(Workshop WetIce98)

Using Components for Modeling Intelligent and Collaborative Mobile Agents

Min-Jung YOO(1), Jean-Pierre BRIOT(1) and Jacques FERBER (2)

1. LIP6 : Laboratoire d'Informatique de Paris 6, UPMC. Tour 46/0, 2 me tage, Case 169, 4 place Jussieu, 75252 Paris Cedex 05, France. Tel. +33.1.44.27.73.89, Fax. +33.1.44.27.70.00
E-mail°: {Min.Yoo, Jean-Pierre.Briot}@lip6.fr

2. LIRMM°: Universit Montpellier II, 161 rue Ada, 34392 Montpellier Cedex 5, France
E-mail°: ferber@lirmm.fr

Keywords
software agent, mobile agent, collaboration, intelligence, componential approach, Java, Voyager

Abstract
Intelligence, collaboration and mobility are the main important features in designing agents for electronic commercial systems. But very few systems permit to implement agents which have all these characteristics. In this paper, we address the problems of designing *intelligent collaborative mobile agents*. We propose a componential approach for designing and implementing such agents. Our framework architecture for agents decouples components expressing the internal behavior of an agent from components expressing coordination between agents in order to enhance modularity and reusability. Our agent design and implementation environment consists in three distinct phases°: modeling, validation and implementation. We have developed a component description language for easy modeling of componential agent architecture. The component specification is automatically translated into Java source code by our compiler, and then integrated into a mobile agent platform. We have integrated our agent implementation environment into two different mobile agent platforms, namely JNA (JavaNet Agent) and Voyager. We have tested our agent development system with the Travel Agency scenario, one of the standard electronic commercial service examples.

1. INTRODUCTION
The recent rapid growth of information networks (e.g. Internet) enabled end-users to get more easily the various informations they want. This promoted the development of new electronic commercial products for the end users and the amount of such information providers is ever increasing. Such applications are not standalone but are expected to communicate with each other (even though they may have been initially developed independently). As described in [Genesereth 97], agentification process hides heterogeneity by wrapping different applications into agents. We call these inter-operable heterogeneous applications software agents which can communicate and work together in collaboration.

For example in our test example Travel Agency service system , the applications for the Travel Agency and Travel Services (e.g., Air France, United Airlines, hotels, rent-a-car, etc.) are connected through Internet. For the Travel Agency application to provide travel planning and reservation service, these service providers have to cooperate so as to: (1) find compatible schedules for services (i.e. selecting appropriate flight company, hotel for the user s schedule), and (2) adapt these schedules in case of a modification occurring to one service (e.g. a flight is delayed or canceled). We would like to develop each applications as appropriate software agents so as to construct collaborative electronic commercial services.

1.1 Characteristics of software agents

When we talk about agents in general, the commonly accepted nature in the literature [ATAL 96][Bradshow 97] can be categorized by three points of views°: *intelligence, collaboration*, and *mobility*.

1.1.1 Intelligence

The Agents ability concerning reasoning, planning, and learning capability can be regarded as various degrees of intelligence. From the view-points of the DAI (Distributed Artificial Intelligence) community [Demazeau M ller 91], there are two representative types of intelligent agents: *reactive* and *intentional* agents. A *reactive agent* reacts to changes in its environment or to messages from other agents. An *intentional agent* is able to reason on its intentions and beliefs, to create plans and actions, and to execute those plans.

1.1.2 Collaboration

One of the important issues in multiple software agents system is the agent collaboration which provides a more elaborated service through the agents cooperation. The agent cooperation can be achieved by two means: common *agent communication language* and *high-level agent conversation protocols*.

The purpose of common Agent Communication Language (ACL) is to provide inter-operability to different types of applications. The agents which are capable of understanding an ACL are distinguished from other kinds of software by their ability to communicate using a shared message semantic. The message semantics are typed and independent of the applications. The most well-known ACL standards are the KQML [Finin et al. 97] and the FIPA specification for standard ACL [FIPA 97].

