Proceedings of the 18th IASTED International Conference
Parallel and Distributed Computing and Systems
November 13–15, 2006, Dallas, TX, USA
ISBN Hardcopy: 0-88986-638-4 / CD: 0-88986-640-6

COMBINATION OF XML-RPC AND MOBILE AGENT TECHNOLOGIES

Shinichi MOTOMURA
The Graduate School of Engineering, Tottori University
Tottori University
4–101, Koyama-Minami
Tottori, JAPAN
motomura@tottori-u.ac.jp

Takao KAWAMURA, and Kazunori SUGAHARA
Department of Information and Knowledge Engineering
Tottori University
4–101, Koyama-Minami
Tottori, JAPAN
{kawamura,sugahara}@ike.tottori-u.ac.jp

ABSTRACT

In this paper, we present usages of XML-RPC and the effectiveness in a mobile agent framework named Maglog which is implemented in a Java environment. XML-RPC is used for the following two purposes in Maglog. First, it is a transport mechanism that an agent migrates from one computer to another one. Second, it is an interface which is accessible from applications, written in any other language, which support for XML-RPC. In order to realize an agent migration mechanism, custom serialization mechanism is implemented to customize Java's built-in serialization mechanism. In custom serialization mechanism, objects which represent for an agent are encoded as an XML document. Encoded objects are transferred using XML-RPC. For deserialization, a dynamic class loader is implemented. If no custom serialization is used, an object cannot be deserialized on a remote host, because the class description of the object may not be in the host.

As an example of an interface which is accessible from applications, we show a user interface program of a distributed e-Learning system which we have developed.

KEY WORDS

Distributed software systems and applications, XML-RPC, XML, Mobile agent, Java.

1 Introduction

Recently, XML-RPC[1] is attracting attention as a technology for developing web applications. XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism. XML-RPC is used to develop web services[2], because it is a very simple protocol, defining only a handful of data types and commands. On the other hand, Ajax[3], shorthand for Asynchronous JavaScript and XML, is often used to create interactive web applications recently. In Ajax, XMLHttpRequest Object[4] is used to exchange data asynchronously with a web server. XMLHttpRequest is an API that can be used by JavaScript, JScript, VBScript and other web browser scripting languages. The API is used to transfer and to manipulate XML data to a web server and from a web server using HTTP. In many cases, XML-RPC is implemented to handle client's requests using XMLHttpRequest in web servers.

We propose a mobile agent framework named Maglog[5] which is implemented in a Java environment. In this paper, we describe the following two usages of XML-RPC in Maglog and present the effectiveness. First, it is a transport mechanism that an agent migrates from one computer to another one. Second, it is an interface which is accessible from applications, written in any other language, which support for XML-RPC. For agent migration, an agent is encoded as an XML document and is transferred using HTTP. Maglog has another transport mechanism, RMI. RMI is more superior to XML-RPC from the viewpoint of the migration speed. On the other hand, XML-RPC is more firewall friendly than RMI. 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. RMI may often be blocked.

Adopting XML-RPC in our system, existing many tools to process XML documents can be utilized efficiently. For example, SAX and DOM as programming APIs, Java Architecture for XML Binding (JAXB)[6] as an XML Processing API, and NeoCore[7] and Xindice[8] as XML databases. We consider another usage of XML-RPC in this system for providing persistence of agents. That is, to store the agents in XML databases and to retrieve them.

2 Overview of Maglog

Maglog is a mobile agent framework based on Prolog and is implemented in a Java environment by extending Prolog-Café[9], which is a Prolog-to-Java source-to-source translator system. Figure 1 shows an overview of a mobile agent system built by using Maglog. In the figure, two computers (hereafter referred to as hosts) are connected to a network and agent servers are running on each of them to activate agents and to provide fields for them.

The remainder of this section describes the three basic components of Maglog, namely, agent, agent server, and field.

2.1 Agent

An agent has the following functions:

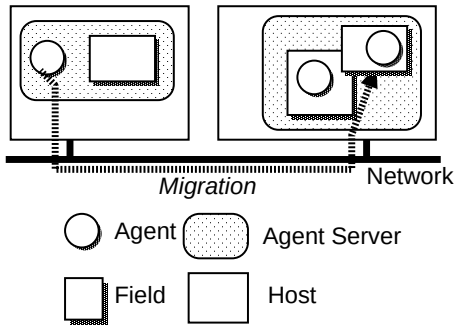

Figure 1. Overview of a mobile agent system built using Maglog.

