Chapter 6

Implementation of MASSIC

6.1 Agent Platforms

There is a need for ‘agent-oriented’ programming language for ease of implementation. And luckily there exist a wide range of such languages. These languages provide libraries and infrastructure, such as

- RMA: the Remote Management Agent which handles the GUI interface
- AMS: the Agent Management Service - the core agent which keeps track of all programs and agents in the system
- DF: the Directory Facility, a yellow page service, where agents can publish their services.

It is imperative to understand the basic requirements of the system, which one wishes to develop, before finalizing agent platform as all agent platforms support plan-based agents only. One can classify agent platforms into three categories:

Category 1: The platforms that focus on internal reasoning and support goals and plans. More specifically these platforms support Belief Desire Intention (BDI). Some platforms in this category are: Procedural Reasoning System (PRS) [82], its successor SPARK (SRI Procedural Agent Realization Kit) [83], UMPRS [84], JAM[85], JACK [86], DECAF [87], Zeus [88], AgentBuilder [89], JADE [78] and JADEX [90].

Category 2: These platforms focus on communication among agents. They provide yellow page service to find a specific agent according to its description or white
Agent platforms

page service that helps in finding an agent according to its name. Some of these language are JADE, Zeus and OAA [91].

Category 3: The languages that concentrates on mobile agents like Grasshopper [92], D'Agents [93] and Aglets [94]

Category 1 languages are most promising for implementing the designs as they follow FIPA specifications. That is why JADE is selected for implementation of MASICC. Brief introduction of JADE is given in Appendix C (ii).

The objective of this section is to provide an overview of the available agent platforms. A more detailed discussion can be found in [95] or one can visit [96]. In the rest of this chapter, we illustrate how MASICC is implemented in JADE. JADE code, embedded in Java [97] (see Appendix C (i)) is used to describe the whole system.

6.2 Ontology Implementation

The agents need to communicate in a way that makes sense for them; share the same language, vocabulary and protocols. All this forms an ontology. An ontology is composed of two parts,

1. Nomenclature of the relationships between concepts, Figure 6.1, (used to describe the semantic and the structure).
2. Vocabulary that describe the terminology of concepts used by agents during communication (Figure 6.2)

In JADE, Ontology class is extended and a set of schemas that describe the structure of concepts, actions and predicates are added. BasicOntology class is also added as shown in Figure 6.1.
public class DiseaseOntology extends Ontology implements DiseaseVocabulary
{
    public static final String ONTOLOGY_NAME = "Disease_Ontology";
    private static Ontology instance = new DiseaseOntology();
    public static Ontology getInstance()
    {
        return instance;
    }

    private DiseaseOntology()
    {
        super(ONTOLOGY_NAME, BasicOntology.getInstance());
        try{
           // Concept Schemas........................................
           // Rural Doctor Account......................................
           ConceptSchema cs = new ConceptSchema(DOCTOR_ACCOUNT);
           cs.add(DOCTOR_ID, (PrimitiveSchema) getSchema(BasicOntology.STRING),
           ObjectSchema.MANDATORY);
           cs.add(DOCTOR_NAME, (PrimitiveSchema) getSchema(BasicOntology.STRING),
           ObjectSchema.OPTIONAL);
           cs.add(DOCTOR_PASSWORD, (PrimitiveSchema)
           getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

           add(as = new AgentActionSchema(QUESTION_QUERY), QuestionQuery.class);
           as.add(QUE_TYPE, (PrimitiveSchema) getSchema(BasicOntology.INTEGER),
           ObjectSchema.MANDATORY);

           add(as = new AgentActionSchema(PROBLEM_5), problem_5.class);
           as.add(ANS_5_1, (PrimitiveSchema) getSchema(BasicOntology.INTEGER),
           ObjectSchema.MANDATORY);
           as.add(ANS_5_2, (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN),
           ObjectSchema.MANDATORY);
           as.add(ANS_5_3, (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN),
           ObjectSchema.MANDATORY);
        }
    }