However an agent can hardly achieve an expected goal only by sending a simple ACL. Sometimes the agents are expected to exchange continually appropriate ACLs until the goal satisfaction. In some cases, these conversations between agents follow some expected message sequences. This typical patterns of message exchange are called conversation protocol or agent cooperation protocol [Von Martial 93], [Burmeister et al.93] [Demazeau 95]. If agents in collaboration can communicate through such a pre-defined protocol, they can cooperate more efficiently to achieve more complex goals.

1.1.3 Mobility

Very recently, agent mobility has been introduced as a good solution for overcoming the limitations of networks - for example, communication bottle-necks, communication faults, etc. This is because mobile computation can reduce the communication band-width, and help a fault-tolerant execution against unpredictable communication failures. Therefore the agent s mobility is especially important in systems composed of multiple agents which are physically connected through networks (ex. Internet) and work together by exchanging messages at distance.

1.2 Issues in designing software agents with mobility

So we would all agree that the agent technology is a pertinent solution for developing collaborative software, and that the agent mobility is a good technical evolution which can be applied to efficient electronic commercial systems. And then what difficulties do we find? The main problems can be shown as follows°:

1. In mobile agent platform oriented researches, the relation between the agent mobility and the intelligent agent behavior has been hardly considered, nor the relation between the agent mobility with its collaboration and conversation protocols.
2. In agent design and implementation oriented researches, the agent mobility has not yet been effectively integrated with an intelligent behavior or cooperative multi-agents paradigm.

That is, there are few systems which permit us to develop both intelligent and collaborative mobile agent. Well-known mobile agent platforms, for example Aglet, Odyssey, Voyager, provide just a technical base concerning mobile computation. Constructing an intelligent agent or providing multiple agents collaboration is absolutely up to the designer s responsibility. Some other agent systems which addressed an intelligent agent implementation [Rao Georgeff 95] [Lalo 96] have not yet addressed agent mobility.

There are a few researches which have addressed jointly these issues; intelligence within a mobile agent or collaboration between mobile agents. We would like to explain them briefly in section 2, and then present our approach. Section 3 is devoted to our componential approach for agent modeling. In section 4, we will show some examples of designing agent conversation protocols using our component description language. Section 5 presents our compiling and implementing technique related with two mobile agent platforms. And then in section 6 we will conclude by summarizing our approach and future direction.

2. MOBILITY WITH INTELLIGENCE AND COLLABORATION

There are some approaches which try to incorporate intelligence and collaboration within a mobile agent architecture, namely [Plangent 97] and [Concordia 97]. Figure 1 summarizes these different approaches.

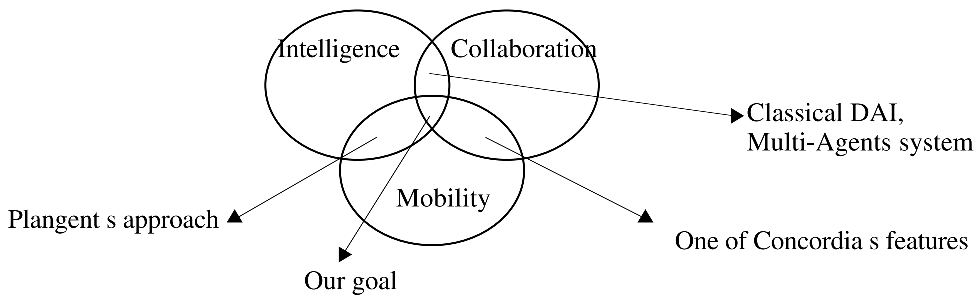


Figure 1 . Different levels of integration of intelligence, collaboration and mobility

2.1 Mobility with intelligence

In Plangent [Oshuga et al. 97], the agent s intelligence is represented as the ability to generate flexible plans. For that reason, the mobile agents must be able to: 1) plan actions that will satisfy the user requirements, 2) act according to the plan, and 3) modify the plan according to actual conditions. The agent s mobility is mainly used to gathering informations which cannot be obtained from the agent s site but possibly supplied by other site s domain knowledge. The agent mobility helps at an intelligent and flexible planning mechanism, by allowing the agent to generate a local plan and move to other sites for completing insufficient information.

