1. Jade Introduction, Basic Bahaviors

In this lecture, we will learn how to use the Jade system and we will create a simple chat system, which will allow us to send messages among the agents.

Jade Installation

The first step is to download the Jade library. The system can be downloaded for free from the http://jade.tilab.com/ webpage. It is enough to add the jade.jar library to the java classpath.

Starting Jade

Jade can be simply started by the command

```
java -cp jade.jar jade.Boot -gui
```

After start, the main Jade window opens. You can see that there are several agents running in the system: gui, df, and ams. The gui agent is the one you are looking at. DF (Directory Facilitator) is a yellow-pages service, where the agents in the system can register their services and can be found by the services. AMS (Agent Management System) is the agent which takes care of the whole system. You can also start the sniffer agent from the GUI. The sniffer can be used to observe the messages the agents send among themselves.

GUI can also be used to control the whole system, add and remove agents, send them messages. If you want to start a new agent, it needs to be in the classpath.

Exercise 1 Try to start an agent prepared for this seminar – cv1.mas.SimpleChatServer and send it a message from the GUI. If it works, it should print the message to the console.

New agents can be started while starting Jade. You can just add the name of the agent, colon, and the class which implements the agent. For example, if we want to start the SimpleChatServer agent from the command line, we can write:

```
java -cp $CP jade.Boot -gui server:cv1.mas.SimpleChatServer
```

where `$CP` contains the classpath with Jade and the agent and `server` is the name of the started agent.

One of the great advantages of Jade, and agent programming in general, is a simple distribution of the computation. Jade can be run on several computers at once. Each computer than runs a so called container, one of the computer runs a main container. All other containers connect to the main container. We can create a new container which connects to the by running:

```
java -cp $CP jade.Boot -container -host u-pl4 -agents […]
```

Creating a new agent

In order to create a new agent, we need to inherit from the base class `Agent`. During the initialization of the agent the `setup()` method is called the first. Here, the agent can register to the yellow pages and add its initial behavior (will be discussed later). During the termination of the agent, the `takeDown()`

method is called. Here, the agent can release its resources and deregister from the yellow pages (if it registered earlier).

Yellow Pages

In the system, there is by default the DF agent, which takes care and provides information about services provided by other agents in the system – the Yellow Pages. An agent needs to register with the DF agent before the service can be found in the DF. It can use the `DFService` class and its `register` method. The method needs, apart from the name of the agent, the description of the agent as `DFAgentDescription`, which contains a list of service descriptions `ServiceDescription`, which the agent provides. As an example, let us look at the registration of the `SimpleChatServer` agent mentioned above:

```
ServiceDescription sd = new ServiceDescription();
sd.setType("messaging-server");
sd.setName("server");

DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(this.getAID());
dfd.addServices(sd);

try {
        DFService.register(this, dfd);
} catch (FIPAException e) {
        e.printStackTrace();
}
```

Before the agent terminates, it needs to be removed from the Yellow Pages using the method `DFService.deregister()`. The same method can be also used to deregister individual services.

The agents can be found in the Yellow Pages using the `DFService.search()` method, which takes a `DFAgentDescription` parameter. The service then finds all agents which match the given description. For example, the following code finds all agent which provide any service of the type `messaging-server`.

```
ServiceDescription sd = new ServiceDescription();
sd.setType("messaging-server");

DFAgentDescription dfd = new DFAgentDescription();
dfd.addServices(sd);

DFAgentDescription[] servers = DFService.search(myAgent, dfd);
```

The first parameter of the `search` method is the name (AID) of the agent, which requested the search.

Behaviors

Each agent contains a set of behaviors it performs. If an agent has more than one behavior, they act as if they were run in parallel, however, they all run in a single thread and thus a long-running behavior can block other behaviors. If a behavior is done with its work, next one is selected randomly.

New behaviors can be created by implementing new classes inherited from an existing behavior. The simplest behavior is the `OneShotBehaviour`, which is executed once and once it finishes, it does nothing. Its actions are implemented in the `action` method. A very useful (and general) behavior is `SimpleBehaviour`, which can be used as a base for most behaviors. It has two methods: `action` and `done`. If it is executed, the body of the `action` method is performed and the behavior is added back to the behaviors of the agents unless the `done` method returns `false`.

The prepared agents use `CyclicBehaviour`, which executes always (i.e. the `done` method always returns `false`), and `TickerBehaviour`, which starts the `onTick()` method cyclically after a given amount of time.

The behaviors have a useful `block()` method, which blocks the behavior until a message is received, or until something (e.g. other behavior) calls the `restart()` method of the blocked behavior. The agent which runs the behavior can be accessed from the behavior through the `myAgent` property.

Sending messages

One of the most common actions agents in a multi-agent system do is sending messages to other agents. Messages provide a way to inform or ask other agents or propose them something. Sending messages is simple, one only needs to create an instance of the `ACLMessage` class, set it a content and recipient and call the `send()` method of the agent with the message as a parameter.

As an example, we can have a look at sending a message with the text "`Hello World`" to a receiver stored in the variable `receiver`.

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(receiver);
msg.setContent("Hello World");
myAgent.send(msg);
```

The parameter of the `ACLMessage` constructor, `ACLMessage.INFORM`, indicates the message is an information for the agent. We will discuss more of these so called performatives next time.

Agents for today's seminar

Today, we will implement a simple chat service. We will use agents of two types – server and client. These agents are implemented in two classes – `SimpleChatServer` and `SimpleChatClient`. `SimpleChatServer` accepts messages from all agents, prints them to the console and sends them to all clients it can find in the Yellow Pages (one `CyclicBehaviour` take care of all this). `SimpleChatClient` sends the "Hi" message to all server it finds in the Yellow Pages.

Exercise 2 Have a look at the sources of both the agents and try to understand how they work.

Exercise 3 Start both the agents and the sniffer and have a look at the messages among them. Are the messages different from the messages between the agent and the DF agent?

After you start the agents, you will notice the client does not receive any messages. This is caused by the fact that it is not registered in the Yellow Pages.

Exercise 4 Update the `SimpleChatClient` so that it registers as an agent who provides the `messaging-client` service. Do not forget to deregister the agent when it terminates.

The client also does not display any messages received from the server, it does not have any behavior to do so.

Exercise 5 Create a behavior which displays the messages received by the client. You can find an inspiration in the behavior which does the same in the server.

Sending messages between two agents on the same computer is not fun, but Jade can be run on multiple computers and you can send messages among yourselves.

Exercise 6 Connect to the main container on `u-pl4` a start your `SimpleChatClient` agent. You should start receiving messages from all other agents and the other agents should receive your messages.

Sending only "Hi" is also boring.

Exercise 7 If you have enough time, try to change the client agent in a way that lets send your own messages.

What have we learned today

1. Install and run Jade
2. Connect to a running platform on a different computer
3. Observe the communication among the agents with the sniffer
4. Register the agent in the Yellow Pages
5. Search the Yellow Pages
6. Send messages among the agents
7. Create simple behaviors

We will discuss 6 and 7 again next time. We will talk about communication protocols and we will implement more complicated behaviors.