JADE: Java Agent DEvelopment Framework
Advanced

Stefano Mariani
s.mariani@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2015/2016
1 Directory Facilitator

2 FIPA Interaction Protocols in JADE

3 JADE Agents & Java Swing
All the material presented in these slides is rearranged by the author from a collection of documents kindly made available by the JADE team.

Credits for all the stuff (text & images) go to the JADE team, in particular to Giovanni Caire.

Credits for all the mistakes go to the author.
Directory Facilitator

1. **Directory Facilitator**
   - API
   - Syntax
   - Usage

2. **FIPA Interaction Protocols in JADE**
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3. **JADE Agents & Java Swing**
Recap

What we already know

*By default*, a singleton **Directory Facilitator** (DF) exists for each **JADE** platform, which:

- provides the *yellow pages* service by keeping track of published services provided by advertising agents—be them local or remote
- should be *explicitly* contacted by **JADE** agents who wish to advertise their capabilities—both to submit an advertisement and to remove it
- supports the *publish/subscribe pattern* by offering a *notification service*
- can be *federated* with other DFs to implement a *truly distributed* yellow pages service
Outline

1. Directory Facilitator
   - API
     - Syntax
     - Usage

2. FIPA Interaction Protocols in JADE
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3. JADE Agents & Java Swing
What’s new

The DF service is implemented as a JADE agent – as the AMS – in class `jade.domain.DFService`

! being JADE DF FIPA-compliant, *all interactions with the DF must follow FIPA’s standards:*

→ interaction protocols taken from package `jade.proto`
→ ACL messages must adhere to the `FIPAManagementVocabulary` (ontology) in package `jade.domain.FIPAAgentManagement`
→ ACL messages content must adhere to the `SLOVocabulary` in package `jade.content.lang.sl`

...
JADE helps us

- static methods are provided to automatically build *semantically-correct ACL* messages:
  - `createRequestMessage()` to request the execution of a fipa-agent-management ontology action by the DF
  - `createSubscriptionMessage()` to request subscription for a given `DFAgentDescription` template
  - `decodeResult()` to process the content of the final message received as a result of `search()` operation
  - `decodeNotification()` to process the content of a notification message received as a consequence of a previous subscription
...to ease developer’s work, a set of static methods embedding such interaction protocols are provided by class DFService

- `register()` called by an agent wishing to advertise a service
- `deregister()` called by an agent who no longer offers a previously advertised service
- `search()` called by `client` agents looking for a service to exploit

! be careful ’cause all these methods are **blocking calls**, therefore every activity of the agent is suspended until success or failure of the call

→ if you need **asynchronous** interactions, go for the FIPA protocols approach
Outline

1. Directory Facilitator
   - API
   - Syntax
   - Usage

2. FIPA Interaction Protocols in JADE
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3. JADE Agents & Java Swing
The DFAgentDescription class (DFD)

The DFD is an entry in the DF, thus must contain (at least):

- the agent ID
- the set of services the agent wishes to advertise, in the form of ServiceDescription
- the set of *ontologies*, *protocols* and *languages* the agent is able to support/understand
The ServiceDescription class (SD)

The SD is a descriptor of the service the agent wishes to publish to the DF, thus must contain (at least):

- the service *name*
- the service *type*
- the set of *ontologies* and *languages* whose knowledge is required to exploit the service
- a number of *service-specific* properties
DF Entries Syntax III

DFAgentDescription {
    Name: AID (mandatory)
    Protocols: set of strings
    Ontologies: set of strings
    Languages: set of strings
    Services {
        Name: String (mandatory)
        Type: String (mandatory)
        Protocols: set of strings
        Ontologies: set of strings
        Languages: set of strings
        Properties: {
            Name: String
            Value: String
        }
    }
}

Figure: Pseudo-code view of a DF entry
Outline

1. Directory Facilitator
   - API
   - Syntax
   - Usage

2. FIPA Interaction Protocols in JADE
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3. JADE Agents & Java Swing
DF API Usage I

