Modeling and Validation of Mobile Agents on the Web

Min-Jung YOO	Walter MERLAT	Jean-Pierre BRIOT
yoo@poleia.lip6.fr	merlat@onera.fr	briot@poleia.lip6.fr
LIP6	ONERA/LIP6	LIP6

LIP6 : Laboratoire d'Informatique de Paris 6, UPMC.
Tour 46/0, 2ème étage, Case 169,
4 place Jussieu, 75252 Paris Cedex 05, FRANCE,
Tel. +33.1.44.27.73.89, Fax. +33.1.44.27.70.00

ONERA : Office National d'Etudes et de Recherches Aérospatiales.
DES/SIA, BP 32, 92232 Chatillon, FRANCE,
Tel +33.1.46.73.41.41

Key Words *: agent, cooperating protocol, Java, reusable component, mobile agent platform, electronic commerce, Web, simulation, colored Petri net*

This work is partially supported by France Telecom CNET through CTI7-CARISMA project.

Abstract

This paper presents compositional modeling and simulation techniques applied to the design of electronic commerce systems on the Web. We discuss an agent-based approach for designing open distributed systems, and propose a compositional modeling technique for agents behaviors using *reusable components*. We apply this technique to the design of a Travel Agency Service using the Contract Net Protocol [Smith 80]. Domain specific components are programmed in Java and cooperating components for the Contract Net protocol are expressed in a variant of Petri nets. Specification of components is automatically translated into a *mobile agent platform* (*Java* language environment) so as to execute the agents. Validation then takes place through simulation and various kinds of analysis of the Petri nets.

1. Open Cooperative Systems

The Internet, on-line networks, and groupwares promoted the development of Open and Distributed Cooperative Systems (ODCS), which are composed of a collection of application softwares (personal organizers, DBMS, ...) that cooperate together so as to satisfy customers' needs.

1.1 The Travel Agency

Let's consider our test application - the Travel Agency Service System. It provides customers with *full featured* travel service reservations, including hotel stays and car rentals. A Travel Agency application (TA) acts as a facilitator for service providers composing the system : travel servers (e.g., Air France, TWA, etc.), rent-a-car servers (e.g. Hertz), « hotels» servers (Novotel, Hilton, etc.), etc. Customers issue requests to the TA, concerning reservations for specific travels. The TA plans each travel service by contracting out a set of «base services» (train or plane trips, hotel stays, car rentals) to service providers.

For example, let's assume a request for a travel from Bordeaux (France) to New York (USA). The TA decomposes by ticketing for rail-road (from Bordeaux to Paris) and flight (from Paris to NY), and hotel reservation (in Paris and in NY), etc. The rail-road service may be contracted out to the French SNCF, the flight service to AirFrance, and the hotel reservations to Novotel in Paris and to Hilton in NY. These services providers have to cooperate so as to (1) find compatible schedules for services (i.e. at Paris stop), and (2) adapt these schedules in case of a modification occurring to one service (e.g. a flight is delayed, or a train is canceled).

Our purpose is to design ODCS that are fault-tolerant, scaleable and work in heterogeneous environment. In section 1.2, we analyze this problem and propose our solutions.

1.2 Issues in Open Distributed Cooperative Systems (ODCS) Design

Heterogeneity

Software applications are initially developed independently, not for cooperation purpose. They run on various OS/hardware platforms and network technologies. Agentification process, as described in [Genesereth and Ketchpel 94], hides heterogeneity by wrapping software into agents. Autonomous software agents, and mobile agents provide useful concepts for OCDS design.

Dynamics

Dynamics in ODCS environment can be considered in two aspects.

1) Openness and extensibility. Software application, and the services they provide, are regularly upgraded to adapt to changes in users needs. Also, new services may be introduced to satisfy to new kinds of needs. We investigate component based approaches [Nierstrasz and Meijler 94] to design agents and manage their evolution more easily.
2) Complex cooperation protocols. As illustrated in the Travel Agency example, sophisticated cooperation protocols are needed to perform cooperation processes involving numerous agents which provide various services : travel, hotel, car rental services, etc. In particular, protocols have to handle exceptions (e.g. changes in schedules) so that agents can adapt to changes. For this purpose, we develop agent-based cooperation protocols [FIPA 97]

The paper is organized as follows. In section 2 and 3, we present compositional agent modeling with different types of components. In section 4 we explain our description language which allows to describe various kinds of components illustrated in section 3. Some characteristics of our mobile agent platform is given in section 5, and section 6 is devoted to illustrate the technique to generate executable code from the model specification, which permits to simulate the agent behavior over the platform. The verification issues about cooperating protocols is presented in section 7. Finally, section 8 is a brief conclusion.

