JMT (Java-based Moderator Templates) for Multi-Agent Planning

OOPSLA'97 Workshop
Java-based Paradigms for Agent Facilities

Kazuhiro Minami and Toshihiro Suzuki
IBM Japan, Yamato Laboratory

1 Introduction

Mobile agents[1] are programs that can be dispatched from one computer and transported to a remote computer for execution on behalf of a user. In recent years, a number of mobile agent frameworks[2],[3],[4],[5],[6] have been implemented in the Java language because of its platform independence. But in case of Java, there is one crucial problem that a mobile agent cannot hold the process state while moving across the network. That is, it is difficult to build a mobile agent that can perform successive jobs in series or parallel at different places.

Thus we adopted the concept of the plan that is common in the field of workflow application into mobile agents. A plan defines the sequence of activities to be performed. Although we referred to standards of the workflow application proposed by WfMC[7] or DARPA/Rome[8], we can't use existing workflow engines since a plan moves along with a mobile agent across the network. Because of this, a set of templates called **JMT(Java-based Moderator Templates)** has been developed. JMT templates define the basic collaborative behaviors among mobile agents and allow the creation of a complex plan by simply combining them so that multiple agents can work together for a common goal.

2 Overview of JMT

JMT framework consists of a set of Java class libraries on top of a Java-based mobile agent framework - currently implemented based on Aglets[2]. Figure 1 illustrates how multiple mobile agents work together in collaboration with major components in JMT. A *planner agent* creates a **plan** that defines the sequence of **activities** to be performed using **moderator templates**. After that, a *moderator agent* gets the plan from the *planner agent* and execute it step by step while moving across the network. In other words, a moderator agent executes a given activity in each visiting place. A *moderator agent* can split to two or more agents to perform activities in parallel and converge into a single one later to share the results under the control of moderator templates.

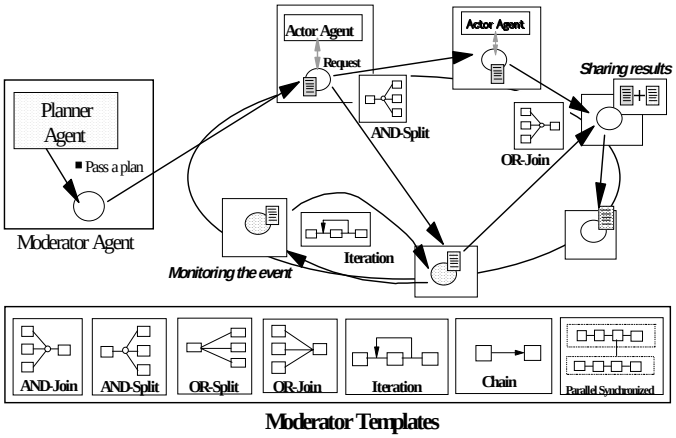

Figure 1 Overview of JMT

3 Object Model of JMT Framework

Figure 2 shows an object diagram of JMT and the relating classes. ModeratorAgent class inherits the mobility from *Aglet* class and hold a Plan object to be executed. Plan maintains a set of PlanNode objects sequentially and has responsibility for keeping track of the current element. That is, a plan retains the process state of the successive jobs instead of a mobile agent.

PlanNode class is an abstract class that defines the interface common in both ModeratorTemplate and Activity class so that they can be treated identically in Plan object. ModeratorTemplate class provides the basic mechanism that enables multiple moderator agents to collaborate with each other. On the other hand, Activity class contains a piece of work within a plan as well as the address of host where it should be performed.

