TOWARD THE MARRIAGE OF XML AND MOBILE AGENTS – UTILIZING XML-RPC AS A MIGRATION MEDIA –

Kazunari MEGURO
The Graduate School of Engineering
Tottori University
4-101, Koyama-Minami
Tottori, JAPAN
email: meguro@tottori-u.ac.jp

Shinichi MOTOMRUA, Takao KAWAMURA and Kazunori SUGAHARA
Information Media Center, Faculty of Engineering
Tottori University
4-101, Koyama-Minami
Tottori, JAPAN
email: motomura@tottori-u.ac.jp, {kawamura, sugahara}@ike.tottori-u.ac.jp

ABSTRACT
In this paper, we present usages of XML-RPC and the effectiveness in mobile agent frameworks. In the mobile agent framework, XML-RPC has a usage as a migration media. To confirm the effectiveness, we implement these functions in a mobile agent framework. In order to realize a migration mechanism, custom Object Serialization is implemented to customize Java's built-in serialization mechanism. For deserialization, a dynamic class loader is implemented. If no custom Object Serialization is used, an object cannot be deserialized on a remote host, because the class description of the object may not be in the remote host.

KEY WORDS
Distributed Agent, XML-RPC, Application Interface, Migration

1 Introduction

The Extensible Markup Language (XML) is a W3C-recommended general-purpose markup language that supports a wide variety of applications. Its main purpose is to facilitate the sharing of data across different information systems, particularly in systems connected via a network. Since XML has characteristics which is content based and allows semantic data interoperability, XML has become widely used as a general purpose data format. For example, languages based on XML (such as RSS, MathML, XHTML, Scalable Vector Graphics) are defined. XML is also used for Remote Procedure Call(RPC). XML-RPC[1] uses XML to encode parameters and results and HTTP as a transport mechanism. XML-RPC is useful for developing distributed systems, in particular web applications.

Mobile agent technology is also attracting attention as a key technology for developing distributed systems. In mobile agent systems, the most important function is agent migration. Generally, RMI is used as a technology that migrates agent[2] [3], however RMI is blocked by common firewalls. HTTP protocol can work through many common firewall security measures without requiring changes to the firewall filtering rules. Therefore, HTTP protocol is suitable in the environment with firewalls as the communication protocol. An Example of a protocol based on HTTP protocol, there is XML-RPC protocol supported in various languages.

However, there is no research on mobile agent systems using XML-RPC as a migration media. On the other hand, XML-RPC is used as application interface of various systems, however there is few research[4] on mobile agent systems using XML-RPC. In this paper, we propose that mobile agent frameworks utilize XML-RPC to realize the following mechanism.

1. Migration mechanism: Agents migrate from one computer to another one using XML-RPC as a migration mechanism. RMI is usually used as the transport mechanism, however RMI is often blocked by many firewalls. HTTP connections used as the transport connections for XML-RPC can work through many common firewall security measures without requiring changes to the firewall filtering rules.

2. Compatibility of application interface: Agents have an interface which is accessible from applications, written in any other language, which support XML-RPC. XML-RPC is supported by many programming languages, such as Perl, Ruby and ECMAScript which is often referred to as JavaScript or JScript.

To confirm efficiency of XML-RPC in mobile agent frameworks, we implement a migration mechanism and an application interface using XML-RPC in a mobile agent framework. This paper is organized in 5 sections. We describe our design goals for the proposed system in Section 2. In Section 3, we describe the implementation of the proposed system. In Section 4, we present result of achievement. Finally, in Section 5, we describe some concluding remarks.

2 Usage of XML-RPC

2.1 Design of migration mechanism using XML-RPC

Migration of agents using XML-RPC is that a procedure is called with the agents as parameters. In order to realize the migration mechanism, client stubs and server stubs are necessary to make a remote procedure call look as much as possible like a local one.

1. Client stubs encode parameters to XML documents and send them over the network to server stubs in another computer. The encoding is called marshalling.

