Design of a Programming System for Mobile Objects

KAZUAKI MAEDA
Department of Business Administration and Information Science,
Chubu University
1200 Matsumoto, Kasugai, Aichi 487-8501, JAPAN
Email: kaz@solan.chubu.ac.jp

Abstract: This paper describes Escar programming language and Yare tool set for object serialization to implement mobile objects. We are engaged in the development of RoboCup soccer agents with mobile objects. To implement it, a new programming language Escar was designed. The main idea is to use time as a primary key for search. It leads us to dynamically reconfigurable systems. Moreover, a new object serialization system was designed. It supports representation independence and language independence. The result is that it is more powerful than the object serialization in Java. We believe that Escar and Yare are indispensable tools used to implement agents with mobile objects. The basic principles are described in this paper.

Key-Word: Agents, Mobile Objects, RoboCup, Object Serialization, Java

1 Introduction

For the last decade, many researchers have devoted their efforts toward building multiagent systems loosely coupled agents[1, 2]. The agents coordinate their actions for mutual benefit. Moreover, they help one another to achieve their goal. Due to the growth of the computing power and the proliferation of networking, the multiagent researches become very important.

For the last few years, there have been many projects for RoboCup(The World Cup Robot Soccer)[3, 4, 5]. RoboCup has been proposed as a standard problem to promote the research of multiagent systems. In RoboCup, robots or agents play a soccer game under given constraints. It consists of some competition tracks, for example, simulator league, real robot small size league, and real robot middle size league. Our agents, called Kasuga-bito II, are software programs written in C for the simulator league. Kasuga-bito II was runner-up in the JapanOpen '98 [6], and was champion in RoboCup Pacific Rim Series '98 [7].

Now Kasuga-bito II is rewritten in Java and it utilizes inter-agent communication by mobile objects. If a soccer agent exhausts its stamina, it does not play enough. In that case, our agent can send its own objects to another agent who has more stamina and delegate its behavior. It is called behavior delegation.

The behavior delegation requires one to load the classes into another remote agent. To do that, we can use ClassLoader in standard Java class libraries[8]. However, the ClassLoader is so primitive that the programming work is error-prone.

Moreover, the behavior delegation requires one to write the state of objects and to send it to another agent. To do that, we can use object serialization. However, the object serialization in Java is not so powerful considering the experience of the development of our agents.

Consequently, a new programming language, Escar, and the runtime environment were designed to build soccer agents with mobile objects capability. The language is syntactically the extension of the Java programming language, but the runtime environment is completely different. Furthermore, a new tool set Yare for object serialization was designed to write/read the state of objects. It supports representation independence and language independence.

In section 2, we will explain RoboCup and the experience of implementing mobile objects in Java. In section 3, we will explain the design of Escar programming language. In the section 4, we will explain Yare tool set for object serialization. In the conclusion, the paper will be summarized.

2 Soccer Agents and its Implementation in Java

2.1 Soccer Agents in RoboCup

RoboCup has features different from typical traditional AI problems such as computer chess in that situations change dynamically and in real-time. Therefore, we cannot apply the traditional techniques to RoboCup. To develop a high level robot soccer game, we must investigate some new techniques for individual agents and teams.

We have been investigating only the software soccer simulation and have been developing our agents called Kasuga-bito II. Kasuga-bito II was runner-up in the JapanOpen '98 [6] and was champion in RoboCup Pacific Rim Series '98 [7]. One of the features of our agents is that they learn better positions during the soccer simulation game by themselves, and they hold the positions onto this data.

In RoboCup soccer simulation, each agent has its own stamina[9]. The agent can dash with power lower than the stamina. The stamina is decreased by the power, and is increased with time. If an agent exhausts its own stamina, it can not play enough. Especially when a forward agent exhausts its stamina, the scoring ability of the team drastically decreases. In that case, our agents can send its own objects to another agent who has more stamina and delegate its behavior.

Once the objects in an agent which exhausts its stamina are sent and the agent delegates to another agent, the delegated agent can play like the original one. We implemented the behavior delegation [10] by class loader and object serialization in Java.

2.2 Class Loader in Java

In Java, we can customize a class loader to define policies for loading Java classes into the runtime environment. An abstract class, ClassLoader, is prepared for it[8]. We can send a class in the form of byte codes and load the class into a remote environment.

A new class loader can be created by defining a subclass of ClassLoader and implementing the abstract method loadClass. The brief procedures of the method are:

1. reading the byte codes of the class,

```
int byteCount;
FileInputStream fis;
....
byte[] data = new byte[byteCount];
fis.read(data);
```

2. creating the class object,

```
Class c=defineClass(data,0,data.length);
```

3. and resolving the class.

```
resolveClass(c);
```

The ClassLoader is very convenient to load class byte codes into a remote environment. It is, however, not so owerful in implementing the behavior delegation of our soccer agents. In Java, a class to be loaded by a ClassLoader and another class to be loaded by another class are regarded as different classes even if both of the names are the same. It means that we can not replace an existing class to a new class in runtime. To implement the behavior delegation, we need hacked codes. The work is error-prone and requires a great deal of effort.

