ ])]).

On the other hand, the message format of JATLite agent looks like:

ServiceName Parameters

Let us take a look at the "get_message (MessageNum, Msg)" service of the mail agent example above. JATLite agent sends this request as follows:

get_message 3001 "This is a test"

After translation, it looks like this:

solvable (get_message (3001,"This is a test"))

OAA agent maps two variables "MessageNum" and "Msg" into "3001" and "This is a test", respectively, and then handles this message.

4. An example

In this section, a detailed example is used to illustrate the process of communication in which a JATLite agent requests a service provided by an OAA agent (see steps (1) through (8) in Fig. 3).

Initial step: When AgentGateway starts, it will register to JATLite Router and OAA Facilitator, request latest services list, and cross-register services to another system.

Step 1: JATLite agent sends a message to JATLite Router to ask for a calculation service. And the content of message is " Calculate + 3 5".

Step 2: The router thought the service were provided by AgentGateway, and returns the address of AgentGateway. Actually, OAA agent provides the service.

Steps 3, 4: JATLite agent sends a request to AgentGateway (Step 3). Upon receiving the request, AgentGateway translates it to XML-based message (see Fig. 4). Then, AgentGateway finds service provider agent, and agent system it resides, from Service Registry. In OAA, Facilitator provides all the services. AgentGateway translates XML-based message to OAA ICL, and then transmits it to Facilitator (Step 4).

Steps 5, 6, 7: Facilitator receives the request, forwards it to OAA agent (the actual provider of calculation service) for processing (Step 5), gets the result returned from OAA agent (Step 6), and returns the result to AgentGateway (Step 7).

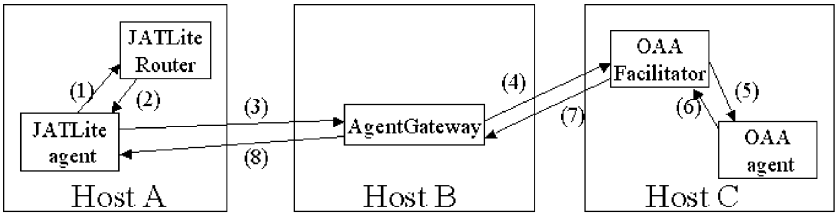

Fig. 3. A JATLite agent requests a service by an OAA agent.

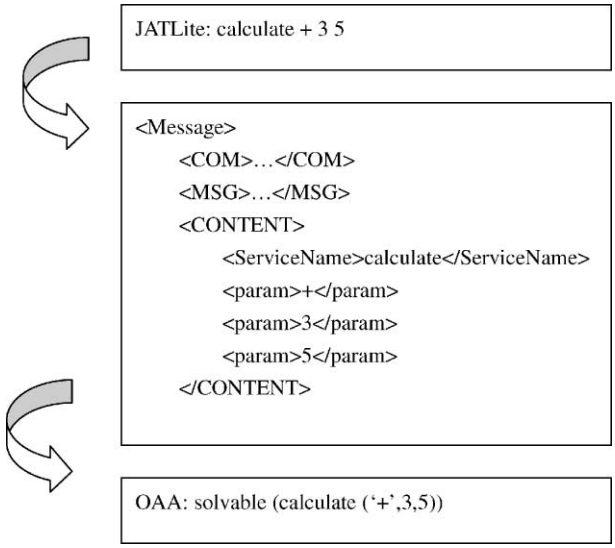

Fig. 4. Message translation.

Step 8: AgentGateway gets the result, finds the original sender, translates the message, and sends it back to the original sender.

From the viewpoint of JATLite agent, AgentGateway provides the service in the process of message transmission and translation. On the other hand, from the viewpoint of OAA Facilitator, AgentGateway makes the request. This explains why the communication is transparent.

5. Conclusions

With the rapid growth of MAS over the Internet, there definitely is a pressing need to be able to communicate among different MAS. One apparent solution is to follow a communication standard. Standards such as OMG, FIPA, KAoS, and so on have been proposed in that regard. Unfortunately,

most MAS at present do not conform to any standard due to their special needs. Therefore, this paper proposes an entirely different approach without using a standard.