2. Server stubs decode XML documents to the parameters and dispatch them to called procedure. The decoding is called unmarshalling.

In mobile agent frameworks which are implemented in a Java environment, agents consist of Java objects. Java provides Object Serialization for object marshalling/unmarshalling. Object Serialization is a mechanism built into a core Java libraries for writing a graph of objects into a stream of data. However, only Object Serialization is not enough. If no custom Object Serialization is used, an object which is an instance of a user defined Java class cannot be deserialized on a remote host, because the class description of the object may not be in the remote host. Therefore dynamic class loaders are introduced to custom Object Serialization. A dynamic class loader contains bytecodes of Java classes of an agent and loads the Java classes if necessary. When the object of an agent is transferred to other computer, the dynamic class loader which is corresponding with the agent is also transferred. Java ObjectOutputStream class and Java ObjectInputStream class which are used in Object Serialization are extended, and the extended classes are used in custom Object Serialization. When an object is serialized by custom Object Serialization, a codebase information is recorded by `annotateClass` method which overrides `annotateClass` method of ObjectOutputStream class. The codebase information shows a position where the dynamic class loader locates. When the object is deserialized, the dynamic class loader is retrieved to load the class by `resolveClass` method which overrides `resolveClass` method of ObjectInputStream class.

2.1.1 Stubs

Figure 1 shows an overview of a client stub and a server stub which are made with custom Object Serialization. The DynamicClassLoader object implements a dynamic class loader. When objects of an agent migrate from computer A to computer B, the following steps are performed.

1. The DynamicClassLoader object which is corresponding with the agent is serialized to bytecodes by Object Serialization. The bytecodes are encoded as an XML document, then the bytecodes are transferred to B.

2. The XML document is decoded as the bytecodes on B, then the bytecodes is deserialized to the DynamicClassLoader object by Object Serialization.

3. The DynamicClassLoader object is registered in the DynamicClassLoaderRegister object on B, then the unique key object which is mapped to the DynamicClassLoader object is generated by the DynamicClassLoaderRegister object. The unique key object is a Java String object. The Java String object is encoded as an XML document, then the XML document is returned to A.

4. The XML document is decoded as the Java String object on A. When the objects of an agent is serialized to bytecodes, the String object, that is the unique key object, as a codebase information is written into the bytecodes by the `annotateClass` method. The bytecodes are encoded as an XML document, then the bytecodes are transferred to B.

5. The XML document is decoded as the bytecodes on B. When the bytecodes try to be deserialized, firstly the unique key object is retrieved by the `resolveClass` method. Then the DynamicClassLoader object which is mapped to the unique key object is retrieved from the DynamicClassLoaderRegister object. Finally, the bytecodes are deserialized to the object of the agent with the DynamicClassLoader by loading the classes.

2.1.2 XML Format

To use XML-RPC as a migration mechanism, an XML format has to be defined. In XML-RPC, there are XML-RPC requests and XML-RPC responses. In an XML-RPC request, the payload is in XML, a single <methodCall> structure. The <methodCall> must contain a <methodName> sub-item, a string, containing the name of the method to be called. If the procedure call has parameters, the <methodCall> must contain a <params> sub-item. The <params> sub-item can contain any number of <param>s, each of which has a <value>. In order to transfer objects of an agent, the name of an agent and the bytecodes of the objects as <param>s are necessary. Moreover, the bytecodes have to be encoded as a base64 string because an XML-RPC message is an HTTP-POST request. Figure 2 shows an example of the XML document in an XML-RPC request when objects of an agent is transferred. In an XML-RPC response, the body of the response is a single XML structure, a <methodResponse>, which can contain a single <params> which contains a single <param> which contains a single <value>. In order to return a transfer result, a boolean data type as a <param> is suitable. Figure 3 shows an example of the XML document in an XML-RPC response when a transfer succeeds. The <methodResponse> could also contain a <fault> which contains a <value> which is a <struct> containing two elements, one named <faultCode>, an <int> and one named <faultString>, a <string>. If a transfer is failed, the reason as a <string> can be returned.

