The American University at Cairo
The Computer Science Department
Csci485-Intelligent Agents
Spring 2006 Dr Rafea
JADE Tutorial
1. Introduction

Intelligent Agents have been extensively used in different aspects of IT world. Intelligent agents have been used in the web, data mining, and software personalization. Before going any further, it would be better to define what an IA is and what its characteristics are. IA is an application that exhibits some AI features that can aid either other application or other applications in the process of gathering or analyzing information. There are many definitions for an agent depending on its role some of these definitions are as follows

- **The MuBot Agent**
  
  "The term agent is used to represent two orthogonal concepts. The first is the agent's ability for autonomous execution. The second is the agent's ability to perform domain oriented reasoning"

- **The AIMA Agent**

  "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors"

- **The IBM Agent**

  "Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires"

- **The KidSim Agent**

  "Let us define an agent as a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. 'Special purpose' distinguishes them from entire multifunction applications; agents are typically much smaller"
Actually, due to the diverse number of applications of IAs, it became harder and harder to come up with a general definition. Thus, rather than listing all IA possible definitions, it would be more reasonable to list its key characteristics. The key characteristics of an IA are

- The Agent should exhibit certain level of self control
- The Agent should exhibit certain level of decision making
- The Agent should pursue a certain goal taking into consideration surrounding environment
- The Agent should communicate to be able to exchange information in a social based pattern
- The Agent should be loosely coupled that provides each agent with required independence
- The Agent communication should be asynchronous such that messages can be discarded or ignored
- The Agent must speak a universal language to enhance the social pattern of operation
- The communication language should be both specific and formal in the sense that possible dialects could exist however these dialects should reference the same formal language basics.
2. Intelligent agents based technologies

2.1. Intelligent agent Paradigm

For the previous requirements of intelligent agents to be realized a certain paradigm or context where IAs interact must be defined and regulated. Following the previous characteristics of IAs, the following requirements for designing an IA environment could be derived.

- **Autonomy**
  Autonomous means that the agent is in control of its actions within a certain environment

- **Flexibility**
  Flexibility means that the agent should be to handle different forms of information including incomplete and imprecise information while making the action flexible not hard coded o scripted

- **Personlizability**
  The agent should be able to handle different individual needs

- **Adaptability**
  The agent should be able to not only to learn from its experience but also change its behaviors according to that experience

- **Mobility**
  The agent should be able to move through networks such that it could access right resource at right places

- **Sociability**
  The agent is not a standalone application such that it should be able to communicate with other parties either other agents or human beings.

- **Reactive behavior**
The agent should be able to change its behavior according to the surrounding environment in an organized way following reasonable time constraints

- Proactive behavior
  Not only should an agent be able to react to environment changes through sensing changes, but also it should be able to act in a goal oriented style to process certain needs

- Temporally continuous
  The agent is a continuously running application

- Temporal storage
  The agent should be able to store information according to surrounding environment constraints

The previous requirements necessitate the usage of certain key technologies that could realize the previous requirements.

2.2. Peer-to-peer technology

In the realm of distributed computing, there are three main known reference models for networking interactions and architectures. These three models differ in how the clients register, join, and interact with other nodes in the network. The three models can be listed as follows

2.2.1. Client/Server Architecture

In this architecture there are no direct interactions between client nodes, but rather all interaction must first go through the server. On the one hand, the server in this model is completely reactive such that it only responds to client requests. On the other hand, the clients are proactive and they are the ones specifying the requests and the goals required. This system prevails in the web interactions. This model has drawbacks in that it forces interactions
through a server for some interaction that does not require such level of regulation like chatting applications and multiplayer games.

2.2.2. Peer-to-peer Architecture

In this architecture the interactions between client node are direct and no middle man is required to process the interactions. The clients are reactive and proactive I the same time. The client node can register to and join the network, and interact with other nodes. This model has drawbacks in security and management. The more the network grows the more it becomes hard to control or manage the user interactions. Moreover, the direct and numerous interactions might affect the bandwidth and network coherence. Moreover, node discovery become harder in the absence of server nodes.

2.2.3. Hybrid Architecture

In this architecture, nodes are allowed to communicate with each other. However, certain services are only allowed through nodes that function as servers. For instance, node registration and node discovery are done through these nodes which boost the network coherence and realize easier and more controlled population of a network. Moreover, security is enhanced through better traffic management.