A tool called AgentGateway is prototyped that translates agent communication messages from one MAS to an XML-based intermediate message, which is then translated to messages for another MAS. This prototype shows that communication between two MAS, namely, JATLite and OAA can be done using this approach. Three benefits of this tool can be expected:

(1) *Scalable architecture:* The XML-based communication message is used as intermediate ACL in this tool. For each MAS there is one communication interface that translates messages of the MAS to the XML-based messages. Assuming that the tool currently handles MAS 1 and MAS 2, and a new MAS 3 joins in. All you need to do is to build one communication interface for MAS 3, and one translator (that translates MAS 3 messages to the XML-based messages). Thus, new MAS can easily join in this tool.

(2) *Transparent communication:* Since this tool hides communication details, agents thought they were communicating only with this tool, without being aware of the existence of other MAS. Thus, MAS need not be modified for the purpose of communication. This transparency of communication clearly simplifies maintenance of the tool. Moreover, this communication is likely to serve as a basis for building high-level intelligent social behavior among MAS.

(3) *Reliable message transmission:* Since this tool keeps all the messages in persistent databases, in case that a crash occurs, a message can be resent based on what is kept in databases. This is expected to guarantee reliable communication.

Acknowledgements

The authors wish to thank supports of the Ministry of Education in Taiwan under project no.: EX-91-E-FA06-4-4.

References

[1] M.L. Griss, G. Pour, Accelerating development with agent components, IEEE Computer (2001) 37–43.
[2] T. Finin, Y. Labrou, J. Mayfield, KQML as an agent communication language, Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94), 1994.
[3] Foundation for Intelligent Physical Agents, FIPA 97 Specification Part 2: Agent Communication Language, Ver. 1.2, 1997.
[4] E.H. Durfee, V.R. Lesser, Corkill, Trends in cooperative distributed problem solving, IEEE Transactions on Knowledge and Data Engineering KDE-1 (1) (1989) 63–83.

[5] N.R. Jennings, K. Sycara, M. Wooldridge, A roadmap of agent research and development, Autonomous Agents and Multi-Agent Systems Journal 1 (1998) 7–38.
[6] K. Decker, K. Sycara, M. Williamson, Middle-agents for the Internet, Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI-97), 1997.
[7] J.M. Bradshaw, S. Dutfield, P. Benoit, J.D. Woolley, An Introduction to software agents, in: Software Agents, AAAI Press, 1997, pp. 3–46.
[8] G. Wiederhold, Mediators in the architecture of future information systems, IEEE Computer (1992) 38–49.
[9] K. Decker, K. Sycara, M. Williamson, Matchmaking and brokering, Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), December 1996.
[10] J.M. Bradshaw, S. Dutfield, P. Benoit, J.D. Woolley, KAoS: Toward an industrial-strength open agent architecture, Software Agents (1997) 375–418.
[11] H.P. Nii, Blackboard systems, in: The Handbook of Artificial Intelligence, vol. IV, 1982, pp. 1–82 [Chapter XVI].
[12] S. Virdhagriswaran, D. Osisek, P. O'Connor, Standardizing agent technology, ACM StandardView 3 (3) (1995) 96–101.
[13] Foundation for Intelligent Physical Agents, FIPA '97 Specifications Part 1: Agent Management, Ver. 1.2, 1997.
[14] Joseph A. Giampapa, Massimo Paolucci, Katia Sycara, Agent interoperation across multagent system boundaries, Proceedings of the Fourth International Conference on Autonomous Agents, 2000.
[15] K. Sycara, K. Decker, A. Pannu, M. Williamson, D. Zeng, Distributed intelligent agents, IEEE Expert, Intelligent Systems and their Applications (1996) 36–45.
[16] D. Martin, A. Cheyer, D. Moran, The open agent architecture: A framework for building distributed software systems, Applied Artificial Intelligence 13 (1–2) (1996) 92–128.
[17] Extensible Markup Language. http://www.w3.org/XML.
[18] H. Jeon, C. Petrie, M.R. Cutkosky, JATLite: a Java agent infrastructure with message routing, IEEE Internet Computing 4 (2000) 87–96.