We may remark that in Plangent, KIF (Knowledge Interchange Format) [Finin et al. 97] is used for the knowledge representation of each site. KIF is indeed one of the main proposals considered by the various committees involved in the Knowledge sharing effort [Finin et al. 97][Genesereth 97]. Nevertheless KIF is just an inner language of the ACL and agents need an ˙˚outer˚¨ language like KQML in order to communicate through ACL semantics. Plangent has successfully extended the originally conceived planning mechanism towards planning mobile agent , but in terms of inter-operable agents, the Plangent lacks cooperative aspects.

2.2 Mobility with collaboration

What distinguishes Concordia from other mobile agent platforms - like Telescript or Aglet - is the included collaboration framework mechanism. Concordia s collaboration framework provided by *AgentGroup* and *Collaborator agents* facilitates some kind of agent coordination to solve complex problems. Within an agent application, a complex task is divided into smaller pieces (sub-tasks) and delegated to agents that migrate to other sites to accomplish them. These agents perform distributed computations and synchronize at the Collaboration point ˚to share their results. Then the *AgentGroup* analyzes the global result after all *Collaborator agents* have arrived at their synchronization point.

But as we have mentioned, the system does not provide much facilities for the design of an intelligent agent with the capability of reasoning, planning, etc.

2.3 Our solution°: Componential architecture for integration of intelligence with collaboration

With Plangent and Concordia, or with any other mobile agent platforms, it is not easy to implement both intelligent and collaborative agent s behavior. Even though some kind of integration is partially solved, - for example intelligence with mobility in Plangent, or collaboration with mobility in Concordia - the integration of remained features is not so easy. In this context, what we want is to provide the model of an *intelligent collaborative agent* being integrated *with mobility*.

In order to achieve integration of these three perspectives, our approach is:

1. to decompose an agent into two distinctive components - a) an intelligent behavior component, and b) a collaborative component,
2. to encapsulate this componential agent into a structure of mobile agent.

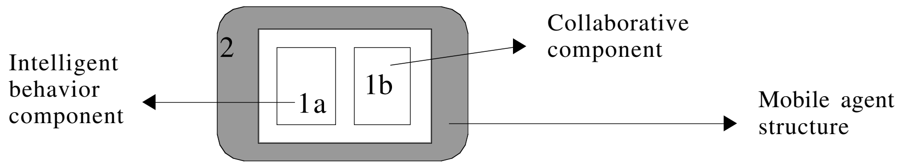


Figure 2. Simplified componential agent architecture

Figure 2 shows our agent architecture which contains the two important agent s features. But rather than mixing up them all within an agent s structure, we separate the different parts as a connectable component. More details about the componential agent architecture will be described in section 3.

2.4 An environment for developing collaborative intelligent mobile agents

We have developed a mobile agent modeling and implementation environment with which the designer models agents using different types of software components which can be easily connectable and replaceable.

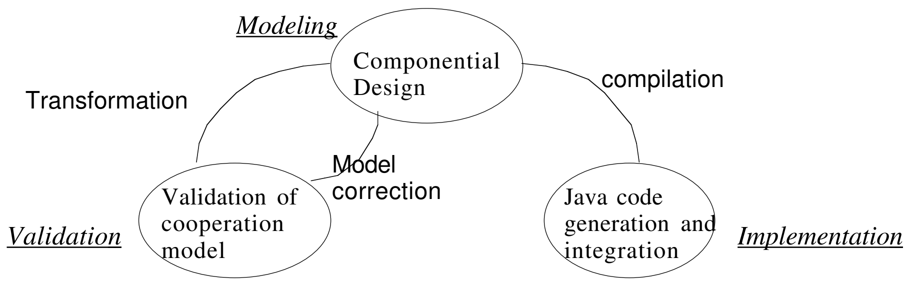


Figure 3. Agents system design and implementation environment

In this environment the agent modeling and implementation phase can be divided into three different phase:

1. In the *agent modeling phase*, the intelligent behavior and the collaborative aspect may be differently designed as distinctive components of the agent architecture concerning the mobility. This phase concerns only the abstract high-level agent model, that is to say, the designer does not need to directly manipulate the agent migration method which is specific to a mobile agent platform. We use a special description language which will be presented in section 3.2.
2. The *compilation into Java source code* and *implementation* phase makes it possible to generate automatically the operational Java code and implement the agent into a mobile agent

platform, which corresponds exactly to the model specified in the modeling phase. One benefit of our approach is to minimize the dependency of agent modeling and implementation environment from the mobile agent platform itself. This is shown by our experimentation with two different types of mobile agent platforms; one for JNA (JavaNet Agent) mobile agent platform [Merlat et al. 97] and the other with Voyager [Voyager 97].

