

Agent-Based Web Service Composition with JADE and JXTA

Shenghua Liu(1), Peep K ungas(2) and Mihhail Matskin(1)

(1)Royal Institute of Technology (KTH), Isafjordsgatan 39, SE-16440 Kista, Sweden
Phone: +46 8 7904128, Fax: +46 8 7511793, Emails: shenghua@kth.se, misha@imit.kth.se

(2)Norwegian University of Science and Technology (NTNU), IDI, NO-7491 Trondheim,
Norway

Phone: +47 735 91875, Fax: +47 735 94466, Email: peep@idi.ntnu.no

Abstract. *Rapid development of the Internet and increasing number of available Web services has generated a need for tools and environments facilitating automated composition of atomic Web services into more complex Web processes. JADE is an agent development environment where Web services and agents can be linked together to enable semantic Web applications. However, the current JADE message transportation protocols do not allow agent communication through firewalls and network address translators (NAT-s). Fortunately, the firewall/NAT issue can be solved by using the current JXTA implementation for agent communication. In this paper we describe our efforts to incorporate JXTA protocols into JADE for facilitating inter-agent communication over the Internet. We also describe the design and implementation of an agent-based Web service composition environment, where service registration and discovery are resolved using the JXTA advertisements..By combining the capabilities of JADE and JXTA, agent-based Web service applications can be supported in JADE at a higher level of abstraction.*

Keywords: Automated Web service composition, multi-agent systems, P2P networks, JADE, JXTA.

1. Introduction

JADE is a FIPA compliant agent development environment which facilitates the implementation of multi-agent systems. Since Web services middleware has been integrated into JADE, agents implemented in JADE can exploit Web services as computational resources. A Web service can be published as a JADE agent service and an agent service can be symmetrically published as a Web service endpoint. Invoking a Web service is just like invoking a normal agent service. In addition, Web services' clients can also search for and invoke agent services hosted within JADE containers.

The Web Services Integration Gateway (WSIG) uses a Gateway agent to control the gateway from within a JADE container. Interaction among agents on different platforms is achieved through the Agent Communication Channel (ACC). Whenever a JADE agent sends a message and the receiver lives on a different agent platform, a Message Transport Protocol (MTP) is used to implement lower level message delivery procedures [2]. There are two main MTP-s available today to support this inter-platform agent communication - CORBA IIOP-based and HTTP-based MTP.

However, there are some problems to be considered. First, a single Web service may not always satisfy constantly changing system requirements in dynamic systems like MAS. This creates a need for automated Web service composition allowing to construct powerful, robust service network by binding together a number of collaborated agent-based Web services. Second, with the increasing complexity of large-scale agent applications, communication between agents has to handle connections behind firewalls and Network Address Translators (NAT-s). This requires an implementation of a new JADE

MTP for JADE.

JXTA is a set of open protocols for P2P networking. The JXTA protocols enable developers to build and deploy P2P applications through a unified medium. The main JXTA concepts include peers, peer groups, pipes, advertisements, and JXTA services [3]. A JXTA peer is any networked device that implements one or more of the JXTA protocols. Several peers can self-organize into peer groups, which provide a common set of services. Pipes are bound to specific endpoints at runtime, such as a TCP port and an associated IP address. The supported objects for transmission are binary code, data strings and Java-based objects.

JXTA peers advertise their services within advertisements, which enable other peers on the network to learn how to connect to, and interact with peer's services. The advertisements are XML-based documents composed of a series of hierarchically arranged elements. The JXTA's Peer Discovery Protocol (PDP) is used to discover any published resources, both on client and server sides. Obviously, JXTA is a suitable architecture for implementing MTP-s for JADE. Not only JADE agent communication can be facilitated, but also agent-based automated service composition also can be implemented conveniently by incorporating JXTA technology into JADE.

In this paper, we describe an efficient automated Web service composition algorithm which was designed for a P2P-based multi-agent environment. We also present experimental results and evaluate our implementation of an JXTA-based JADE MTP.