2. Compositional Modeling

We use components for incremental design and reuse. Agents are made of a collection of components, and can easily be upgraded by changing components or inserting new ones.

In the following subsection, we describe the general characteristics of the software components we propose, and then explain their use for modeling the agent behavior.

2.1 General Characteristics of Software Components

Our notion of component is in the spirit of [Nierstrasz and Meijler 94]. A software component can be defined by a ‘behavior’ and its ‘connections’. The connections represents its interactive feature and the behavior represents its internal procedure which might be activated by the interaction with other components.

Connections between two or more components are realized by establishing its communication lines. There are two types of communications possible: (1) asynchronous message passing, and (2) synchronous message passing, e.g., service request/response.

The component’s type can be characterized by way in which its behavior is described. We propose four types of software components : Petri net component, Primary component, Generic connector component, and Compound component. Compound components are constructed by composition of other components, i.e., Petri net component or primitive component or connector component or compound component itself. In the following subsections, we explain each type of component we propose.

2.2 Petri Net Components (Bric)

They are <Bric (Block-like Representation for Interacting Component) [Ferber 96]> components. The Bric formalism is derived from the colored Petri nets for the facility of modeling concurrent processes in a modular way. We use Petri net components (Bric) for the design of conversation protocols for two reasons.

1) The state-transition mechanism of the Petri net is suitable to describe the conversation protocol of an agent. Indeed, agent conversation protocols are often defined through a state-transition mechanism called conversation rules ([Barbuceanu and Fox 95]). These rules specify how an agent in a given state receives a message of a specified type, does local actions, sends out messages, and switches to another state.
2) It enables to verify the composition of conversation protocols. To avoid making errors in composing them, we use the validation and verification algorithms applied to Petri nets [Jensen 92].

More examples about designing agent protocols using Bric components (Petri net component) have been explored in [Ferber 96].

2.3 Primitive Components

Their computational behaviors are programmed in Java; this means that if there is any Java application or Java applet, it can be encapsulated as a connectable component without modifying the source code. Primitive components are useful to design agent specific tasks for representing arbitrary agent behavior.

2.4 Generic Connector Components

We provide a class of specific connecting components. They are useful to establish one-to-many connections between arbitrary number of agents. For the time being, there are two types of connector components: Synchronizer and Selector. Examples of use will be shown in section 3.2.

3. Modeling the Travel Agency Example

Using various components, we designed 'Travel Agency' and 'Service Provider' agents which cooperate through the Contract Net protocol [Smith 80].

3.1 Overall Scenario

Three machine sites are considered, one for the Travel Agency agent, and one for each of the service providers (TWA, Air France). The simplified agent model is shown in figure 1.
In this scenario, the Travel Agency agent starts off the Contract Net protocol(1) to collect informations(2) about related services (in this case, available seats with cost), and selects(3) one service provider in comparing proposed cost, then it moves(4) to the site where the selected service provider resides. For modeling the agent conversation protocol, we adopted 'Agent Communication Language' standard proposed by [FIPA 97], that is, 'cfp', 'bid', 'accept-offer', and so on.