3. The *validation of multi-agents collaboration model*. In our system we also address the validation of conversation protocols before executing the agents in real world. After designing an agent model, the validation phase verifies certain basic properties of the conversation protocol (no deadlock, no livelock, etc.) through the translation of the protocol specification into a Petri net formalism [Yoo et al. 98].

Now we would like to detail our componential approach for integrating agent mobility within collaborative intelligent agent architecture.

3. COMPONENTIAL APPROACH FOR AGENT MODELING

As we have mentioned in [Yoo et al. 98], the componential approach for the modeling and implementation of open systems - like software agents over Internet - gives flexibility during the modeling phase and for system maintenance. In the course of new model designing, the designer can combine the reusable parts which have been developed for other applications. One can also easily modify the old model by disconnecting and replacing the respective parts. So we have chosen such a componential approach for agent modeling.

3.1 Agent architecture constructed with components

The following figure shows an example of agent architecture. An agent can be viewed as a compound component composed of various sub-components. Each sub-component may be recursively decomposed into further sub-components.

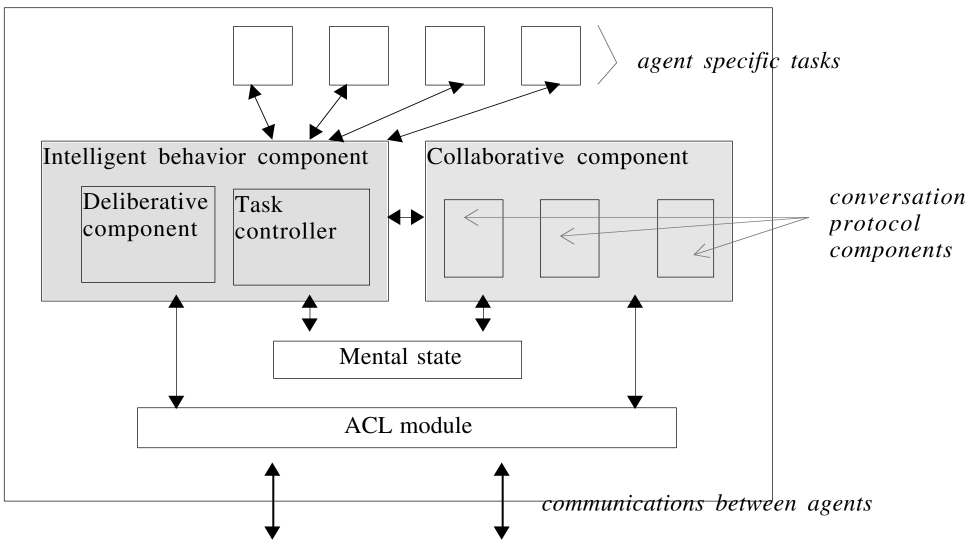


Figure 4. Componential agent architecture

The ACL module is responsible for agent communications using a standard ACL. The mental state component keeps track of the agent s knowledge about itself and its environment. The agent s specific task components represent the agent s computational behavior. And more, there are two distinct parts within the agent architecture: the intelligent behavior component and the collaborative component. In this paper, we will discuss more about the collaborative component, its sub-components for conversation protocols. The other components will not be further detailed.

3.2 SoftComponent Description (SCD)

We have developed the Soft-Component Description (SCD) language in order to describe different agent parts as connectable components. The agent designer uses this language to model an agent behavior by defining component classes and then composing component instances.
The essential characteristics of SCD are that:

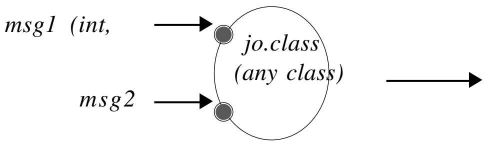
1. it is appropriate to describe the agent conversation protocols or the Task Controller components. SCD semantic is mainly based on the state/transition mechanisms. In case of Conversation protocol components, it is pertinent to represent the conversation states and transition conditions to the next conversation state.
   For example, a general framework of conversation protocol is:

   if an agent is in conversation state A and if it receives an agent message M(arg1, arg2)
   Then the agent is in conversation state B and it send a message N(arg3).

