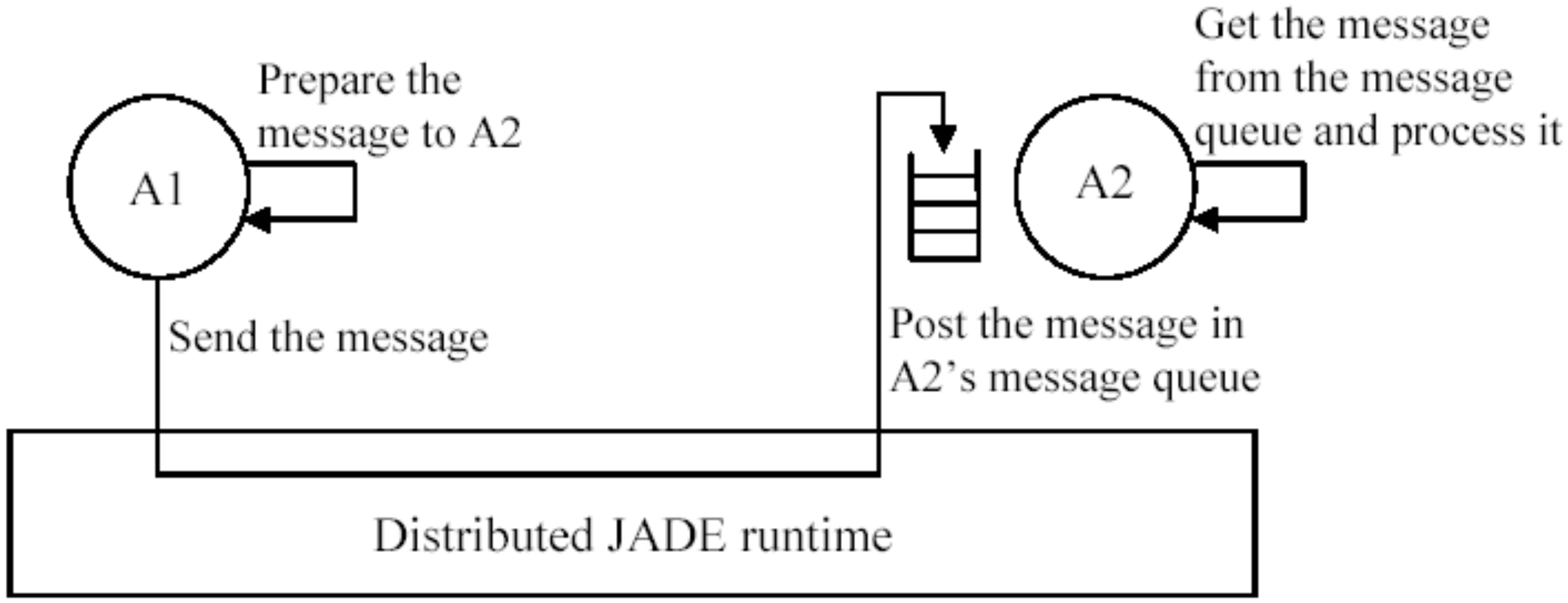
Distributed Artificial Intelligence and Intelligent Agents

Programming with JADE (2)

JADE message sending


Prepare the message to A2
A1
Send the message
Get the message from the message queue and process it
A2
Post the message in A2's message queue
Distributed JADE runtime

Structure of a JADE Message

- **Performative** - FIPA message *type* (INFORM, QUERY, PROPOSE, ...)
- Addressing
  - **Receiver**
  - **Sender** (initialized automatically)
- **Content** - This is the main content of the message
- **ConversationID** - Used to link messages in same conversation

- **Language** - Specifies which language is used in the content
- **Ontology** - Specifies which ontology is used in the content

- **Protocol** - Specifies the protocol
- **ReplyWith** - Another field to help distinguish answers
- **InReplyTo** - Sender uses to help distinguish answers
- **ReplyBy** - Used to set a time limit on an answer

JADE provides get and set methods for accessing all attributes

Sending messages

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID(“Peter”, AID.ISLOCALNAME));
msg.setLanguage(“English”);
msg.setOntology(“Weather-forecast-ontology”);
msg.setContent(“Today it’s raining”);
send(msg);
```

Book Buying

```
// Message carrying a request for offer

ACLMessage cfp = newACLMessage(ACLMessage.CFP);
   for (int i = 0; i < sellerAgents.length; ++i){
        cfp.addReceiver(sellerAgents[i]);
   }
cfp.setContent(targetBookTitle);
myAgent.send(cfp);
```

Catalogue of Communicative Acts

Communicative act	Information passing	Requesting information	Negotiation	Action performing	Error handling
Accept-proposal			✓		
Agree				✓	
Cancel				✓	
Cfp			✓		
Confirm	✓				
Disconfirm	✓				
Failure					✓
Inform	✓				
Inform-if (macro act)	✓				
Inform-ref (macro act)	✓				
Not-understood					✓
Propagate				✓	
Propose			✓		
Proxy				✓	
Query-if		✓			
Query-ref		✓			
Refuse				✓	
Reject-proposal			✓		
Request				✓	
Request-when				✓	
Request-whenever				✓	
Subscribe		✓			

Receiving messages

- Receive:

```
ACLMessage msg = receive();
if (msg != null) {
  // Process the message
}
```

- Blocking receive

```
ACLMessage msg = blockingReceive();
```

By using **`blockingReceive()`**, the receiving agent suspends all its activities until a message arrives

Receiving messages behaviour

```
/** Inner class OfferRequestsServer. This is the behaviour used by Book-seller
   agents to serve incoming requests for offer from buyer agents. If the
   requested book is in the local catalogue the seller agent replies with a
   PROPOSE message specifying the price. Otherwise a REFUSE message is sent
   back.
*/
private class OfferRequestsServer extends CyclicBehaviour
   {  public void action() {
        ACLMessage msg = myAgent.receive();
        if (msg != null) {
             // Message received. Process it
             String title = msg.getContent();
             ACLMessage reply = msg.createReply();
             Integer price = (Integer) catalogue.get(title);
             if (price != null) {
             // The requested book is available for sale. Reply with the price
                 reply.setPerformative(ACLMessage.PROPOSE);
                 reply.setContent(String.valueOf(price.intValue()));
             }
             else {
                 // The requested book is NOT available for sale.
                 reply.setPerformative(ACLMessage.REFUSE);
                 reply.setContent("not-available");
             }
             myAgent.send(reply);
        }
   }
} // End of inner class OfferRequestsServer
```

Blocking behaviour

```
public void action() {
  ACLMessage msg = myAgent.receive();
  if (msg != null) {
     // Message received. Process it
     ...
  }
  else {
     block();
  }
}
```

The above code is the typical (and strongly suggested) pattern for receiving messages inside a behaviour.

Message templates

```
public void action()
   {  MessageTemplate
                   mt =
       MessageTemplate.MatchPerformative(ACLMessage.CFP);
   ACLMessage msg = myAgent.receive(mt);
   if (msg != null) {
       // CFP Message received. Process it
       ...
   }
   else {
       block();
   }
 }
```

Conversations

```
/**
Inner class RequestPerformer. This is the behaviour used by Book-buyer agents to
   request seller agents the target book.
*/
private class RequestPerformer extends Behaviour {
private AID bestSeller;           // The agent who provides the best offer
private int bestPrice;            // The best offered price
private int repliesCnt = 0;       // The counter of replies from seller agents
private MessageTemplate mt;       // The template to receive replies
private int step = 0;
public void action() {
   switch (step)
   {  case 0:
        // Send the cfp to all sellers
        ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
        for (int i = 0; i < sellerAgents.length; ++i) {
                cfp.addReceiver(sellerAgents[i]);
        }
        cfp.setContent(targetBookTitle);
        cfp.setConversationId(“book-trade”);
        cfp.setReplyWith(“cfp”+System.currentTimeMillis()); // Unique value
        myAgent.send(cfp);
        // Prepare the template to get proposals
        mt=MessageTemplate.and(MessageTemplate.MatchConversationId(“book-trade”),
                MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
        step = 1;
        break;
```

Conversations (cntd.)

```
case 1:
     // Receive all proposals/refusals from seller agents
     ACLMessage reply = myAgent.receive(mt);
     if (reply != null) {
             // Reply received
             if (reply.getPerformative() == ACLMessage.PROPOSE) {
                     // This is an offer
                     int price =
                          Integer.parseInt(reply.getContent());
                     if (bestSeller == null || price < bestPrice) {
                             // This is the best offer at present
                             bestPrice = price;
                             bestSeller = reply.getSender();
                     }
             }
             repliesCnt++;
             if (repliesCnt >= sellerAgents.length) {
                     // We received all replies
                     step = 2;
             }
     }
     else {
             block();
     }
break;
```

Conversations (cntd.)

```
case 2:
    // Send the purchase order to the seller that
    // provided the best offer
    ACLMessage order=new
          ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
    order.addReceiver(bestSeller);
    order.setContent(targetBookTitle);
    order.setConversationId("book-trade");
    order.setReplyWith("order"+System.currentTimeMillis());
    myAgent.send(order);
    // Prepare the template to get the purchase order reply
    mt = MessageTemplate.and(
        MessageTemplate.MatchConversationId("book-trade"),
        MessageTemplate.MatchInReplyTo(order.getReplyWith()));
    step = 3;
    break;
```

Conversations (cntd.)

```
      case 3:
         // Receive the purchase order reply
         reply = myAgent.receive(mt);
         if (reply != null) {
                // Purchase order reply received
                if (reply.getPerformative() == ACLMessage.INFORM) {
                       // Purchase successful. We can terminate
                       System.out.println(targetBookTitle+
                              “successfully purchased.”);
                       System.out.println(“Price = ”+bestPrice);
                       myAgent.doDelete();
                }
                step = 4;
         }
         else {
                block();
         }
         break;
      }
    }
    public boolean done() {
         return ((step == 2 && bestSeller == null) || step == 4);
    }
} // End of inner class RequestPerformer
```