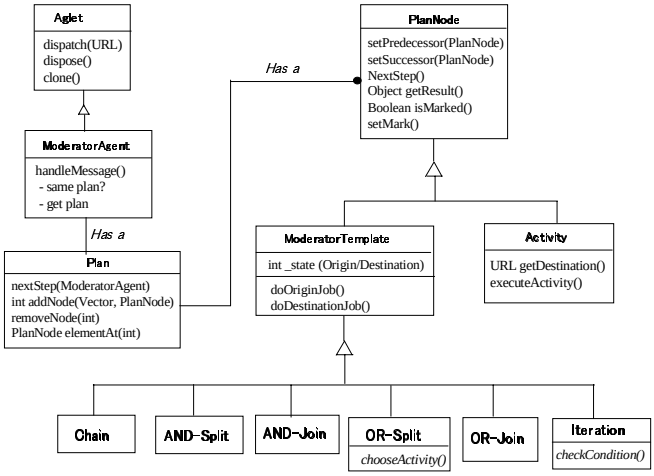

Figure 2 Object Diagram of JMT Framework

4 Structure of Plan

The structure of a Plan object can be represented as a decomposition graph as shown in Figure 3. There are two kinds of nodes in the graph. The first is Activity class enclosed in a rectangle and the second is ModeratorTemplate class that is indicated as a circle. Plan maintains the ordered list that stores the references to each PlanNode. PlanIterator class is responsibility for keeping track of the current node in the list. Each PlanNode has bidirectional link with adjoining ones. Each Activity object has to be connected with each other via a ModeratorTemplate object.

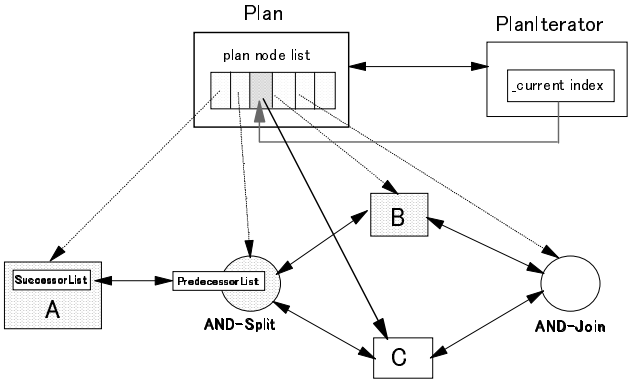

Figure 3 The structure of Plan object

5 Conceptual Message Flow

Figure 4 shows the conceptual message flow of JMT Framework. ModeratorAgent moves across the network with a plan object described in Section 4. Every time ModeratorAgent arrives at a new place, it calls the *nextStep()* method of Plan class. Then the message is forwarded to the current node such as ModeratorTemplate object or Activity object. If the current one is ModeratorTemplate object, it calls back the *clone()* method of ModeratorAgent to make a copy of agent, or the *dispatch()* method to move to the other place, or the *dispose()* method to delete itself. These three are basic methods for mobile agents inherited from Aglet class. In case of Activity object, *executeActivity()* method is invoked inside the *nextStep()* method to execute a given activity. The point is that the execution code to perform a set of jobs is completely separated from a mobile agent and encapsulated in Plan object.

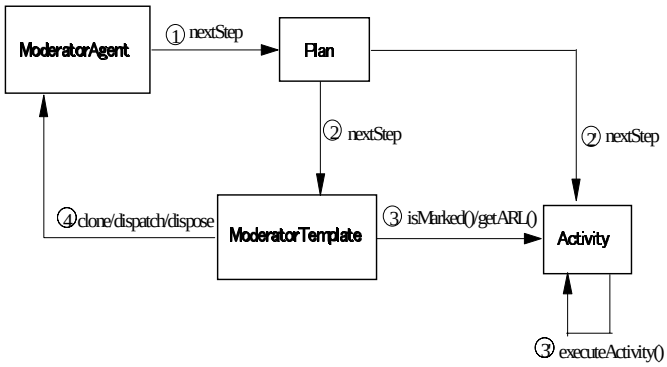

Figure 4 Conceptual Message Flow

6 Collaboration patterns among mobile agents

Currently, six concrete classes of ModeratorTemplate are implemented as shown in Figure 2 to be enough to conform to the workflow reference models[7],[8]. A few of major template are described to describe the collaboration patterns among mobile agents in the following.