   This feature can be described using SCD syntax as follows:

   ```
   any_action_name {
                   INPUT    M(arg1, arg2)    ONSTATE (A)
                   THEN     {SEND N(arg3)    NEXTSTATE ( B) }
                   END}
   ```

   The same representation scheme for conversation protocol - i.e., conversation states and transitions - can be shown in [VonMartial 93] or COOL [Barbuceanu Fox 95]. In case of the Task controller component (in figure 4), SCD is capable of describing pre-conditions satisfied to start some tasks and post-actions (e.g., for re-initialization) which must be achieved after each tasks.
2. for modeling agent specific tasks the designer can encapsulate any Java programs into a connectable component using SCD syntax. This means that any Java application or Applet can be encapsulated and then composed within an agent architecture. For example, if we want to encapsulate a Java object as a connectable component;




   ```
   VAR    {INTEGER i, CHARACTER c}
   MSGIN    {compo_msg1 (i, c), compo_msg2()}
   ENCAPSULATE jo INTO my-component
   BIND   { compo_msg1  TO  msg1,
            compo_msg2  TO  msg2}
   ```

3. SCD is able to describe a compound component which can be obtained by the composition of different types of component, i.e., the type 1 or 2 of components listed above, or the compound component itself.

In this paper, we omit the full syntax of SCD but we will give a simple example in section 4.

4. EXAMPLE

With our componential approach we are developing the Travel Agency service scenario, one of the well-known examples of collaborative electronic commerce examples. We have chosen the message semantics of FIPA ACL [FIPA 97] for common agent communication language.

4.1 Agents configuration

In considering the Travel Agency scenario, we have selected three types of agents: User Assistant agent, Travel Agency agent, and different types of travel service agents (United Airlines , Air France , Hilton hotel, etc.). Figure 4 shows the agent configuration in that each agent is located on different sites.

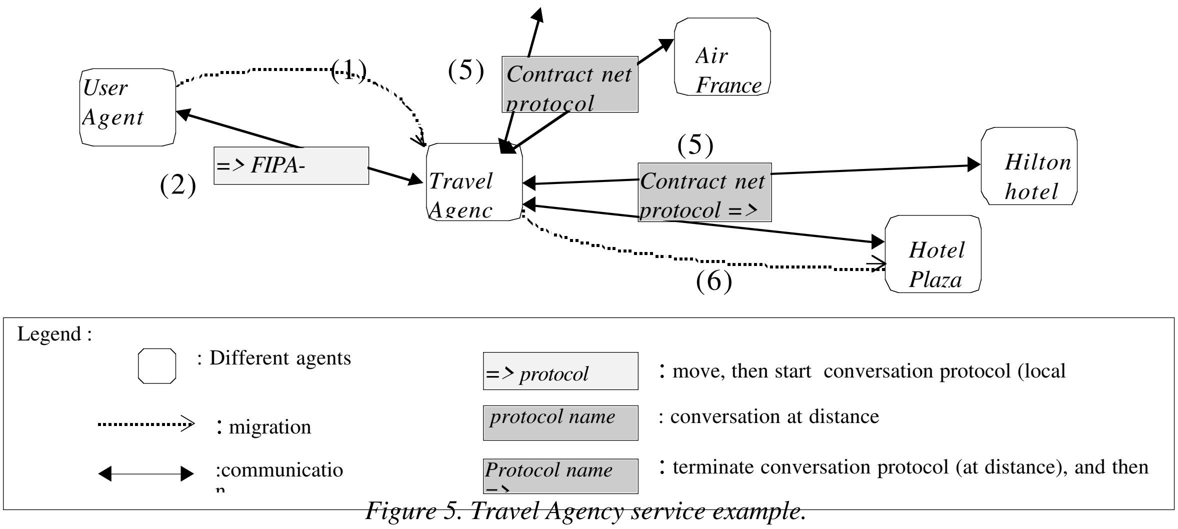


Figure 5. Travel Agency service example.