2. Related work

There are several articles related to automated Web service composition. Ontology-Driven Web Services Composition Platform [1] aims to generate a composite service out of semantically described existing services. In this work, the possible automatic compositions are obtained through interface-matching, which checks semantic similarities between interfaces of individual services. Different services can be integrated to satisfy user requirements.

SWORD [5] is a developer toolkit for building composite Web services. SWORD does not deploy the emerging service-description standards such as WSDL and DAML-S, instead, it uses Entity-Relation (ER) model to specify the inputs and the outputs of Web services. As a result, the reasoning is based on the entity and attribute information provided by an ER model.

Thakkar et al [7] consider dynamic composition of Web services using mediator-based agent system architecture. The mediator takes care of user queries, generates wrappers around information services and constructs a service integration plan. In [8] SHOP2 planner is applied for automatic composition of DAML-S services.

Paolucci et al [4] evaluate a broker for constructing OWL-S Web services. They also identify some drawbacks of the current OWL-S specification and propose a workaround for the problem. Advantages of applying the broker architecture, like anonymisation, trusted intermediary and communication facilitation, are emphasised there. Sycara et al [6] describe a methodology for constructing composite services written in DAML-S. Additionally the implemented DAML-S virtual machine is evaluated there. Still, the two preceding articles consider service composition as matching suitable atomic components. Similarly, the preceding articles view service advertisements as service templates, which describe the inputs/outputs and preconditions/effects of particular services. The concept of advertisements could be significantly extended with domain-specific knowledge.

3. Agent-based Web service composition

In a P2P-based multi-agent environment, any agent can publish or search services for its purposes. Although the number of available Web services increases rapidly, an atomic Web service may not satisfy agent's requirements. Thus there is a need for composing existing Web services into composite services.

For example, let us assume that there are three agents A1, A2 and A3 in the agent system. A1 has Web service S1 which provides an ISBN number for a book name. A2 has service S2 which accepts the ISBN number and returns the book price in US dollars. A3 has service S3 which converts a book price in US dollars to Swedish krone. Now, if a user is looking for a service for taking a book name for input and returning its price in Swedish krone, there are none available. Anyway, through automated Web service composition, S1, S2, S3 can be composed into a new service which accepts the book name as input and returns the book price in Swedish krone.

The composition technique aims to find an optimal composition of services considering semantic matching of parameters. Generally, our agent-based Web service composition mechanism consists of three sequential phases: (I) service registration, (II) construction of the service graph, and (III) service discovery together with composition.

In the service registration phase, agents register their services at a JXTA network by publishing their descriptions to the network as JXTA advertisements. An agent description contains an Agent Identifier (AID), a set of descriptions of available services, the list of languages and ontologies that other agents need to know in order to interact with this agent. For each published service a description is provided, including the service type, the service name, its input and output parameters, languages and ontologies required to exploit that service and a number of service-specific properties.

In the graph construction phase matching inputs and outputs of services are logically connected in a way that they form a potential workflow [1]. In order to ensure completeness of the distributed graph, every participating agent has to search its successors after publishing its services. A successor of an agent is another agent with services whose input matches with its predecessor agent's output. In the example described above, S1's successor is S2 and S2's predecessor is S1. We say that the whole system gets stabilized, if every agent has identified its successors and a complete directed service graph is constructed. In order to keep the system stabilized, agents should perform the search process periodically to update their successors.

Discovery and composition of a specific service is performed by following agents' successors recursively. When an agent posts a request with an input and an expected output parameter to the service graph, the request is relayed among associated agents in the graph. The search procedure ends when one of the following conditions is fulfilled: 1) a solution for the expected service has been found; 2) the request reaches an agent without successors and doesn't find a solution for the requested service; 3) agents detect a loop when delivering requests.

4. Optimization of the service composition algorithm

In the service composition methodology described in Section 3, a request is forwarded to all the successors if an agent cannot provide the expected service. We notice that the number of forwarded requests can be reduced if a forwarding agent knows through which successor it can reach the target service with the shortest path. In the book-selling example, assuming additionally that agent A4 provides service S4 that converts a book price in US dollars to euros and another agent A5 provides service S5, which can convert euro to Swedish krone, the corresponding service graph is depicted in

Figure 1.