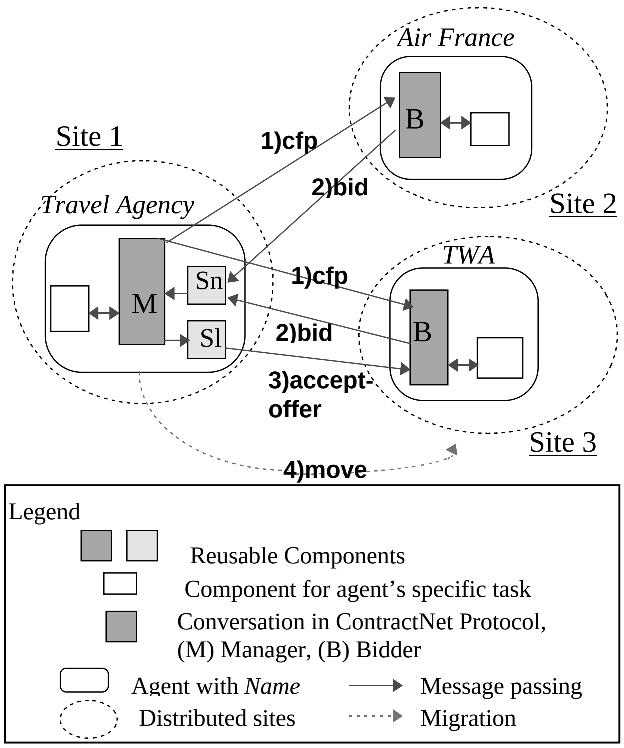

Figure1. <Agent Model for Travel Agency Service>

3.2 Compositional Design for the Agents

Among the model shown in figure 1, the components for the Contract Net protocol (component (M) and (B)) are modeled as Petri net components.

Primitive components, which are represented with white boxes in figure1, are used to encapsulate service softwares (e.g., the mechanism for selecting a service in the behavior of the Travel Agency agent, or maintaining database accesses in the behavior of the Service Providers).

Connector components are represented with soft-gray boxes. The synchronizer (Sn) component waits until all connected agents send an expected message (bid). If all messages are arrived, it passes the set of arguments included in the messages to the connected component (M). The selector (Sl) component functions as a dispatcher between one agent and two or more agents. It selects one agent among the Service Providers, and sends a message (accept-offer) from the Travel Agency (M) to the selected agent (TWA), and then moves to its site.

All different types of components are composed into one compound component to construct an agent's behavior, including its specific task and its conversation protocol. In this example, an agent structure embodies one component which stands for its behavioral aspects. Thus, in figure1, one compound component is viewed as one agent.

4. Soft-Component Description Language

The aim of the Soft-Component Description (SCD) language is to provide a formal expression language to describe the components. This language enables the agent designer to model an agent behavior by defining component classes and then composing component instances.

For example, by referring to figure1, the Travel Agency agent is designed with one compound component ('Travel_Agency') described in table1. This component is composed of four sub-components. Among these, Sn and Sl are the system-provided components, and the task-specific component (Manager_Task_Compo) is given in table2. The Petri net component for the manager role of the Contract Net protocol (Contr_Manager) is given in table3.

The instanciations of these sub-components and establishing its connections will be automatically created at the instanciation time of the 'Travel Agency' component

Table 1 : Travel Agency Component

```
// ——Example of a Compound Component ——

DEFCOMPO Travel_Agency {
  SUPER SoftComponent
  VAR {Condition condt, Vector bidList, Award award}
  MSGIN {beginContract(condt), bid(bidList}
          //Receives two asynchronous messages,
  MSGOUT {announce(condt), accept-offer (award)}
          // Sends two asynchronous messages,
  SUBCOMPONENT {
          // Four sub-components are instanciated by
          // 'instance_name'  ISA  'component_class name'
          task_compo ISA Manager_Task_Compo {...}
          manager_role ISA Contr_Manager {...}
          waitBids ISA Synchronizer <...>
          sendAward ISA Selector <...>   }
  CONNECTION {
          //connects sub-components to each other
          // composing component instances
          <....>  }}
```

Table2: Manager_Task_Compo component

```
// ——Example of a Primary Component ——

DEFCOMPO Manager_Task_Compo {
          < ..... >
  REPLY {selectService(bidList) WITHTYPE Award}
          //This component is connected by
          // synchronous message type
  ENCAPSULATE  Manager_Task AS myWork
          // It encapsulates a Java object
          // 'Manager_Task.class '
  BIND { selectService TO selectService }}
   // The connection port 'selectService'
   // of this component binds to the method 'selectService'
   // of the encapsulated object.
```