    Figure 6.1 Code Snippet of Childhood Disease Ontology used by agents
public interface DiseaseVocabulary {
    //-------> Disease vocabulary
    public int REGISTER = 1;
    public int LOGIN = 2;
    public int HELP = 3;
    
    public static final String QUE_1 = "The infant had convulsions??";
    public static final String QUE_2 = "The body appears yellow??";
    public static final String QUE_3 = "Stools have changed from usual pattern. High frequency and watery??";
    public static final String QUE_4 = "Does the child have cough or difficult breathing??";
    public static final String QUE_5 = "Does the child have fever??";
    public static final String QUE_6 = "Does the child have fever and measles now or within the last 3 months??";
    
    public static final String DISEASE_5 = "VERY SEVERE FEBRILE DISEASE."
             SOLUTION_5 = "Treat the child to prevent low blood sugar in Give paracetamol for high fever (38.5Deg. C) or above) in Refer URGENTLY to hospital."
             DISEASE_5_1 = "MALARIA."
}

Figure 6.2 Code Snippet of Childhood Disease Vocabulary used by agents

An Ontology package is developed that contains 21 java programs. For each main problem or main question (discussed earlier in section 5.1) a java class is made that implements AgentAction. A sample case is shown in Figure 6.3.

package ontologies;
import jade.content.*;

public class problem_4 implements AgentAction {
    private int A_4_l;
    private int A_4_2;

    public int getA_4_l()
    {
    }

Figure 6.3 Code Snippet for a main problem.
The interactions between agents are implemented through java classes. These classes represent the concepts and actions of MASICC. Some of them are listed below:

**Doctor Account**: concept of a HCP account at rural site

**Register Doctor**: action of registering the HCP doctor

**Operation**: concept of a MASICC operation

**Solution**: concept of receiving disease and treatment plan.

**Information**: concept of querying information about the disease.

**Problem**: result of an action that shares value(s) of sign-symptoms.

The subsequent section describes the usage of childhood disease ontology by an agent located at rural site.

### 6.3 Implementation of the agent at rural areas

In this section, the implementation of rural agent, namely UA, is discussed. This agent behaves as a client to the IPA. It uses or implements a common interface, defined in previous section 6.2. This interface defines the constant terms that comprise the specific language for the agents. For instance, **Register Doctor** class, shown in Figure 6.4, is used by UA for registering the HCP located at rural site or at PHC. This class is used in the main program designed for UA. This is shown in Figure 6.5.

The interaction between the agents follows a simple protocol. To create an account for the HCP or log-in the HCP or sending a main question query, the UA uses the REQUEST performative to IPA. This performative is a part of the ACL message. The IPA responds with an INFORM performative after processing the request or with an NOT_UNDERSTOOD if it is unable to decode the content of the message. To query about the specific set of sign-symptoms, this agent sends a QUERY_REF to the IPA.
Implementation of the agent at rural areas

agent which responds back with an INFORM after processing the query or with a NOT_UNDERSTOOD if it is unable to decode the content of the message. This is shown in Figure 6.6

```java
package ontologies;
import jade.content.*;

public class RegisterDoctor implements AgentAction {
    private String name;
    private String password;

    public String getName() {
        return name;
    }

    public String getPassword() {
        return password;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

**Figure 6.4** Code Snippet for Registering HCP at rural site