1. Execution of a program that describes the behavior of the agent,
2. Execution of procedures stored in a field where the agent is currently located,
3. Communication with other agents through a field,
4. Creation of agents and fields,
5. Migration to another host in a network.

An agent has Prolog interpreter because the class of migration is strong migration, which involves the transparent migration of an agent's execution state as well as its program and data. If an agent is running on JVM directly without Prolog interpreter, when the agent migrates to another host, the agent cannot perform strong migration. Because Java API does not provide methods to access program counter and stack in order to access a thread's execution state. In contrast, Maglog has the WAM[10] as Prolog interpreter which is an abstract machine tailored to Prolog.

2.2 Agent Server

An agent server is a runtime environment that provides required functions for agents. For example, an agent server provides a migration function. When an agent migrates from host-A to host-B, the agent server on host-A suspends the agent's execution and transports the agent to host-B. After that, the agent server on host-B resumes execution of the agent.

An agent server also manages fields and provides functions that enable an agent to utilize them. An Agent server has an XML-RPC interface, which is accessible from applications written in any other language with support for XML-RPC.

2.3 Field

A field is an object managed by an agent server to hold Prolog clauses. An agent communicates with other agents indirectly through fields so that the following built-in predicates are provided in Maglog:

```
fassert(Clause, Field)
fretract(Clause, Field)
```

The first argument `Clause` of these predicates is a clause to be added or removed from the field specified by the second argument `Field`. `fassert/2` inserts the clause in front of all the other clauses with the same functor and arity. Functor and arity mean the name of a predicate and its number of arguments, respectively. `fretract/2` removes the next unifiable clause that matches the argument from the field. This built-in predicate is re-executable, that is, each time it is executed it attempts to remove the next clause that matches its argument. If there are no more clauses to remove, then this predicate fails.

By using these predicates, an agent can communicate with other agents not only asynchronously but also synchronously. An agent has two modes for execution of procedures stored in a field. In the fail mode, the execution fails when an agent attempts to execute or to retract a non-existent clause in a field. In the block mode, an agent that attempts to execute or to retract a non-existent clause in a field is blocked until another agent adds the target clause to the field. For agents in the block mode, a field can be used as a synchronous communication mechanism such as a tuple space in the Linda model[11].

3 Usage of XML-RPC

XML-RPC is a remote procedure call protocol using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, and processed and returned. An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on a server and the value it returns is also formatted in XML. Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.

XML-RPC is used for the following two purposes in Maglog. First, it is an agent migration mechanism. Second, it is an interface which is accessible from applications, written in any other language, which support for XML-RPC. The remainder of this section describes the two usages of XML-RPC.

3.1 Agent Migration Mechanism

In Maglog, an agent consists of Java objects (hereafter referred to as objects). Therefore, migration of agents is that objects and Java classes (hereafter referred to as classes) of the objects are transferred. In order to transfer objects and classes, marshalling mechanism and dynamic class loading mechanism are required. The mechanisms are described

the following from the viewpoints of marshalling and unmarshalling.

In order to transfer objects, they must be marshalled. Java provides object serialization for object marshalling. Serialization is a mechanism built into the core Java libraries for writing a graph of objects into a stream of data. This stream of data can be programmatically manipulated, and a deep copy of the objects can be made by reversing the process. This reversal is often called deserialization or unmarshalling.

An XML-RPC message is an HTTP-POST request. The body of the request must be in XML so that a serialized object is encoded as a base64 string. In addition, custom serialization is required. If no custom serialization is used, an object cannot be deserialized on a remote host, because the class description of the object may not be in the host. Therefore when an object is serialized, an annotation is used to record a codebase information.

Figure 2 shows an overview of the agent migration mechanism which is used in Maglog. Agent-A has a DynamicClassLoader which contains bytecodes of agent-A's classes and loads the classes if necessary. When agent-A migrates from AgentServer-A to AgentServer-B, the following two steps are performed. First step is transfer of the DynamicClassLoader contained in agent-A, and in the second step, agent-A is transferred.