A1: serviceX
   - serviceY

A2: serviceZ

A3: serviceW
   - serviceK
   - serviceH

Yellow Pages service

DF

A4

Search for agents providing the required services

A5

A6

Exploit required service

Publish provided services
Registering to the DF

1. instantiate a DFAgentDescription object
   → DFAgentDescription dfd = new DFAgentDescription();

2. fill in (at least) its Name field with the advertising agent AID
   → dfd.setName(getAID());

3. instantiate a ServiceDescription object
   → ServiceDescription sd = new ServiceDescription();

4. fill in (at least) its Name and Type fields with meaningful strings
   → sd.setType("buyer");
   sd.setName("online trad");

5. add the ServiceDescription to the DFAgentDescription
   → dfd.addServices(sd);

6. call DFService.register(this, dfd);
Deregistering from the DF

Since dead agent’s AIDs are automatically removed *solely from the AMS*, it is a good practice to deregister agents upon death

- a good place where to do so is in `takeDown()` callback method
  
  $$ \rightarrow \text{DFService}.\text{deregister}(\text{this}); $$

- keep in mind that each agent is allowed to have *only one entry* in the DF
  
  $$ \rightarrow \text{each attempt to register an already registered agent throws an exception} $$
Client agents may query the DF to know if any agent offers the services they are looking for and then acquire their AIDs:

1. create a DFD (with no AID, obviously...) filling its fields with the properties you look for
   
   → `DFAgentDescription dfd = new DFAgentDescription();
   ServiceDescription sd = new ServiceDescription();
   sd.setType("buyer");
   dfd.addServices(sd);

2. specify as `SearchConstraints` that you want to get all the agents offering the service (skip this if you need only one)
   
   → `SearchConstraints all = new SearchConstraints();
   all.setMaxResults(new Long(-1));`
DF API Usage V

Browsing the DF II

1. Launch the search process (skip last parameter if skipped previous point)
   \[ \rightarrow \text{DFAgentDescription[]}\text{result} = \text{DFService.search(this, dfd, all);} \]

2. Extract the AID(s) from the results set
   \[ \rightarrow \text{AID[]}\text{providers} = \text{new AID[results.length];}
   \text{for (int i = 0; i < result.length; i++) { providers[i] = results[i].getName(); } } \]

Check the \text{ds.lab.jade.bookTrading} example for the whole code.
Dirrectory Facilitator

DF API Usage VI

Subscribing to the DF I

**JADE** agents can ask the DF to **notify** them *as soon as* a given service is advertised:

1. as usual, create a DFD suited for the service you wish to be notified about...
   
   →  `DFAgentDescription dfd = new DFAgentDescription();
       `ServiceDescription sd = new ServiceDescription();
       sd.setType(...);
       dfd.addServices(sd);

2. ...configure your chosen `SearchConstraints` (if you please)...
   
   →  `SearchConstraints sc = new SearchConstraints();
       sc.setMaxResults(new Long(1));

...
Subscribing to the DF II

... then, perform your subscription

→ send(
    DFServe.createSubscriptionMessage(this, getDefaultDF(), dfd, sc)
);

Now the DF will send an `ACLMessage.INFORM` to the subscribed agent whenever an agent matching the supplied description registers
Outline

1. Directory Facilitator
   - API
   - Syntax
   - Usage

2. FIPA Interaction Protocols in JADE
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3. JADE Agents & Java Swing
FIPA Interaction Protocols in JADE

Interaction Protocols I

FIPA definition

“Predefined sequences of messages that can be reused in different domains to implement a given interaction”—some kind of “design pattern” for communications

The jade.proto package

jade.proto contains behaviours implementing both the initiator and responder roles in most common interaction protocols

- managing the flow of messages and checking that it is consistent to the protocol
- providing callback methods that can be overridden to take the necessary actions when a message is received
Interaction Protocols II

(Some) Protocol classes I

**AchieveRE[Initiator/Responder]** factorization of all the FIPA Request-like interaction protocols\(^a\), that is, those in which the initiator aims to achieve a **RE (Rational Effect)** and needs to verify if it has been achieved or not.