The next diagram illustrates the difference between the previously mentioned protocols. Usually IA based system are based in the peer-to-peer because of its flexibility in independent client interconnection which would realize the requirements of sociability, reactivity, and proactivity.
Figure 1
Client/Server (left), pure P2P (right), hybrid P2P (centre)

a) Client/Server  b) Hybrid P2P  c) Pure P2P
2.3. The Foundation for Intelligent Physical Agents (FIPA) standard

Due to the diversity of IAs requirements and applications, standardization became indispensable. In order to write an application that complies with the previous requirements to allow certain features like mobility, adaptability, and flexibility, a standardized specification for agent interactions had to be developed. This would mean that different implementations of agent environments that comply with certain standard could develop agents that are capable of communication, collaboration, and interoperation. FIPA is an IEEE based organization that promotes agent based technology through the release of specifications for agent standards. The following diagram explains interactions and relationships between different specifications through categorizing them into five categories that each implements number of specifications as follows
2.4. FIPA Applications

Specification for specific fields or areas where FIPA can be applied

- FIPA Nomadic Application Support Specification
- FIPA Agent Software Integration Specification
- FIPA Personal Travel Assistant Specification
- FIPA Audio Visual Entertainment and Broadcasting Specification
- FIPA Network Management and Provisioning Specification
- FIPA Personal Assistant Specification
- FIPA Messaging Buffer Service Specification
- FIPA Quality of service Specification

2.5. FIPA Abstract Architecture

Specifications for entities and parts that represent agents and their environment

- FIPA Abstract Architecture Specification
- FIPA Domains and Policies Specification

2.6. FIPA Agent Communications

- FIPA ACL Message Structure Specification
- FIPA Ontology Service Specification

2.6.1. Interaction Protocols

Deals with pre-agreed message exchange protocols

- FIPA Request Interaction Protocol Specification
- FIPA Query Interaction Protocol Specification
- FIPA Request When Interaction Protocol Specification
- FIPA Iterated Contract Net Interaction Protocol Specification
- FIPA English Auction Interaction Protocol Specification
• FIPA Dutch Auction Interaction Protocol Specification
• FIPA Brokering Interaction Protocol Specification
• FIPA Recruiting Interaction Protocol Specification
• FIPA Subscribe Interaction Protocol Specification
• FIPA Propose Interaction Protocol Specification

2.6.2. Communicative Acts

Deals with different ways of writing ACLs

• FIPA Communicative Act Library Specification

2.6.3. Content Languages

Deals with different structures of the ACLs content

• FIPA SL Content Language Specification
• FIPA KIF Content Language Specification
• FIPA CCL Content Language Specification
• FIPA RDF Content Language Specification

2.7. FIPA Agent Management

Specification for Agent managements within and across different zones

• FIPA Agent Management Specification
• FIPA Agent Discovery Service Specification
• FIPA JXTA Discovery Middleware Specification

2.8. FIPA Agent Message Transport

Deals with the transportation of message through different network types including wireless and wired

• FIPA Agent Message Transport Service Specification
• FIPA Messaging Interoperability Service Specification

2.8.1. ACL Representations
Deals with different representation of ACL messages

- FIPA ACL Message Representation in Bit-Efficient Specification
- FIPA ACL Message Representation in String Specification
- FIPA ACL Message Representation in XML Specification

2.8.2. Envelope Representations

Deals with different envelop representations of ACL messages

- FIPA Agent Message Transport Envelope Representation in XML Specification
- FIPA Agent Message Transport Envelope Representation in Bit Efficient Specification

2.8.3. Transport Protocols

Deals with different protocols for ACL delivery and transportation

- FIPA Agent Message Transport Protocol for IIOP Specification
- FIPA Agent Message Transport Protocol for WAP Specification
- FIPA Agent Message Transport Protocol for HTTP Specification

The next diagram explains how the specifications collaborate in designing an interconnected system for agent communication. Each stage of communication is bound to a specification that explicitly specifies its requirements, input, and output. This layered structure of specification oriented communication enable abstraction and independent definition of each layer.
3. Introduction to JAVA Agent Development Framework (JADE) Middleware

3.1. What is JADE

A middleware is a set of high-level API designed to take advantage of code reusability and improve OS independence. JADE is a java based middleware that complies with FIPA specifications offering an API in the form of JAVA packages that can be used to implement agents. The goal of JADE is to present the developers with an agent development framework that ease the process of implementing intelligent agents while realizing the requirements of agents discussed in section one. Not only does JADE comply with FIPA specification resulting in universal interoperability, but implements the framework using java which even eliminate the constraint of having a common OS. Any machine with JAVA virtual machine can run JADE agents regardless of the OS running. This would also boost the agent mobility requirement. The next diagram explains how a middleware resides.
3.2. Why JADE

JADE presents the developers many unique features that make it a remarkable tool for designing agents. Some of these features can be listed as follows

- FIPA Compliance
- OS Independent
- Peer-to-peer IA approach enabling distributed Agents
- Abstraction
- Extensive API
- Open source software under LGPL license
- Flexible Transport protocols like (RMI, HTTP, etc...)
- Multithreaded Agents are possible
- GUI interface
- Remote Administration
- Need to Know coding paradigm
- Minimal foot print
- Mobility of code and execution state
- Yellow pages service
- Security and authentication mechanisms
4. **JADE Overview**

The JADE platform is constructed of three main structures that interact and communicate in a way that both abide by the FIPA specifications and ease the process of developing distributed agents. These structures can be divided into three categories as follows: JADE run time environment, JADE libraries, and JADE tools.