The first step is performed in the following manner. First, the DynamicClassLoader is serialized, then it is encoded as an XML document using XMLEncoder. The DynamicClassLoader is transferred to AgentServer-B using XML-RPCHandler. XML-RPCHandler in AgentServer-A sends an XML-RPC request to XML-RPCServer in AgentServer-B. When XML-RPCServer in AgentServer-B receives the request, the XML document in the request is dispatched to XMLDecoder. Then the XML document is decoded, and the objects which represent for the DynamicClassLoader are deserialized. Finally the DynamicClassLoader is dispatched to DynamicClassLoaderRegister, then the key which is mapped to the DynamicClassLoader is generated by DynamicClassLoaderRegister and is returned to AgentServer-A.

Second step is performed in the following manner. Agent-A is serialized using MaglogSerializer, and the above key is recorded in the bytecodes as an annotation. Then agent-A is transferred to AgentServer-B using XML-RPCHandler. XML-RPCHandler in AgentServer-A sends an XML-RPC request to XML-RPCServer in AgentServer-B. Listing 1 shows the XML document in the XML-RPC request. The <methodName> is sendAgent. The name of agent-A is contained within first <param>. The type of the value which is contained within the <param> is <string>. Second <param> contains the base64 string which is encoded from the bytecodes of the classes. The type of the value which is contained within the <param> is <base64>. When XML-RPCServer in AgentServer-B receives the request, the XML document in the request is dispatched to XMLDecoder. Then the XML document is de-

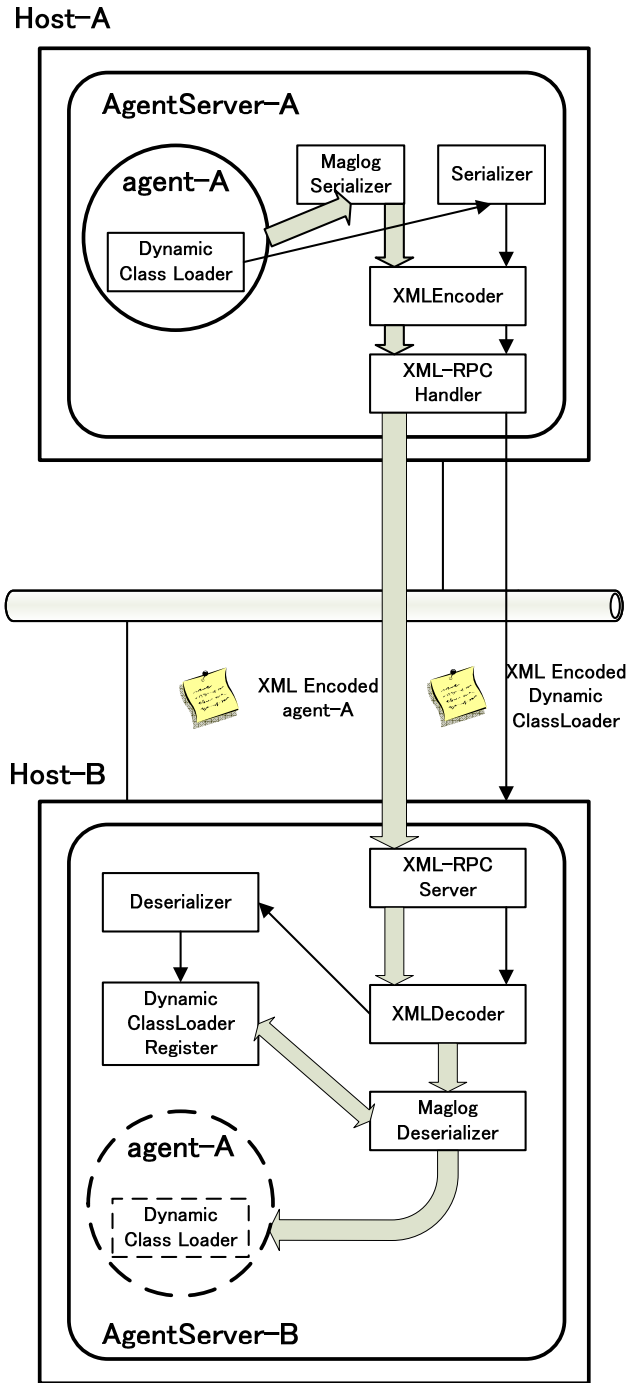

Figure 2. An overview of the agent migration mechanism which is used in Maglog.