**ContractNet[Initiator/Responder]** allows the initiator to send a **Call for Proposal** to a set of responders, evaluate their proposals and then accept the preferred one (or even reject all of them).

\[^a\]such as FIPA-Request, FIPA-query, FIPA-Request-When, FIPA-recruiting, FIPA-brokering.
Propose[Initiator/Responder] allows the initiator to send a PROPOSE message to the participants indicating *its will to perform some action if they agrees*. The participants responds by either accepting or rejecting such proposal, then the initiator either carries out the action or not accordingly.

Subscription[Initiator/Responder] allows the initiator to *subscribe* to a target agent for certain kind of events. If the participant agrees, it communicates all content *matching the subscription condition* using an INFORM–RESULT.

... refer to *JADE* API for the others.
Outline

1 Directory Facilitator
   - API
   - Syntax
   - Usage

2 FIPA Interaction Protocols in JADE
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3 JADE Agents & Java Swing
AchieveRE I

Figure: FIPA AchieveRE protocol message flow
AchieveRE II

**AchieveREInitiator**

*Initiator role* for FIPA request-like protocols

- constructed by passing the protocol-starting *ACL* message
  - be sure to set the protocol field of the *ACLMessage* with the proper constant taken from *FIPANames.InteractionProtocols* in package *jade.domain*
- to be extended by overriding its *handle[...] callback* methods, which provide hooks to handle all the states of the protocol
  - e.g. *handleAgree(), handleInform(), ...*
  - be aware of the functioning of callbacks such as *handleOutOfSequence(), handleAllResponses(), handleAllResultNotifications*—refer to *JADE programmer’s guide*
- manages an *expiration timeout* expressed by the value of the *reply-by* slot in *ACLMessage*
  - as defined by FIPA, such timeout refers to the *first response*: second response timeouts can be managed “by hand”
AchieveRE III

**AchieveREResponder**

*Responder role* for FIPA request-like protocols

- constructed by passing the MessageTemplate describing $ACL$ messages we’d like to manage
  - method `createMessageTemplate` is provided to create templates for each interaction protocol
- to be extended by overriding its `handle/prepare[...]` *callback* methods, which provide hooks to handle all the states of the protocol
  - `handleRequest()` to reply to first initiator message
  - `prepareResultNotification()` to send the final response about the RE achieved
  - ...
FIPA Interaction Protocols in JADE

Achieve RE IV

```java
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
addBehaviour(new AchieveREInitiator(this, msg){
    @Override
    protected void handleAgree(ACLMessage agree) {
    }
    @Override
    protected void handleFailure(ACLMessage failure) {
    }
    @Override
    protected void handleInform(ACLMessage inform) {
    }
    @Override
    protected void handleNotUnderstood(ACLMessage notUnderstood) {
    }
    @Override
    protected void handleRefuse(ACLMessage refuse) {
    }
});

Figure: JADE AchieveREInitiator
AchieveRE V