The User Agent asks the Travel Agency for a trip-planning service: a travel from Paris to San Francisco during 5 days. To start this conversation;

1. the User Agent first moves to the site of the Travel Agency agent.
2. Then the User agent starts the *FIPA-request* [FIPA 97] conversation protocol to request the Travel Agency s service *Trip-Planning* .
3. When the Travel Agency agent receives the request concerning the trip-planning service,
4. The Travel agency agent decomposes the trip summary (for example, *flight from Paris to San Francisco* and *hotel during 5 days*). It uses its acquaintance table to retrieve appropriate agent addresses which satisfy the decomposed service items (for example, addresses of service agents for *flight service* and *hotel*).
5. And then the Travel Agency agent starts the Contract Net protocols with flight service agents and hotel reservation agents, in order to find a ticket of minimum cost and a hotel room of minimum cost within the maximum satisfaction of user s need respectively.
6. After selecting an appropriate flight company and hotel, the travel agency agent migrates to the travel service agent s site to resume conversations. For example, for reserving a room the hotel agent might need more detailed informations about the customer s preferences. To respond more efficiently to these questions, the Travel Agency moves to the selected service agent s site.

The figure 5 shows the agent migration and conversation protocols used. In the following sub-section we would like to discuss only about conversation protocols (2, 5)

4.2 Designing conversation protocol components

The agent conversation protocol contains two types of participants: *invoker* and *invokee*. For example in the Contract net protocol [Smith 80], there are manager and bidder as participants. The manager is a protocol invoker and the bidder is an invokee of the Contract net protocol. This brings us to design two separated components for each participant. The example of Contract net protocol and FIPA-request protocols are shown in figure 6. The invoker and invokee parts are represented as dashed line boxes.

The conversation protocol components have been described using SoftComponent Description (SCD) language. For example the SCD specification of Contract net protocol components for manager and bidder parts are shown in figure 7a and 7b. For the moment the agent specific tasks are directly programmed in Java. But ultimately travel service agents may be developed with JDBC. The User Agent may be developed as a Java Applet and we compose these with our conversation protocol component.

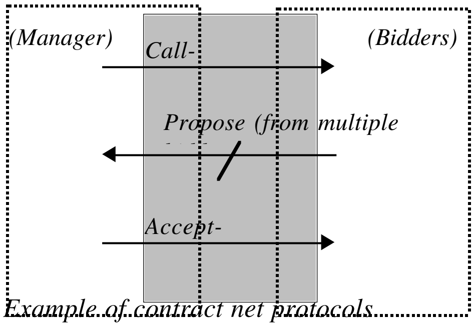


Figure 6a. Example of contract net protocols

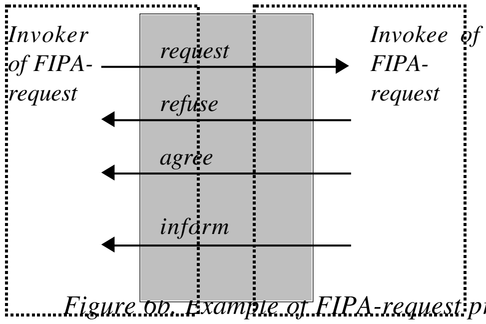


Figure 6b. Example of FIPA-request protocol

```
DEFCOMPO Contr_Manager { // Manager part of
                          // the Contract net protocol

SUPER SoftComponent
VAR {Condition condt, Vector bidList, Award award,
AgentAddress dest}
MSGIN {beginContract (condt), Propose (bidList),
migrate (dest)}
MSGOUT {Call-for-proposal (condt), Accept-
proposal (award)}
REQUEST {selectService (bidList) WITHTYPE Award}

STATE {offered, migrated}
OPERATION {
   action1 {  INPUT {beginContract (condt)}
              THEN SEND {Call-for-proposal (condt)}
              END}
   action2 {  INPUT {propose (bidList) }
              IF (bidList.size() > 0)
              THEN ASK selectService
              FOR award
              SEND {Accept-offer (award)}
              NEXTSTATE {offered ()}  END}
  action3 {  ONSTATE {offered ()}
              INPUT {migrate (dest)}
```