coded, and the objects which represent for agent-A are dispatched to MaglogDeserializer. MaglogDeserializer takes the DynamicClassLoader which is mapped to the key from DynamicClassLoaderRegister. If the class descriptions of agent-A are not in AgentServer-B, agent-A is deserialized using the DynamicClassLoader. When the deserialization succeeds or not, the XML-RPCServer returns an XML-RPC response. Listing 2 shows an XML document when a deserialization succeeds. The value which is contained within the <param> is 1(true), and the type of the value is <boolean>. Listing 3 shows an XML document when a deserialization fails. The type of one of the <param>s is <boolean>, and the value which is contained within the <param> is 0(false). For the reason which is failed, a base64 string is contained within the other <param>, and the type of the value which is contained within the <param> is <base64>. The base64 string is that an exception object is encoded as a string.

Listing 1. An example of the XML document in an XML-RPC request when an agent is transferred.

```
<?xml version="1.0"?>
<methodCall>
  <methodName>maglog.sendAgent</methodName>
  <params>
    <param>
      <value><string>agent−A</string></value>
    </param>
    <param>
      <value>
        <base64>
          91IGNhbid0IHJlYWQgdGhpcyE.....
        </base64>
      </value>
    </param>
  </params>
</methodCall>
```

Listing 2. An example of the XML document in an XML-RPC response when a deserialization succeeds.