```java
MessageTemplate template = AchieveREResponder.createMessageTemplate(
    FIPANames.InteractionProtocol.FIPA_REQUEST);
addBehaviour(new AchieveREResponder(this, template){
    @Override
    protected ACLMessage handleRequest(ACLMessage request)
        throws NotUnderstoodException, RefuseException {
        return new ACLMessage(ACLMessage.AGREE);
    }
    @Override
    protected ACLMessage prepareResultNotification(ACLMessage request,
        ACLMessage response) throws FailureException {
        return new ACLMessage(ACLMessage.INFORM);
    }
});
```

**Figure:** JADE AchieveREResponder
Outline

1. Directory Facilitator
   - API
   - Syntax
   - Usage

2. FIPA Interaction Protocols in JADE
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3. JADE Agents & Java Swing
ContractNet I

- cfp
  - action
  - preconditions1

- not-understood

- refuse
  - reason

- propose
  - preconditions2

  Deadline for proposals

  - reject-proposal
    - reason

  - accept-proposal
    - proposal

  - failure
    - reason

  - inform
    - Done(action)

  - cancel
    - reason

the manager cancels the contract due to a change of situation
ContractNet II

**ContractNetInitiator**

*Initiator role* for FIPA contract-net protocol

- constructed by passing the protocol-starting *ACL* message
  
  ! again, use FIPANames.InteractionProtocols to set the protocol field of the ACLMessage

- to be extended by overriding its *handle[...]* callback methods
  
  e.g. `handlePropose()`, `handleInform()`, ...

  ! be sure to implement `handleAllResponses()` by adding to the *acceptances* Vector all the `ACLMessage.ACCEPT_PROPOSAL` `ACL` messages to send

- manages the *expiration timeout*

  ! again, reply-by timeout refers to the *first response*

  ! late answers *are not consumed*, thus remain in the agent message box
ContractNet Responder

**Responder role** for FIPA contract-net protocol

- constructed by passing the proper `MessageTemplate` again, use method `createMessageTemplate`
- to be extended by overriding its `handle[...] callback` methods
  - `handleCfp()` the initial CFP message
  - `handleAcceptProposal()` when an ACCEPT_PROPOSAL message is received from the initiator
- ...

Check the `ds.lab.jade.bookTrading.contractNet` example for the code.
Outline

1. Directory Facilitator
   - API
   - Syntax
   - Usage

2. FIPA Interaction Protocols in JADE
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3. JADE Agents & Java Swing
Responder Behaviours I

Cyclic vs. single-session responders

Responder behaviours may have two forms:

**Cyclic** Serve interactions initiated by different agents *sequentially*
1. wait for the protocol initiation message
2. serve the protocol
3. go back waiting for a new protocol initiation message

**Single-Session** Serve interactions initiated by different agents *in parallel*
1. get the protocol initiation message in the constructor → requires an external behaviour to be used
2. serve the protocol
3. terminate

Check the jade.proto package to learn more.
Outline

1. Directory Facilitator
   - API
   - Syntax
   - Usage

2. FIPA Interaction Protocols in JADE
   - Achieve Rational Effect
   - Contract Net
   - More on Responders

3. JADE Agents & Java Swing
What’s the problem?

Whenever developing JADE agents which need to interact with a Java GUI, the *thread-per-agent* concurrency model of JADE agents must work together with the Swing *Event Dispatcher Thread (EDT)* concurrency model.
More in detail

- As you should know, the Swing framework is not thread-safe, so any code that updates the GUI elements must be executed within the EDT.
  
  → Since modifying a model object triggers an update of the GUI, model objects too have to be manipulated just by the EDT.

- The SwingUtilities class exposes two static methods to delegate execution of Runnable objects to the EDT.

  invokeLater() puts the Runnable into the System Event Queue (SEQ) (accessed by the EDT only) and returns immediately—asynchronous call.
  
  invokeAndWait() puts the Runnable into the SEQ and blocks waiting its completion—synchronous call.
GuiAgent class

To develop JADE agents interacting with a GUI, simply extend GuiAgent class in package jade.gui

onGuiEvent(GuiEvent e) may be viewed as the equivalent of the actionPerformed() method in Java Swing, that is, a callback invoked by JADE platform as soon as a GuiEvent is generated

postGuiEvent(GuiEvent e) used by the agent’s GUI to queue GUI events for later processing—similar to queueing ACL messages in its mailbox
GuiEvent class

A GuiEvent object has:

- two *mandatory* attributes
  - `source` the Object source of the event
  - `type` an integer identifying the kind of event generated
- an optional list of parameters eventually used for events processing

  - `addParameter()` takes the Object to add as a GuiEvent parameter
  - `getParameter()` gets the *i*-th parameter
  - `getAllParameter()` returns an Iterator to browse all parameters
One advice

From JADE programmer’s guide:

“In general, it is not a good thing that an external software component maintains a direct object reference to an agent, because this component could directly call any public method of the agent, skipping the asynchronous message passing layer and turning an autonomous agent into a server object, slave to its caller. The correct approach is that to gather all the external methods into an interface, implemented by the agent class, then an object reference of that interface will be passed to the external software component (e.g., a GUI) so that only the external methods will be available from event handlers.”

Check the ds.lab.jade.bookTrading.gui example carefully.
JADE: Java Agent DEvelopment Framework
Advanced

Stefano Mariani
s.mariani@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2015/2016