Agent A2 in Figure 1 has two successors - A3 and A4. Apparently, if Swedish krone is required, it does not make sense to forward the request from A2 to A4, because this path to krone is longer than the path through A3. This kind of optimization is made during the service discovery.

In our system every agent maintains a successor table and a service table to forward the requests in an optimal way. The successor table is a set of channel/output pairs. While a *channel* represents a successor of an agent, *output* list contains all the outputs of services which can be reached through the channel. Whenever an agent finds a new successor, a record as (successor, null) is inserted into the table. The output in (channel/output) pair could be updated according to the replies from its successors. In the book-selling example, A2 has the successor table represented as follows.

Channel (successor)	Output
A3	krone
A4	euro, krone

The service table consists of (*output,channel,hops*) triples. *Output* in this table represents an output of a service. *Channel* identifies through which agents the required service can be reached. *Hops* represents the distance from the agent to a specific service. Initially, every agent has the following service table, where *agent* is the agent itself:

Output	Channel	Hops
Output of agent's service	agent	0

In the book-selling example, A2 has the service table as follows:

Output	Channel	Hops
euro	A4	1
krone	A3	1
dollar	A2	0

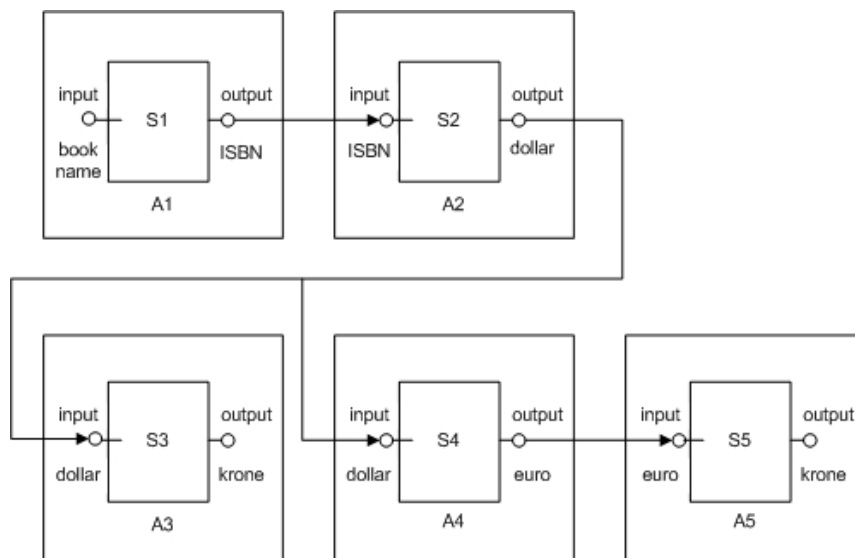


Figure 1. A constructed service graph.

In the service discovery phase, when an agent receives a request and its service happens to match the request, the agent replies to the requester agent with the output of its matching service as well as the agent's service table. When an agent receives a reply from its successors, it extracts the service table from it, and for each output in the table, the agent checks whether there exists a better path to the output. If this happens, after updating its own service table and successor table, the agent creates a new reply and sends it to its predecessor. Figure 2 depicts the shortest path algorithm.

In order to demonstrate the shortest path algorithm let us consider the case where agent A2

maintains a service table with record (krone,A4,2). When it receives the service table from its successor A3 with (krone,A3,0), it will update (krone,A4,2) to (krone,A3,1). Anyway, if A2 receives the service table from A4 with record (euro,A4,0) and euro is not contained in A2's service table, record (euro,A4,1) is inserted into the service table and euro is added to the A4's corresponding output field in A2's successor table.