```java
void problem_5(int a_1, boolean a_2, boolean a_3, boolean a_4, boolean a_5, boolean a_6){
    problem_5 p5 = new problem_5();
p5.setA_1(a_1);
p5.setA_2(a_2);
p5.setA_3(a_3);
p5.setA_4(a_4);
p5.setA_5(a_5);
p5.setA_6(a_6);
sendMessage(ACLMessage.QUERY_REF, p5);
}
```

**Figure 6.6** Code Snippet showing Query_Ref performative used by UA
public class DiseaseClientAgent extends GuiAgent implements DiseaseVocabulary {

    private AID server;
    private Codec codec = new SLCodecQ();
    private Ontology ontology = DiseaseOntology.getInstance();

    transient protected DoctorRegisterGui myGui1;
    transient protected ChildDiseaseGui myGui2;

    String name = ((String)ev.getParameter(0).toString());
    String password = ((String)ev.getParameter(1).toString());
    RegisterDoctor(name, password);

    void RegisterDoctor(String name, String password) {
        RegisterDoctor rd = new RegisterDoctor();
        rd.setName(name);
        rd.setPassword(password);
        sendMessage(ACLMessage.REQUEST, rd);
    }

    else if (msg.getPerformative() == ACLMessage.INFORM) {
        try {
            ContentElement content = getContentManager().extractContent(msg);
            if (content instanceof Result) {
                Result result = (Result) content;
                if (result.getValue() instanceof Problem) {
                    Problem prob = (Problem) result.getValue();
                    alertGui(prob);
                    myGui.setVisible(true);
                    myGui.setAlwaysOnTop(true);
                }
            }
        }
    }
}

Figure 6.5 Code Snippet of UA

This above mentioned code has been developed using NetBeans IDE (Integrated Development Environment) [98]. The UA provides a mechanism by which the HCP registers himself with the IPA and gets a unique identification number, shown in
Figure 6.7. Using this unique ID he logs in whenever he wishes to seek help from IPA. This is shown in Figure 6.8.

Figure 6.7 GUI of UA for Registering HCP.

Figure 6.8 GUI of UA for Doctor Log-in.

The Figure 6.9 shows the main functionality of the UA. It provides an interface to the HCP to enter the child name, his age and the main problem. This information is sent to the IPA, which in turns sends back a set of related sign-symptoms whose values are to be observed and filled by the HCP. This is shown in Figure 6.10.
Implementation of MASICC

Figure 6.9 GUI of UA for sending the information to IPA and receiving back the disease and the treatment plan(s).

If the stools have changed from usual pattern?

For How Long days? [ ]
Is there Blood in stool? [ ]
Is the infant Lethargic or Unconscious? [ ]
Is the infant Restless or irritable? [ ]
Look for Sunken Eyes. [ ]
Pinch the skin of the abdomen. Does it go back ... [ ]

Figure 6.10 GUI of UA for sending observed sign-symptoms by HCP.
In the next section, implementation of IPA will be discussed in detail. The implementation will discuss the behaviors and the performatives used by IPA. GUI of IPA will also be depicted.

### 6.4 Implementation of the agent at urban areas

The agent at urban site or at CHC serves the UAs which are located at rural areas. Both the agents form client-server architecture. This agent is the core of MASICC as it imitates the behavior of a pediatrician.

This agent also uses the common interface provided in the childhood disease ontology as being used by UA too. This is shown in the code snapshot in Figure 6.11

```java
import ontologies.*;

public class DiseaseServerAgent extends GuiAgent implements DiseaseVocabulary {
    private int idCnt = 0;
    private AID server;
    ...
    private Codec codec = new SLCodec();
    private Ontology ontology = DiseaseOntology.getInstance();
    transient protected servergui myGui;

    protected void setup() {
        //-> Register language and ontology
        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(ontology);
    }
}
```

**Figure 6.11** Code Snippet of IPA implementing Childhood Diseases Ontology
Implementation of MASICC

Section 5.2.3 provides a detailed description of the behaviors used by IPA. Here the code snapshots are shown in Figures 6.12 and 6.13. Figure 6.11 depicts the OneShotBehaviour used to respond to one of the main questions. Similarly, Figure 6.13 shows the CyclicBehavior of receiving messages from UA. It also shows the extraction of performatives from the ACL message received from UA.

```java
class HandleProblem_7 extends OneShotBehaviour {
    private ACLMessage query;

    HandleProblem_7(Agent a, ACLMessage query) {
        super(a);
        this.query = query;
    }

    public void action() {
        try {
            ContentElement content = getContentManager().extractContent(query);
            problem_7 p7 = (problem_7)((Action)content).getAction();
            Object obj = processproblem_7(p7);