**4.1. JADE run time environment**

The JADE run time environment consists of containers and platforms.

**4.2. JADE Container**

This exists in any host that runs agents. This consists of the agents in addition to the JRE environment that is why it is called container because it contains the agents.
4.3. JADE Main Container

This only exists in one host—usually the one that ran the first agent—which simplifies a central registry center. The exemplifies a collection of running agents in addition to three other structures:

- **Agent Management Service (AMS)**

  Keeps track with agents' life cycles or the white pages and can be found only in the main container.

- **Directory Facilitator (DF)**

  Responsible for Yellow pages service such that agent discovery become easier and eliminating the shortcomings of peer-to-peer technology.
4.4. JADE Platform

The next diagram describes a model system of JADE where different agents from different containers interact.

The JADE platform consists of the following elements.

- **Application Agent**
  
  The agent itself implemented and running.

- **RMI Registry**
  
  The Remote method invocation Jade registry.

- **Java container**
  
  This is the run time environment designed for the agents on any host.

- **Java Main container**
  
  This is the run time environment of the first agent that runs and includes the AMS and the DF.

- **JRE 1.x**
  
  The java run time system provides the agents with execution environment.
4.5. **JADE Distributed Platform over different networks**

The JADE system is designed to allow different agents to communicate regardless of their location or the systems they are running on. The next diagram describes how one or more JADE platforms can interact over a network. The size of the network can range from simple to very complex like the Internet. In the case of complex networks like the Internet, the role of FIPA and JADE stands out. The interactions over the Internet using JADE are simplified thanks to its structured and well-thought implementation.
4.6. JADE Packages

The JADE environment provides the developer with a set of libraries in the form of Java packages such that the developer need not implement an agent from scratch, but rather extends the classes and implements its methods. The following list contains some of the basic JADE packages that are usually extended.

- **Jade.Core**
  The kernel of the system and includes the agent class
- **Jade.Core.Behaviours**
  Includes different agent action types ranging from simple to more complex
- **Jade.Lang.ACL**
  Includes the implementation of Agent Communication Language
- **Jade.Content**
  Includes the classes needed to write user-defined ontologies
- **Jade.Domain**
  Includes all agent management system classes
- **Jade.Gui**
  Includes all the classes required to create GUI interfaces for editing agents
- **Jade.Mtp**
  Includes interface for creating message transport protocol
- **Jade.Proto**
  Includes classes of prototype protocol that would help the user creating his/her own protocol
- **Jade_WRAPPER**
  Includes functions that allow wrapping of JADE high-level functionality such that it could be used by other applications which access JADE as a library
4.7. Jade Tools

The JADE environment provide the user with set of GUI interfaces that ease the process of developing agents through making either agent creation, management, logging and debugging more user friendly. These tools will come in handy while designing any agent based system. The next listing aims at providing a general introduction to each tool. A detailed introduction will be included in the next section.

- Remote Management Agent (RMA)
  RMA is used to remotely administrate agents from different containers within one platform. There can be more than one RMA as long as they have different names.

- Dummy Agent
  Dummy agent is an agent in definition that is used to communicate with other agents within the container to enable the user to trace communication messages.

- The Sniffer
  The JADE sniffer is very similar to network sniffers in the sense that they keep track with all the messages exchanged among different agents. A sniffer can sniff messages directed to a specific agent or a specific group of agents.

- The Socket Proxy Agent
  This is an ordinary agent that is used to facilitate the communication between agents through sockets. This is similar in meaning to normal proxy servers.

- The DF GUI

- The SL Language formatter
  This is used to format the user expression in order to make it more readable.