Consequently, a new programming language Escar was designed. It is syntactically the extension of the Java programming language. The unique feature is the dynamic reconfiguration. Section 3 explains it.

2.3 Object Serialization in Java

The object serialization is included in standard Java classes[11]. It supports the encoding of objects (and the objects reachable from them) into a byte stream (ObjectOutputStream). The encoding is called serialization. Moreover, it supports the reconstruction of the objects from a byte stream (ObjectInputStream). The reconstruction is called deserialization. The writeObject method in the class ObjectOutputStream is responsible for writing the state of the object. Furthermore, the corresponding readObject method in the class ObjectInputStream can restore it.

The object serialization in Java is helpful in implementing mobile objects. However, it has two drawbacks:

1. Serialization and deserialization are low level for programmers.

   A program must read objects in the same order as the program that writes objects. It means that programmers make their codes like primitive sequential file access operations. This is tedious and error-prone work.

2. All kinds of objects are not serializable.

   Default methods, writeObject and readObject, do not serialize static objects. Moreover, the variables in the class without serializable interface are not serialized. We can customize those two default methods by overriding them. However, programmers have to write codes for all variables they want to serialize.

Consequently, a new tool set Yare for object serialization was designed. The section 4 briefly explains it.

3 Escar: A New Programming Language

A new programming language, called Escar, was designed to implement the behavior delegation. The main idea is to use "time" as a primary key for search. Time is one of the absolute measures around us and is useful to search for what we want. Next sections describe the storage model of Escar and the execution model.

3.1 The Storage Model

We have to manage a lot of information in the daily office work. In the typical method, we usually classify the information by some category and keep it in boxes or file cabinets. However, the classification categories tend to change as time passes. Moreover, after classification, the stored information is sometimes useless.

To overcome the problem, an exciting method is proposed[12]. It emphasizes the use of "time" as a primary key for search. It says that it is useless to classify information because it takes trouble and time to do it. If we can use "time" as the primary key, we do not have to classify information. The past time does not change so that the classification problem does not happen at all.

According to our experience of applying it to real office work, it is very useful to search for information by time order.

Escar is a programming language that applies the above method. Everything (cf. objects, classes, methods, data, etc.) is stored and wrapped in an "envelope". The envelope is a basic storage unit and it can have a name to identify it. An envelope with no name is also available. It can have some attributes to characterize the envelope

For management of the envelope, all we have to do is push the envelope into a "shelf" like stack data structure(Fig.1). The shelf manages the envelopes by time order. When the envelope is pushed onto the shelf, the envelope enters from the leftmost edge.

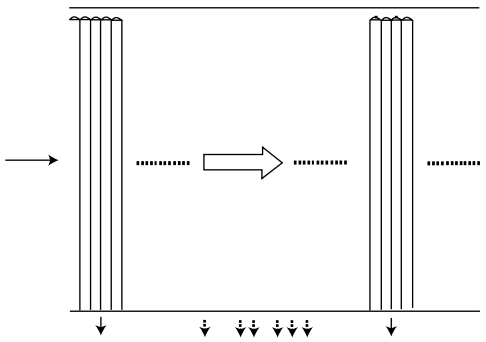

Fig. 1: Pushing and extracting the envelope

When we modify or refer to an envelope, we look for it by time order and we can extract it from anywhere. After the modification or the reference, the envelope reenters from the leftmost edge of the shelf.

The merit for time ordered management is firstly that we can access the newer envelope faster than the older one. This is because the recently accessed envelopes gather on the left edge of the shelf. Secondly, we can easily find the useless envelopes. The reason for this is that the envelopes which have not been recently accessed gather around the right edge of the shelf.

3.2 The Execution Model

In Escar, we can execute the contents of the envelope by message passing and delegation mechanisms[13]. When we want to execute some program codes, we pass the message to the corresponding envelope. If the contents of the envelope are matched to the message, the contents are invoked. If the contents are not matched to it, the message is delegated to the next envelope by time order.

In message passing, we can specify two kinds of message receivers. One is the newest envelope and the other is the envelope identified by the envelope name. If we specify the newest envelope and its contents are matched to the message, the message will not go to the older envelope. If we specify the envelope identified by its name, we can invoke the envelope without relation to time order.

4 Yare: A Tool Set for Object Serialization

A new tool set, called Yare, was designed for object serialization. The objects which the users want to serialize, are put into a kind of container. It is called abstract-data.

The abstract-data realizes representation independence. Let us refer to Fig. 2. A cloud in the center of the figure depicts the abstract-data. It supports abstraction to the internal representation. For access of the objects, the users are forced to invoke the methods, setter/getter, which are generated by a Yare tool. The setter method sets an object to a variable, and the getter method gets an object from a variable. As a result, they do not have to know about the concrete internal representation of the abstract-data.