When an agent receives a request, and it cannot provide a service providing the expected output, it will route the request according to its successor table and service table by using the following rules:

- If the required output is not in the agent's service table, it will forward the request to all of its successors.
- If the required output is in the service table, it forwards the request to the corresponding successor found in the service table and the successor whose output field in the successor table is null. We do that because in the second case the successor is a new channel through which the path to the target output may be shorter.

```

var ST: agent's service table; RT : successor's service table; SCT: agent's successor table;
HOPu: array of Hops in service table ; CHu:array of Channel in service table u
OUTPUTu: array of Output in service table u; o: output of a service
begin
  receive a reply from a successor;
  for each o in OUTPUTRT do
    if o is already in ST
      if HOPRT[o] + 1 < HOPST[o]
        CHST[o] := successor; HOPST[o] := HOPRT[o] + 1 ;
      else
        add o to ST in which
          CHST[o] := successor; HOPST[o] := HOPRT[o] + 1 ;
        add o to the output list in SCT
  end

```

Figure 2. The shortest path algorithm.

For instance, let us suppose that A2 has the following successor table

Channel (successor)	Output
A3	Null
A4	euro, krone

and the following service table

Output	Channel	Hops
Euro	A4	1
Krone	A4	2

When A2 receives a request for a service requiring the book price in Swedish krone, it first checks the service table and forwards the request to A4 since krone is in its service table. It also forwards the request to A3 because the A3's output in the successor table is null.

5. Comparison of JADE MTPs in inter-platform configuration

By default JADE includes two MTP implementations – IIOP- and HTTP-based. We have implemented the third alternative – JXTA-based MTP, which allows inter-platform communication behind firewalls and NATs. In this section we evaluate the performance of JXTA MTP and compare it with existing HTTP and IIOP MTP.

We use a testbed implemented by Cortese et al [2] to evaluate the performance of different JADE MTP-s. In this testbed, the performance of each MTP implementation is measured as the roundtrip time per message for a couple of agents. The roundtrip time is defined as the time required for a circular

exchange of an ACL message between a sender agent and a receiver agent. For each measurement we obtain average roundtrip time (*avgRTT*) defined as the total testing time (*t*) divided by the number messages per each couple of agents (*m*) and by the number of couples (*c*):
$$avgRTT = \frac{t}{m \cdot c}$$

The experiment was carried out on an IBM Thinkpad laptop with 2.4GHz processor and 512MB RAM, and a desktop computer with 1.6 GHz processor and 256MB RAM. We compared the results of different MTPs in the inter-platform communication scenario. Measurements were repeated for several configurations for each MTP implementation, with different number of agent couples among inter-platforms. In our experiment, every couple of agents completed at least 50 roundtrips.

Experimentation results are summarized in Figure 3. One can see there that as the number of agents increases, *avgRTT* increases as well. As Figure 3 shows, the roundtrip time for JXTA MTP rises from 1033ms to 4225ms with the number of agents increasing from 10 couples to 50 couples, grows from 4225ms to 10954ms with the number of agents increasing to 100 couples, and rises to 18720ms when the number of agents reaches 200 couples. This growth rate is proportional to the number of agents, fluctuating from 0.8 to 1.29 per couple of agents.

While comparing the test results of HTTP and IIOP MTP implementations, we found that, in the considered range, performance of MTP-s scales linearly and *avgRTT* doesn't vary significantly among different MTP-s when the number of agent couples is less than 200. For example, we can see that the roundtrip time for JXTA, HTTP and IIOP is 10954ms, 11916ms, and 10230ms respectively, based on the communication among 100 couples of agents. It shows that the performance of JXTA MTP implementation is as good as the two already existing MTP implementations.

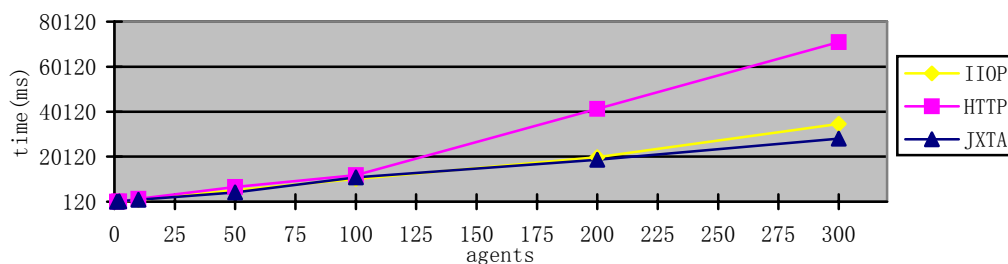


Figure 3. Roundtrip time for different MTP implementations.