Table 3 : Contr_Manager component

```
// ——Example of a Petri net Component ——

DEFCOMPO    Contr_Manager {
          < .....>
REQUEST {selectService(bidList) WITHTYPE Award}
//this component invokes a synchronous communication
STATE {announced, offered}
          // Informations about colored Petri net places
OPERATION {
          //informations about Petri net transitions
    < .....>
     action2 {INPUT {bid (bidList)}
              IF ("bidList.size() >0")
              THEN ASK selectService FOR award
```

```
SEND {accept-offer (award)}
     NEXTSTATE {offered ()}   END}
action3 {  ONSTATE {offered()}
     INPUT {move(dest)}
     THEN MOVETO dest
          NEXTSTATE {migrated()}   END}
     // migrates to the site indicated with dest}}
```

5. Mobile Agent Platform

JavaNetAgents (JNA) is a mobile agent platform, developed in Java at ONERA and LIP6 for ODCS design. Like most such platforms, its architecture is based on places (sites) and agents [Telescript 95]. JNA agents are Java objects that can move from one place to another one, across the Internet. Local and distant agents (i.e. agents hosted on the same or distant places) can easily communicate. Security features prevents malicious agents from causing making unauthorized access to local resources (files, sockets, threads, etc.). Figure2 is a screen example of JNA on the site3, after the Travel Agency agent moves to that site.

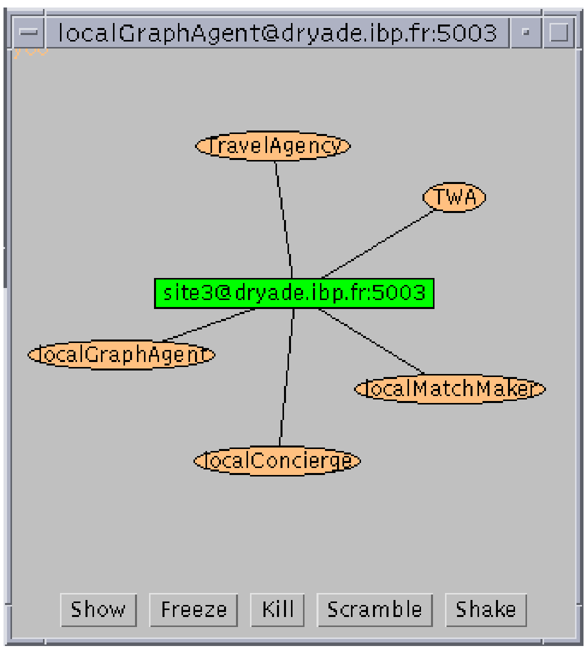

Figure2 <JNA Screen Example>

Agent's mobility presents good properties:

- better use of network resources. In particular, less bandwidth is needed, a better response times are obtained ([Oracle 95]).
- transactions in disconnected mode, and better fault-tolerance ([Telescript 95]).

For a complete discussion of the JNA platform and the benefits of mobile agents in open distributed cooperative system, see [Merlat et al. 97].

6. From Model to Implementation

The translation from compositional model to JNA agents is achieved automatically. We have implemented a compiler which produces Java source program from the model specified in the Soft-Component Description language. From one component class description, the compiler produces one Java class definition.

Take, as an example, the Travel Agency agent model in section 4. Implementing and simulating agent behavior begins by compiling the descriptions in the table 1 to 3 into Java source code. For example, the following list was produced by compiling the SCD in table 3.

```
<....>   // for Java 'import' clauses
public class Contr_Manager extends SoftAgent {
   Award award ;
   Place announced = new Place ("announced") ;
<....>
   public void bid (Vector bidList) {
      if (bidList.size() >0) {
            award = (Award) require ("selectService" ,
                                     loadArgument ( <...>) ;
            sendMessage ("accept-offer", <.....>); }}
   public void move (AgentAddress dest) {
<.....>   }}
```

Next, the resulting code is encapsulated into the JNA agent structure to execute the agent behavior. The component's behavior must be closely related to the physical structure of the JNA agent. In other words, the component's behavior which expresses the agent's behavior must be appropriately injected into the agent's initializing process(agentInit()), the agent awaking process after migration (agentAwake()), and the repeated execution cycle (agentRun()). This is very important while implementing mobility from Soft-Component Description, and we have achieved this in the definition of 'SoftAgent' class, a sub class of JNA 'Agent' class.