When users want to serialize or deserialize the objects, all they have to do is invoke the serialization method or the deserialization method. Yare supports more than one data format for serialization, for example, XML[14] and Java serial-

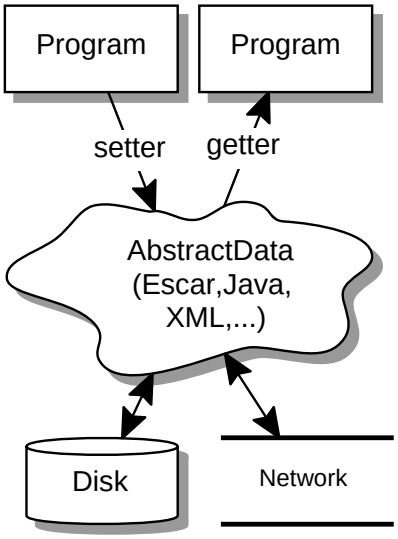

Fig. 2: Basic Design of Yare

ized data format. When users invoke the serialization method, the data format is specified. This implies that the concrete serialized data format is hidden from the programs.

The abstract-data realizes language independence. We can basically use the primitive data type, that is, Integer, Real, String and Boolean. In addition to that, we can use abstract-data as data type, for example, ExchangedData in Fig. 3. The Yare tool generates the language specific codes from the definition of the abstract-data.

In addition to the abstract-data, abstract-interface is defined in the Yare language (described in Fig. 3). The abstract-data has language independent definitions. On the other hand, the abstract-interface includes language specific definitions. As previously stated, the abstract-data is a collection of data . The Yare users put the data which they want to serialize into the abstract-data. The abstract-interface specifies:

- a target programming language,
- language specific definitions, and
- the name of serialization methods and deserialization methods.

```
AbstractData ExchangedData Is
  String s;
  Date   d;
End
AbstractInterface ExchangedData Is
  Lang Java {
    Import java.util.*;
  };
  In   LoadObject Use XML;
  Out  SaveObject Use XML;
End
```

Fig. 3: An example of abstract-data and abstract-interface

The Yare tool set generates some codes of the specified target language by "Lang" statements. Language dependent definitions must be described in it. As an example, Java language is specified in Fig. 3. Moreover, "import java.util.*" is described so that users can use "Date" data type.

In the abstract-interface, we can specify the name of the serialization method and the deserialization method by the keywords "In" and "Out" respectively. In Figure 3, "LoadObject" is the serialization method, and "SaveObject" is the deserialization method.

5 Conclusion

This paper described Escar programming language and Yare tool set. They are designed to build mobile objects.

The main idea of Escar is to use time as a primary key for search. As natural result, a program written in Escar is dynamically reconfigurable. Yare supports new object serialization. The features of Yare are representation independence and language independence.

We are using Escar and Yare to develop RoboCup soccer agents. The agent can send the

objects to another agent via inter-agent communication and it can delegate the behavior. We believe that Escar and Yare are indispensable tools used to implement it. We will try another application to demonstrate the capabilities.

Acknowledgements

This work has been partially supported by the Hori Information Science Promotion Foundation.

References:

[1] Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, 1998.

[2] Alan H. Bond and Les Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, 1988.

[3] The RoboCup Federation. *RoboCup: The Robot World Cup Initiative*. http://www.robocup.org/, 1998.

[4] Minoru Asada, editor. *The Second RoboCup Workshop*. Springer-Verlag, 1998.

[5] Hiroaki Kitano, Minoru Asada, et al. The RoboCup: The Robot World Cup Initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife*, pages 19–24, 1995.

[6] JapanOpen98 and JSME Robotics-Mechatronics Symposia98. http://www.robocup.org/games/321.html, 1998.

[7] Kazuaki Maeda, Akinori Kohketsu, and Tomoichi Takahashi. Goal-Keeping Skills in Soccer Simulation Games. In *PRICAI'98 RoboCup Workshop*, pages 96 – 101, 1998.

[8] Patrick Chan and Rosanna Lee, editors. *The Java Class Libraries: An Annotated Reffernce*. Addison-Wesley, 1997.

[9] David Andre, Emiel Corten, et al. *Soccerserver Manual*. http://www.dsv.su.se/~johank/, 1999.

[10] Kazuaki Maeda and Akinori Kohketsu. A Consideration of Mobile Objects in Soccer Simulation Games (in Japanese, to appear). In *JSAI Technical Report*, 1999.

[11] Sun Microsystems. *JDK 1.2 Documentation*. http://java.sun.com/products/jdk/1.2/docs/, 1999.

[12] Yukio Noguchi. *Super Method for Information Management (in Japanese)*. Chuo Shinsho, 1993.

[13] David Ungar and Rondall B. Smith. Self: The Power of Simplicity. In *Proceedings of the 1987 SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 227–242, 1987.

[14] World Wide Web Consortium. *Extensible Markup Language(XML)*. http://www.w3.org/XML/, 1999.