If the number of agents exceeds 200 couples, *avgRTT* of HTTP implementation rises suddenly. However, the JXTA MTP performs still almost as well as IIOP MTP does, even a little better than the IIOP MTP. As shown in the Figure 3, in the case of communication among 200 couples of agents, the roundtrip time of JXTA MTP implementation is 18720ms, contrasting to the 19821ms in IIOP implementation and 41280ms in HTTP implementation. This means that the scalability of our JXTA MTP implementation is in general as good as or even better than the existing MTP implementations. Furthermore, we conclude from the results that the JXTA and IIOP MTP implementations are more suitable for large-scale agent applications than HTTP MTP implementation.

6. Conclusions and future work

In this paper we described an agent-based Web service composition prototype in which service registration and discovery issues are resolved through publishing and discovering JXTA advertisements. In the service composition algorithm, several related services are connected to a service network such that system requirements can be met by transferring the request through the service network. Moreover, we optimized the service composition algorithm such that users' requests can be delivered in the

shortest way. We proposed this automated service composition algorithm just to provide a basic, extensible prototype for Web service composition in agent-based systems, which can be extended to resolve more abstract and complicated service composition tasks.

Furthermore, we evaluated a JXTA-based inter-platform message exchange mechanism for JADE. We compared the performance of the JXTA MTP implementation with two other existing JADE MTP-s - IIOP and HTTP implementation. The experimental results showed that the performance of JXTA MTP implementation is similar to the IIOP and HTTP implementation and is even better, if the number of participating agents exceeds 100. This means that the scalability of JXTA MTP implementation is good enough to be applied in large-scale multi-agent systems.

Since our service composition method is based on the semantic similarities between descriptions of individual services, the possible compositions could be ranked by the degree of similarity and the Quality of the Service (QoS). Thus, an optimal composition could be selected by considering these two factors. This paper only considered services with one input and one output, which is the simplest case for automated service composition. This should be extended to suit more complicated services with multiple inputs and outputs.

Acknowledgements

This work was partially supported by the Norwegian Research Foundation in the framework of Information and Communication Technology (IKT-2010) program – the ADIS project.

References

- [1] B. Arpinar, B. Aleman-Meza, R. Zhang, A. Maduko. *Ontology-Driven Web Services Composition Platform*. In 2004 IEEE International Conference on E-Commerce Technology (CEC'04), July 6-9, 2004, San Diego, California, USA, pp. 146-152, IEEE Computer Society Press, 2004.
- [2] E. Cortese, F. Quarta, G. Vitaglione, P. Vrba. Scalability and Performance of the JADE Message Transport System. *Analysis of Suitability for Holonic Manufacturing Systems*, *exp*, 3(3), pp. 52-65, 2002.
- [3] J. D. Gradecki. *Mastering JXTA: Building Java Peer-to-Peer Applications*, John Wiley & Sons, 528 pages, 2002.
- [4] M. Paolucci, J. Soudry, N. Srinivasan, K. Sycara. A Broker for OWL-S Web Services. In *Proceedings of the First International Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series, March 22-24, 2004*, pp. 92-99, AAAI Press, 2004.
- [5] S. R. Ponnekanti, A. Fox. SWORD: A Developer Toolkit for Web Service Composition. In *Proceedings of The Eleventh World Wide Web Conference (Web Engineering Track), Honolulu, Hawaii, USA, May 7-11, 2002*, pp. 83-107, 2002.
- [6] K. Sycara, M. Paolucci, A. Ankolekar, N. Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1), pp. 27-46, 2003.
- [7] S. Thakkar, C. A. Knoblock, J. L. Ambite, C. Shahabi. Dynamically Composing Web Services from On-line Sources. In *Proceeding of 2002 AAAI Workshop on Intelligent Service Integration, Edmonton, Alberta, Canada, 2002*.
- [8] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau. Automating DAML-S Web Services Composition Using SHOP2. In *Proceedings of the 2nd International Semantic Web Conference, ISWC 2003, Sanibel Island, Florida, USA, October 20-23, 2003*.