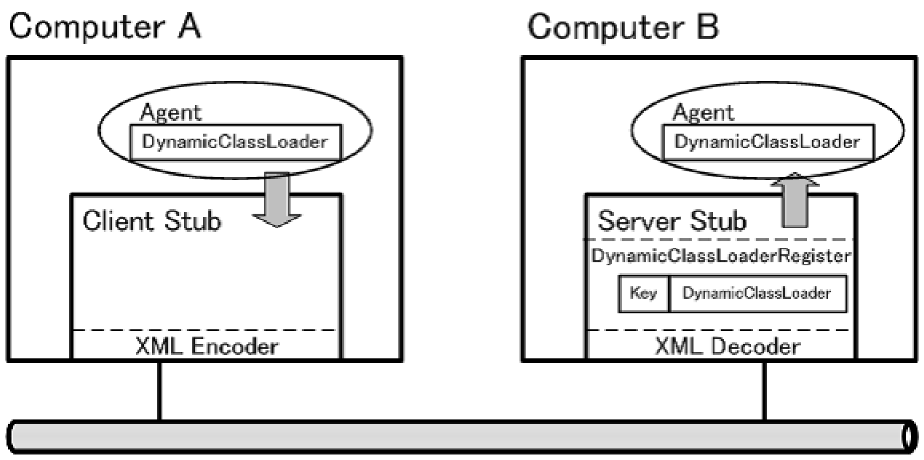
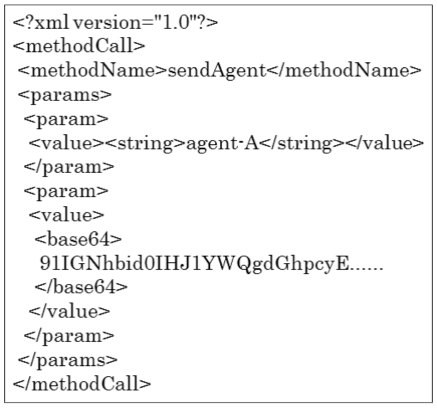
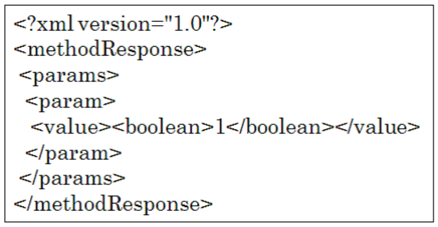


Figure 1. An overview of a client stub and a server stub which are made with custom Object Serialization.



```
<?xml version="1.0"?>
<methodCall>
 <methodName>sendAgent</methodName>
 <params>
  <param>
   <value><string>agent-A</string></value>
  </param>
  <param>
   <value>
    <base64>
     91IGNhbid0IHJ1YWQgdGhpcyE......
    </base64>
   </value>
  </param>
 </params>
</methodCall>
```

Figure 2. An example of the XML document in an XML-RPC request when an agent is transferred.

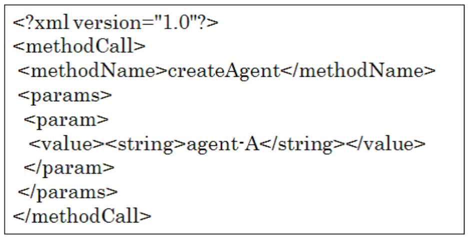


```
<?xml version="1.0"?>
<methodResponse>
 <params>
  <param>
   <value><boolean>1</boolean></value>
  </param>
 </params>
</methodResponse>
```

Figure 3. An example of the XML document in an XML-RPC response when a deserialization succeeds.