            if (obj == null) replyNotUnderstood(query);
            else {
                ACLMessage reply = query.createReply();
                reply.setPerformative(ACLMessage.INFORM);
                Result result = new Result((Action)content, obj);
                getContentManager().fillContent(reply, result);
                send(reply);
                System.out.println("Request processed.");
            }
        }
        catch(Exception ex) {ex.printStackTrace();}
    }
}
```

Figure 6.12 One Shot Behavior of IPA for replying to a main problem.
The GUI of this agent is also developed so that the pediatrician at CHC can view the proceedings of the diagnostic process. This is a fully automated agent and hence there is no need on the part of pediatrician to intervene. The GUI has been constructed for monitoring purpose only. This is shown in Figure 6.14. It depicts that
‘Ravinder’, a HCP, is currently seeking assistance. His unique identification number is ‘206323810’. This is generated by IPA only.

![GUI of IPA showing active UAs and unique ID.](image)

**Figure 6.14** GUI of IPA showing active UAs and unique ID.

The IPA is to process the whole diagnostic procedure. The UA is simply an interface between HCP and IPA. For instance, the IPA decides to suggest the HCP about the disease and its treatment plan(s). IPA fills the ‘Disease’ and the ‘Treatment’ fields of Figure 6.9. The revised version of the figure is shown in Figure 6.15.
Implementation of the agent at urban areas

Figure 6.15 GUI of UA: Partially filled by HCP and partially by IPA.

To view the interaction between agents of MASICC, the ‘Sniffer Agent’ provided by JADE environment has been used. It records the ACL messages being sent and received by various agents of the system. This mechanism is shown in Figure 6.16.

Figure 6.16 Sniffing the interaction between agents.
The above Figure also shows the Agent Management Service handled by an agent called ‘ams’ and the Remote Management Agent (RMA). RMA is responsible for managing GUI interfaces. The directory facility or the yellow page service is also depicted. This is handled by an agent named ‘df’. The ACL message sent by HCP through UA to IPA containing a REQUEST performative is shown Figure 6.17.

```
(REQUEST
  .sender (agent-identifier:name HCP@vijay:6677/JADE )
  .receiver (set (agent-identifier:name IPA@vijay:6677/JADE :addresses (sequence IOR:0000000000000020002 ))
  .content "((action (agent-identifier:name IPA@vijay:6677/JADE :addresses (sequence IOR:0000000000000020002 )) (Register Doctor":name "Ravinder":password max))")
 :language fipa-sl
 :ontology Disease Ontology)
```

**Figure 6.17** ACL message sent by UA to IPA.

Similarly the ACL message sent by IPA to UA informing the disease and its associated treatment plans are shown in Figure 6.18.

```
(INFORM
 :sender (agent-identifier:name IPA@vijay:6677/JADE :addresses (sequence IOR:000000000000000011...0000020002 ))
 :receiver (set (agent-identifier:name HCP@vijay:6677/JADE ))
 :content "((result (action (agent-identifier:name Pediatrician@vijay:6677/JADE :addresses (sequence IOR:000000000000011494...0000020002 )) (Problem_4 :A_4_1 20 :A_4_2 65 :A_4_3 false :A_4_4 true :A_4_5 true)) (solution :disease \"SEVERE PNEUMONIA OR VERY SEVERE DISEASE.\" :results \"Give pre-referral treatment. Give oxygen if present. Abort convulsion if present. Refer the client URGENTLY to hospital.\")")
 :reply-with Health_Pract@vijay:6677/JADE1140246218296
 :language fipa-sl
 :ontology Disease Ontology)
```

**Figure 6.18** ACL message sent by IPA to UA

In this section, the implementation details of the IPA have been discussed in detail. In the next section, the procedure used to verify the system is described.
6.5 Verification

It is mandatory for the system to adhere to the protocol [16]. This is the premise of MASICC. To verify the system, some sample cases are tested. The step-wise procedure, for a sample test case, is described below:

**Step 1:** It is assumed that a child having difficulty in breathing comes to the PHC.

**Step 2:** The HCP observes that child is coughing also.

**Step 3:** The HCP uses the MASICC (Figure 6.9) to send this information to a virtual pediatrician.

**Step 4:** The virtual pediatrician, more precisely, the IPA seeks more information from the HCP by sending a form which is to be filled by him. A sample form is shown in Figure 6.10.

**Step 5:** The IPA receives this information, process it and then supplies the disease and the treatment plan(s), shown in Figure 6.15.

This testing procedure has been conducted rigorously and it is ensured that the system works in accordance to the diagnostic protocol. Dr. Ravinder Mehta, a pediatrician, collaborated with us during testing of the system.