6-1 Parallel Execution (AND-Split)

ANDSplit has multiple successor activities to be executed in parallel in different places. It conceptually represents the flow where a single agent splits into two or more agents that has different destinations.

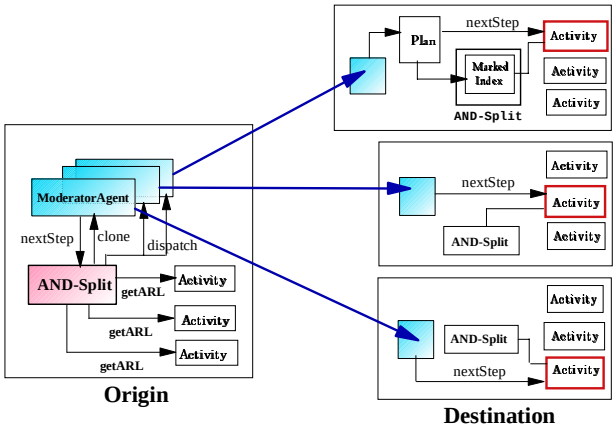

Figure 5 AND-Split

In the origin, ANDSplit creates the clones of a moderator agent according to the number of activities it holds. Next, ANDSplit dispatches each moderator agent to its destination place. In the destination place, one of successor activities is chosen as the current node in the plan and the activity chosen will be executed as the next current node.

6-2 Convergence (AND-Join)

ANDJoin, which is usually used with ANDSplit, has multiple predecessor activities and a single successor one respectively. In the origin, each moderator agent performs one of predecessor activities and moves to the same destination place.

In the destination, one agent is decided to be be responsible for merging with other agents. That agent gets plans from others and put s the results of activities to be done by others into its own plan object. Then other agents are disposed and a remaining one continues to execute the plan.

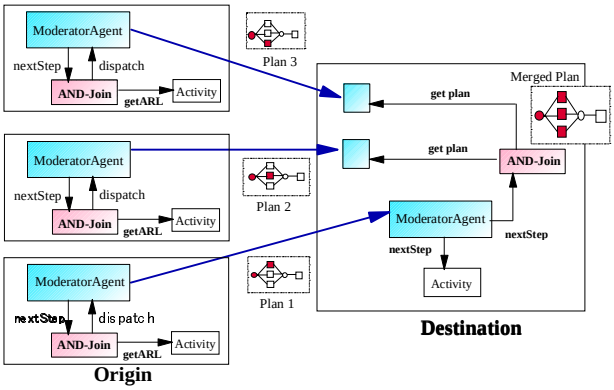

Figure 6 AND-Join

7 Conclusion

JMT enables developers to build a mobile agent application without doing any programming with the APIs of mobile agent framework. Instead, a mobile agent can be build just by creating a suite of concrete activities and combining them into a plan. Thus JMT is perfectly capable of being the basis of building-block environment for building mobile agent systems, incorporating the paradigm of componentware.

Future work will include providing a mechanism for a higher level of reuse, so that a plan itself can be treated as a subcomponent of a more complex plan.

References

[1] J. E. White, "Telescript Technology: Mobile Agents", General Magic White Paper, 1996.[2] The Aglets Library - Programming Mobile Internet Agents in Java, IBM Tokyo Research Lab, URL=http://java.trl.ibm.com

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

[4] W. Li, Java-To-Go, Univ. of California, Berkeley, URL= http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/

[5] Concordia , Mitsubishi Electric Information Technology Center America URL=http://www.meitca.com/HSL/Projects/Concordia/Welcome.html

[6]ObjectSpace - Voyager URL=http://www.objectspace.com/Voyager/voyager.html

[7] Workflow Management Coalition, URL=http://www.aiai.ed.ac.uk/project/wfmc/

[8] ARPI Plan Ontology Construction Group (POCG) URL=http://www.ai.rl.af.mil/PI/Ontology.html