2.2 Design of Application Interface

In mobile agent systems, the methods that application interface need is create agent, kill agent, transfer result and so on. Figure 4 shows an example of XML document in an XML-RPC request when objects of agent is created. To use XML-RPC as an application interface, also an XML format has to be defined. In an XML-RPC request, the body of the request is a single <methodCall> structure. In order to create agent, the parameter of <methodName> is "createAgent". If creation of an agent succeed, an example of the XML document becomes it as well as Figure 3. In an XML-RPC response, also the body of the response is a single <methodResponse> structure. If create agent is succeed, 1 is returned as a data of <value> .



```
<?xml version="1.0"?>
<methodCall>
 <methodName>createAgent</methodName>
 <params>
  <param>
   <value><string>agent-A</string></value>
  </param>
 </params>
</methodCall>
```

Figure 4. An example of the XML document in an XML-RPC request when an agent is created.

3 Implementation

Maglog[5] which is our proposed mobile agent framework is took up as an example of a framework. It is based on Prolog and is implemented by extending PrologCafe[6], which is a Prolog-to-Java source-to-source translator system. Java is adapted because of its huge class libraries to build network applications. In Maglog, agent communicates directly with other agents through a object by the name of a field.

XML-RPC is used for two purposes in Maglog. One is an agent migration mechanism that is realized by using client stubs and server stubs. The other is application interface which is accessible from any other language.

3.1 Client Stubs and Server Stubs

We implement client stubs and server stubs in Maglog. Figure 5 shows a UML diagram which is an overview of Maglog classes. The client stub is implemented by MaglogXmlAgentSerializer class. MaglogXmlAgentSerializer class is extended from ObjectOutputStream class which defined in the java.io package and is implemented the serialization algorithm. The server stub is implemented by MaglogXmlAgentDeserializer class and DynamicClassLoaderRegister class. For deserialization, MaglogXmlAgentDeserializer class is extended from ObjectInputStream class which defined in the java.io package and is implemented the deserialization algorithm. The XML encoder and the XML decoder are implemented by MaglogXmlRpcInterface class. MaglogXmlRpcInterface class has functions as an XML-RPC client and an XML-RPC server, so that the class handles an XML-RPC request and returns an XML-RPC response. In addition, the class provides an XML-RPC interface which is accessible from applications. The class is implemented by Apache XML-RPC[7].

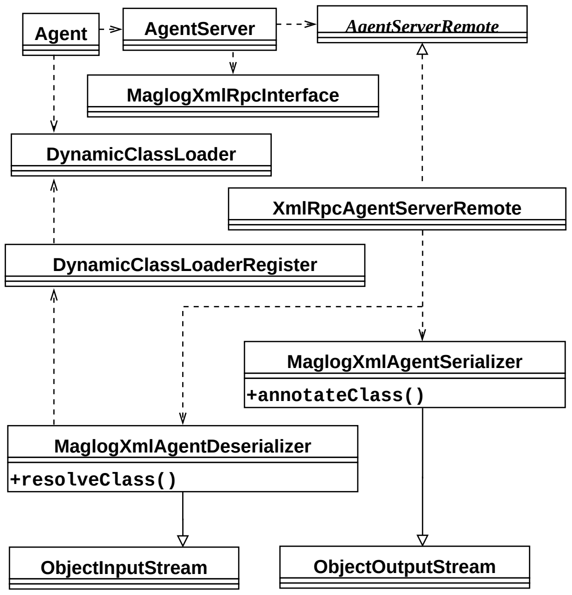


Figure 5. A UML diagram which is an overview of classes of Maglog.

3.2 Communication with an Agent

An agent server has an XML-RPC interface, which is accessible from applications written in any other languages which support XML-RPC. The following operations from other systems are available through XML-RPC.

1. Create and kill agents
2. Create and delete fields
3. Assert clauses into fields and retract clauses from fields
4. Get a list of names of fields
5. Get a list of IDs of agents currently existing

Table 1. Method names of XML-RPC interface in Maglog.

No.	Method Name	Operation
1	createAgent or killAgent	an agent is created or is killed
2	createField or deleteField	a field is created or is deleted
3	fieldAssert or fieldRetract	a clause is added in a field or is deleted from a field
4	getFieldListTerm	a list of names of fields is gotten
5	getAgentListTerm	a list of IDs of agents is gotten

The method names of XML-RPC interface are summarized in Table 1.