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

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

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><boolean>0</boolean></value>
    </param>
    <param>
      <value>
        <base64>
          Diens03S4Ho4e4.....
        </base64>
      </value>
    </param>
  </params>
</methodResponse>
```

3.2 Application Interface

An agent server has an XML-RPC interface, which is accessible from applications written in any other language with support for 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.

The method names of XML-RPC interface are shown 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.

We have developed an application written in Maglog, and the user interface program of the application is accessed by using XML-RPC to an agent server. We describes about the application below in Section 6.

4 Implementation

Figure 3 shows a UML diagram which is an overview of classes of Maglog. Maglog has not only XML-RPC but also RMI as a transport mechanism. Therefore, there are XmlRpcAgentServerRemote class and RmiAgentServerRemote class that implements AgentServerRemote interface and each of the classes can be used selectively according to intended purpose.

In order to customize Java's built-in serialization mechanism, MaglogXmlAgentSerializer class is extended from ObjectOutputStream class which defined in the java.io package and is implemented the serialization algorithm. And annotateClass() method is overridden to record

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

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>

a codebase information which is a key generated by DynamicClassLoaderRegister class. For deserialization, MaglogXmlAgentDeserializer class is extended from ObjectInputStream class which defined in the java.io package and is implemented the deserialization algorithm. And resolveClass() method is overridden to load a class using DynamicClassLoader 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[12].

5 Experiments

This section presents the experimental results for comparison of agent's migration speed between RMI and XML-RPC. In the experiment, two PCs with an Intel Xeon 3.4 GHz processor and 1 GB of RAM are connected via a 1000Base-T network. TurboLinux Server10 is used as the operating system. The version of the Java language runtime environment is 1.4.2. For the experiments, 100 times migratoin of one agent between two PCs are examined, and the average times are summarized in **Table 3**. Migration of agents using XML-RPC is about a quarter speed of using RMI.

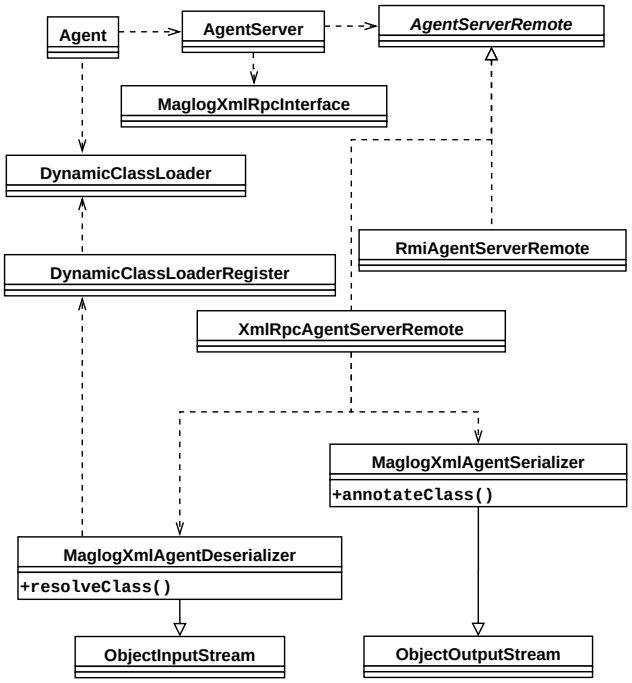

Figure 3. A UML diagram which is an overview of classes of Maglog.

Table 3. Migration times which an agent migrates from one computer to another one.

RMI	37.90 msec
XML-RPC	152.68 msec

6 Application

A distributed e-Learning system[13, 14] has been built using Maglog. This e-Learning system has two distinguishing features. Firstly, it is based on P2P architecture for scalability and robustness. Secondly, each content in the system is not only data but an agent so that it can mark user's answers, tell the correct answers, and show some extra information without human instruction. Maglog plays an important role to realize the both features.

Figure 4 shows a screen-shot of the a 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 an 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.

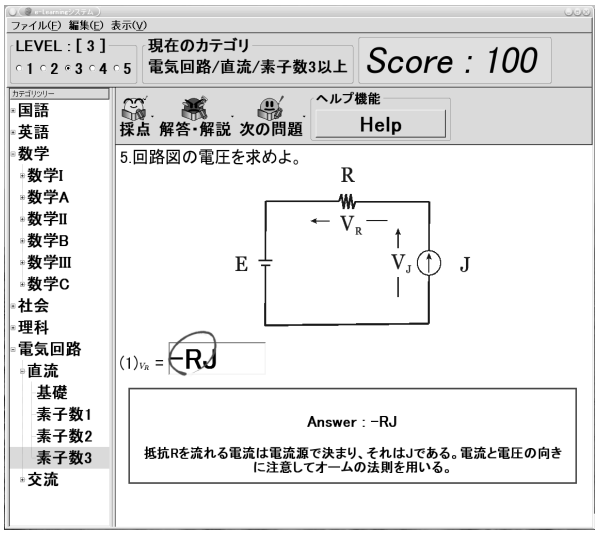

Figure 4. A screen-shot of a user interface of the distributed e-Learning system.

7 Conclusion

We have developed a mobile agent framework named Maglog, and XML-RPC has been used for the following two purposes. First, it is a transport mechanism that an agent migrates from one computer to another one. Second, it is an interface which is accessible from applications, written in any other language, which support for XML-RPC. In order to realize an agent migration mechanism, custom serialization mechanism has been implemented to customize Java's built-in serialization mechanism. In custom serialization mechanism, objects which represent for an agent are encoded as XML documents. Encoded objects are transferred using XML-RPC. For deserialization, a dynamic class loader has been implemented.

We have developed a distributed e-Learning system which has been built using Maglog. The user interface program of the system communicates with agents using XML-RPC.

Since XML-RPC provides all network functions requiring for Maglog, the implementation of Maglog can be simple, and an agent server only opens one port so that firewall filtering rules may not be required to change.

In future work, we consider that the efficient utilization of XML databases which can store agents written in XML form.

References

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

[2] The World Wide Web Consortium: Web Services Activity, http://www.w3.org/2002/ws/.

[3] Garrett, J. J.: Ajax: A New Approach to Web Applications, http://adaptivepath.com/publications/essays/archives/000385.php.

[4] The World Wide Web Consortium: The XMLHttpRequest Object, http://www.w3.org/TR/XMLHttpRequest/.

[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] Microsystems, S.: Java Architecture for XML Binding, http://java.sun.com/webservices/jaxb/.

[7] Xpriori LLC: NeoCore, http://www.xpriori.com/.

[8] Apache Software Foundation: Apache Xindice, http://xml.apache.org/xindice/.

[9] 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).

[10] Aït-Kaci, H.: Warren's Abstract Machine A Tutorial Reconstruction (1999).

[11] Carriero, N. and Gelernter, D.: Linda in Context, *Communications of the ACM*, Vol. 32, No. 4, pp. 444–458 (1989).

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

[13] Kawamura, T. and Sugahara, K.: A Mobile Agent-Based P2P e-Learning System, *IPSJ Journal*, Vol. 46, No. 1, pp. 222–225 (2005).

[14] 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.