Figure 7a. Component description for Manager part of Contract net protocol

```
DEFCOMPO Contr_Bidder { // Bidder part of the
                         // Contract net protocol

SUPER SoftComponent
VAR {Condition condt, Bid bid, Award award}

MSGIN {Call-for-proposal (condt), Accept-proposal
(award)}
MSGOUT {propose (bid), beginService (award)}

STATE {BidCondition {TYPE condition}}
OPERATION {
   action1 { INPUT {Call-for-proposal (condt)}
             THEN  NEXTSTATE {BidCondition (condt)}
             END}
   action2 { ONSTATE {BidCondition (condt) }
             IFEXT hasService
             THEN  ASK makeBid FOR bid
             SEND {Propose (bid)}   END}
   action3 { INPUT {inform (Accept-proposal)}
             THEN  SEND {beginService (award)}  END}
}}
```

Figure 7b. Component description for Bidder part of Contract net Protocol.

4.3 Specification of mobility within conversation protocols

Within the description of action3 in Contr_Manager component (figure 7a), there is a moveTo field (SCD syntax) which indicates that the agent migrates to the site indicated by dest variable. We have modeled a Contract net protocol and FIPA-request protocol with the agent mobility.

For almost all conversation protocols, unless the conversation is successfully carried until the end of protocol, the agents cannot obtain the expected current results. Therefore if we imagine that two agents communicate with each other only by the message passing at distance and the communication failure arrives during the agents conversation, this might lead to an unexpected result, i.e., incoherent internal states of each agents which had participated to this conversation. Such an incoherent knowledge about other agents or its environment would make an agent to take a bad decision. Or the agents must restart the entire conversation protocol once again. By using a mobile agent, we can prevent the conversation failure due to a sudden communication fault and bring to both an economic and safe collaboration. Thus we want to integrate the mobility within the model of conversation protocols.

For FIPA-request, we modeled the protocol component which starts with moveTo (dest) , i.e., the migration to the site of the requested agent (the invokee of FIPA-request protocol). After requesting a service, there are different responses possible (refuse, agree, inform). Getting safely a correct response is an important fact in deciding next behavior of the invoker (in our case, Travel Agency) agent. In case of Contract Net protocol, some conditions such that the manager must receive bid proposal *from all bidders*, or that the bidder to which was addressed the Call-for-proposal message *must always receive* this message and participate in the contract, etc. is not so important. In contrast, what is more important is that, after selecting a bidder, the manager must resume the conversation in a safe state with the chosen bidder (now called contractor). So we have modeled a Contract net protocol which leads to the agent migration to the site of a selected contractor.

5. Compilation and Implementation into Mobile Agent Platforms

The translation from a componential model to a mobile agent is achieved automatically. Once the designer describes the component specification using SCD (SoftComponent Description), it is compiled into Java code which is then encapsulated within a mobile agent s execution body.

5.1 Compilation of component descriptions

We have implemented a compiler which produces Java source code from the component model specified using SCD. From one component class description, the compiler produces a component class definition in Java. For example, the SCD description presented in figure 7a has been compiled into Java code in figure 8. The figure 9 shows the SCD compiling environment.

```
<…>     //descriptions of included packages

public class Contr_Manager extends SoftComponent
{
   Award award ;
   Place announced = new Place (“announced”) ;
   Place offered = new Place (“offered”) ;
   Place migrate = new Place (“migrate”) ;

   public Contr_Manager (String name, String
msgin, String msgout) {
         super (name, msgin, msgout) ;

   void beginContract (Condition condt) {
      if (plDistance.isCurrentState()) {
         plDistance.consumeToken () ;
         sendMessage (“callForProposal”, loadArgument
(new Vector(), condt)) ;
      }
   public void propose (Vector bidList) {
      if (bidList.size() > 0) {
         award = (Award) require (“selectService” ,
                  loadArgument ( <...>) ;
         sendMessage (“accept-proposal”, <......>) ; }}
         offered.nextState() ;
      }}
   public void move (AgentAddress dest) {
      if (offered.isCurrentState()) {
         self.moveTo (dest) ;
      }}
   public void _after_move () {
         migrated.setCurrentState() ;
   }
     <.....>
}        // end of class definition Contr_Manager
```

Figure 8. Generated Java code for the manager part s component of the Contract net protocol