An application communicates with an agent by writing data in a field. Table 2 shows the relations between data type of Maglog and XML-RPC.

Table 2. The relations between data type of Maglog and XML-RPC.

Maglog	XML-RPC
IntegerTerm	<i4> or <int>
SymbolTerm	<string>
DoubleTerm	<double>
ListTerm	<array>
StructureTerm or VariableTerm	<struct>

4 Experiments

4.1 Agent Migration

This section presents the experimental results for agent migration using XML-RPC, comparison of agent's migration time on each agent's size, and comparison of agent's migration time between RMI and XML-RPC. The following experiments are examined using RMI and XML-RPC.

- Seven agents migrate between two PCs 100 times.

Table 3. The experimental conditions.

CPU	Intel Pentium4 3.2GHz
Memory	1GB
Network	1000Base-T
OS	CentOS 5
JRE	1.6.0

Each size of the seven agents is 10 KB, 50 KB, 100 KB, 500KB, 1MB, 10MB and 50MB. Table 3 shows the experimental conditions. Figure 6 shows the experimental code of the agents. The average times are summarized in Figure 7. In both cases of using RMI and using XML-RPC, when a size of an agent is smaller, the migration time is shorter. Moreover, in Each size of agent,the migration speed using RMI is more superior than that of XML-RPC.

```
Start:- goto(100,DestA).

goto(N,Dest):-
        N1 is N-1,
        N1 >= 0,
        go(Dest),
        setDest(Dest1,Dest),
        goto(N1,Dest1).
goto(0,Dest).

setDest(DestA,DestB).
setDest(DestB,DestA).
```

Figure 6. The agent migrates 100 times between DestA and DestB.

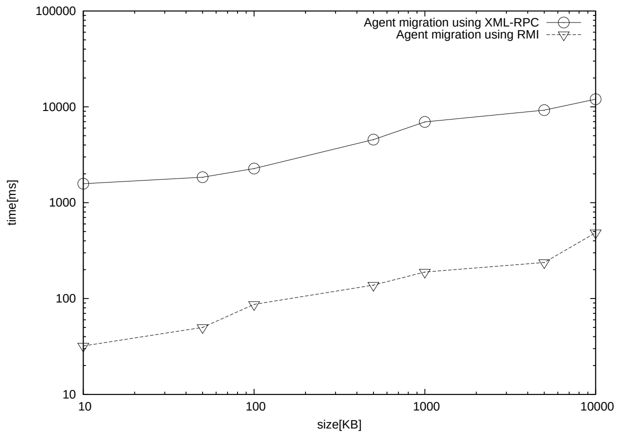
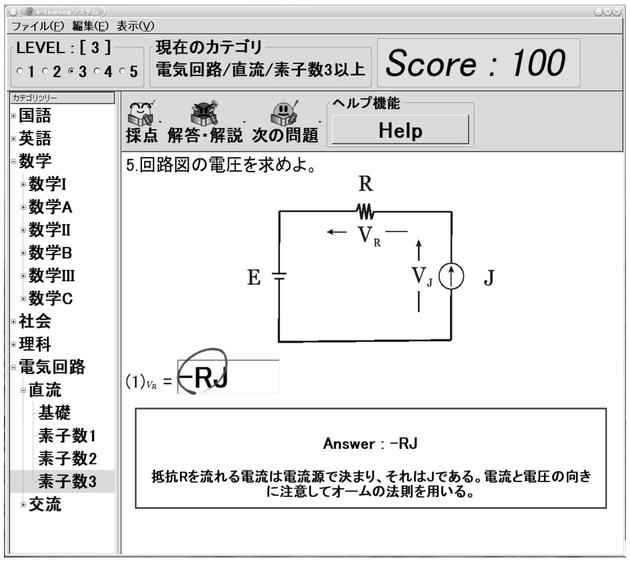


Figure 7. Migration times which seven agents migrate between two PCs and the jar files of the agents are migrated between two PCs.