7. Validation

Whenever components are assembled to perform a common task, there is always an implicit contract between them in terms of interaction and collaboration [Nierstrasz and Meijler 94]. It is thus important to be able to validate components and their connections. To this end, we use Petri net components to verify cooperating conversation model. We are close to [VonMartial 92]'s approach, in that we use a general Petri net tools to simulate and analyze the Petri net components for finding deadlocks and related synchronization problems which may be found in composed conversation protocols.

For Petri net validation, we use the platform [CPN/AMI] developed by another research team within our laboratory. But, for the moment, CPN/AMI and our agent developing environment is not yet combined into a unified platform. We have composed our modular Petri nets to produce a global net using a Petri net editor in CPN/AMI, and we have executed validation algorithms on it.

8. Conclusion

In this paper we described our approach to model, simulate, validate and execute a multi-agent system for electronic commerce applications. We described a Travel Agency Service system as a test application and illustrated the issues in the design of such kind of open distributed cooperative systems.

In order to support the compositional design scheme, we developed the SCD language and its compiler to Java. SCD might be compared with the LOOPN language[Lakos et al. 95] for the Petri net components, and the OLAN component language [Bellisard et al. 95] for the primary component and the compound component description.

Our ultimate goal is to develop the full scenario of the Travel Agency Service system illustrated in section 1. As a starting step, we have developed a subset of the system, and this has been successfully tested between two distant local networks (LIP6 and ONERA LANs). We are currently porting this application to Aglets, IBM mobile agent platform [Aglets96]. The agents structure of Aglets and that of JNA are very much alike, therefore the port should be easy. We are also investigating more sophisticated applications.

Acknowledgments

We wish to thank Pr. J. Ferber at University of Montpellier France for his contribution to this project and kind guidance in the course of this research.

References

[Aglets 96] http://www.trl.ibm.co.jp/aglets/

[Bellisard et al. 95] L.Bellissard, S.Ben Atallah, A.Kerbrat, M.Riveil, ***Component-based Programming and Application Managgement with OLAN***, Workshop on Object-Based Parallel and Distributed Computation, June Tokyo, 1995, LNCS 1107 Springer-Verlag

[CPN/AMI] MARS-Team, ***The CPN-AMI Environment Version 2.0 Beta Release***, Laboratoire Lip6, Université Pierre & Marie Curie (Paris 6), 1996.

[Ferber 96] Jacques Ferber, ***Les Système Multi-Agents***, InterEditions, 1996.

[FIPA 97] Foundation for intelligent Physical Agents, Specification Part1 - Part5, 1997, http://drogo.cselt.stet.it/fipa/spec.

[Genesereth and Ketchpel 94] M.R.Genesereth, S.P.Ketchpel, ***Software Agents***, communications of the ACM, Vol.37, No. 7, July 1994.

[Jensen 92] Kurt Jensen, ***Colored Petri Net, Basic Concepts, Analysis methods and Practical Use***, volume 1, Springer-Verlag, 1992

[Lakos et al. 95] C.Lakos, J.Lamp, C.Keen, B.MARIOTT, ***Modeling Network Protocols with Object Petri Nets***, 16th International Conference on Application and Theory of Petri Nets, Workshop on protocols, Italy Torino, 1995.

[Merlat et al. 97] Walter Merlat, Claude Seyrat, Jacques Ferber, ***Mobile-Agents for Dynamic Organizations: The Conversational-Agent Paradigm***, 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW), Ronnevy Sweden, LNAI 1237, Springer-Verlag 1997.

[Nierstrasz and Meijler 94] Oscar Nierstrasz, Theo Dirts Meijler, ***Requirements for a Composition Language***, ECOOP 94 Workshop on Models and Languages for Coordination of Parallelism and Distribution, Italy, LNCS 924, Springer-Verlag, 1994.

[Oracle95] Oracle Company. ***Oracle Mobile Agents***. Technical Product Summary. Available on http://www.oracle.com.

[Smith 80] Reid G. Smith, ***The Contract Net Protocol: A High Level Negociation Protocol for Distributed Problem Solving***, IEEE Transactions on Computer (29) 1980.

[Telescript95] General Magic. ***Telescript: An Overview***. Available on http://www.genmagic.com.

[VonMartial 92] F. Von Martial , ***Coordinating plans of autonomous agents***, LNCS 610, Springer-Verlag, 1992.