4.2 Application Interface

We have developed a distributed e-Learning[8] [9] system which has been built using Maglog, and the user interface program of the application is accessed by using XML-RPC to an agent server. Figure 8 shows a screen-shot of the user interface program of the system. The program is developed as a plug-in program of Firefox web browser with Javascript and XUL which provides a powerful set of user interface widgets for creating menus, toolbars, tabbed panels, and hierarchical trees. The program communicates with an agent server using XMLHttpRequest asynchronously. For Example, when a user requests an exercise, the program sends a fieldAssert request as encoded XML to an agent server. After that, the program can accept other request from the user without waiting for the response from the agent server.



LEVEL : [3] ○1 ○2 [✓ 3] ○4 ○5

現在のカテゴリ
電気回路/直流/素子数3以上

Score : 100

5.回路図の電圧を求めよ。

(1)V_R = −RJ

Answer : −RJ

抵抗Rを流れる電流は電流源で決まり、それはJである。電流と電圧の向きに注意してオームの法則を用いる。

Figure 8. A user interface of the distributed e-Learning system

5 Conclusion

We proposed two usages of XML-RPC in mobile agent system. One was migration mechanism, the other was application interface. In order to realize the migration mechanism, custom Object Serialization has been implemented to customize Object Serialization. We confirmed two usages of XML-RPC by implementing XML-RPC in Maglog. In the experiment of agent migration, migration speed of agents using XML-RPC was compared with the time using RMI. Migration speed of agents using XML-RPC is slower than that of using RMI. If there are not firewalls among computers, RMI can be used to migrate agents faster. However, most network systems have the firewall. Therefore, in common mobile agent systems, XML-RPC is efficient as compared with RMI. We are implementing persistence in Maglog. The persistence is introduced for suspending agents. For example, when a computer will be stopped while a system is running on Maglog, agents are suspended then their suspended state and data which agents have are stored. When the system is running again, the agents' states and the data are retrieved then the agents are resumed. In order to realize the persistence, agents have to be serialized. In future work, we consider that the efficient utilization of XML databases which can store serialized agents written in XML form. Furthermore, we consider the improved implementation of XML-RPC to reduce the difference of migration speed with RMI.

References

[1]Winer, D.: XML-RPC Specification, http://xmlrcp.com/spec, (1999).

[2]Osuga, A., Nagai, Y., Irie, Y., Hattori, M. and Honiden, S.: Plangent: An Approach to Making Mobile Agents Intelligent, *IEEE Internet Computing*, Vol.1, No.4, pp.50-57 (1997).

[3]General Magic: Odyssey. http://www.genmagic.com/agents/odyssey.html.

[4]van EngelenR, GallivanK, G. G. C.G.: XML-RPC agents for distributed scientific computing, *Proceedings of the IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation* (2000).

[5]Motomura, S., Kawamura, T. and Sugahara, K.: Logic-Based Mobile Agent Framework with a Concept of “ Field ”, *IPSJ Journal*, Vol. 47, No.4, pp.1230-1238 (2006).

[6]Banbara, M. and Tamura, N.: Translating a Linear Logic Programming Language into Java, *Proceedings of the ICLP' 99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages (M.Carro, I.Dutra et al., eds.)*, pp.19-39(1999).

[7]Apache Software Foundation: About Apache XML-RPC, http://ws.apache.org/xmlrpc/.

[8]Kawamura, T., Kinoshita, S. and Sugahara, K.: A Mobile Agent-Based P2P e-Learning System, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems (Gonzalez, T., ed.)*, pp. 873-877(2004). MIT, Cambridge, USA.

[9]Motomura, S., Kawamura, T., Nakatani, R. and Sugahara, K.: P2P Web-Based Training System Using Mobile Agent Technologies, *Proceedings of the 1st International Conference on Web Information Systems and Technologies*, pp.202-205 (2005). Miami, USA.