- The Log Manager Agent
  This is used to log the agents' interactions on any JADE enabled system.
5. JADE development how-to

5.1. Install JADE

As previously mentioned JADE is a set of libraries with some tools included. JADE does not exist in a form of single executable file like .EXE or .MSI through which one can start an application. JADE is rather divided into number of JAR. In order to run the JAR files please follow the next instructions

- Visit JADE homepage and sign in for a user name and password
- After logging in download the JADEALL.zip file into your C driver
- After downloading all the files please extract the archive and all the zipped archives inside it with the same name as their zipped copies.
- After extracting the files you will find five main directories
  - JADE-BIN-3.4
    - This includes the binary files of the JADE environment
  - JADE-DOC-3.4
    - This includes very useful documentation for the JADE environment
  - JADE-EXAMPLES-3.4
    - This includes some examples on JADE agents
  - JADE_SRC-3.4
    - This include the source code of the JADE system in LGNU
  - META-INF
    - Some meta info about the system
- Before proceeding with the next part create a directory inside the JADE-ALL-3.4 directory and call it for instance "Project". Notice** case sensitivity is a big issue here. Inside this directory you will start the implementation of your agent project.
• In order to access the JAR files of the JADE-BIN file, you must change the directory to that specific directory or else you can add these directories to your system environment class path through the following command:

  Set
  
  Path=c:\j2sdk1.4.2_11\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\JADE-all-3.4\ 
  java -classpath C:\JADE-all-3.4\JADE-bin-3.4\jade\lib\jade.jar;C:\JADE-all- 
  3.4;C:\JADE-all-3.4\JADE-bin-3.4\jade\lib\jadeTools.jar;C:\JADE-all-3.4\ 
  JADE-bin-3.4\jade\lib\iiop.jar;C:\JADE-all-3.4\JADE-bin-3.4\jade\lib\commons-codec\commons-codec-1.3.jar;C:\JADE-all- 
  3.4\JADE-bin-3.4\jade\lib\crimson.jar;C:\JADE-all-3.4\JADE-bin- 
  3.4\jade\lib\http.jar; jade.Boot %1 %2 %3 %4 %5 %6 %7 %8 %9

• You can save this command in a file and change it into a .BAT file. You can find a copy of that file on the doctor’s homepage that is called "JADEStart.BAT"

• After the previous command you can access the JADE command from any place on your system.

• In order to start JADE platform you must instruct the system to start a JADE platform on this specific computer that will contain the main container which holds both the RMA agent and the DF agent. This is done through the following command which will start the RMA in a window.

  C:\JADEStart.BAT –gui

• When using this command you specify for java run time the JADE jar files such that JAVA could find them when needed.
Now we created our first container which in this case is our main container. We can see that the container is loaded inside a platform called "home-*******:1099" which exemplify the computer name and the port number of RMI which is the protocol used for message transportation.
5.2. Implement an agent

In this section we will learn how to load or start an agent inside your container. This can be done in number of steps.

- Write you own agent file

This will be discussed later on in this tutorial currently lets consider that we wrote our own agent using jade packages and saved it in the name example under the folder jade-bin-3.4/

- Compile you new agent using jade libraries

In order for us to be able to correctly compile the agent file written using jade libraries we must register to the java compiler some libraries and class files. This can be done through another command file as follows

```java
set path=c:\j2sdk1.4.2_11\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\JADE-all-3.4\javac -classpath C:\JADE-all-3.4\JADE-bin-3.4\jade\lib\jade.jar;C:\JADE-all-3.4\JADE-bin-3.4\jade\lib\jadeTools.jar;C:\JADE-all-3.4\JADE-bin-3.4\jade\lib\uiop.jar;C:\JADE-all-3.4\JADE-bin-3.4\jade\lib\commons-codec\commons-codec-1.3.jar;:. %1 %2 %3 %4 %5 %7 %8 %9
```

It is also possible to convert this command into .BAT file and use it with different files regardless of your location on your system. You can find a copy of that file on the doctor homepage called "JADECompile.BAT"

C:\JADECompile.BAT example.java

After executing this command you will have the class file resulting from the compilation inside the same directory as the source java files.
5.3. Load your agent into JADE environment

Actually, there are two ways through which we can load agents into JADE

- Loading agents through cmd

  o In order to load the agent into JADE using cmd, you only have to specify this while using the "Jadestart" command mentioned earlier while specifying extra parameters like the following

    `C:\Jade-All-3.4\JadeStart.bat –gui <agentname>:<agentclasspath>`

  o Notice that the agent name is just a nickname for identifying the agent. While, the agentclasspath is where JADE can find the class file of the agent resulted from the compilation. In this case, the agent source code resided in a directory called Project with the name Project. Thus, the command would look like the following

    `C:\Jade-All-3.4\JadeStart.bat –gui projectagent:Project.Project`

  o Notice that the class path is case sensitive. Failure in abiding with name case would result in jade failing to load the agent.

  o You can load whatever number of agents you like through adding agents path to the previous command while specifying their names like the following

    `C:\Jade-All-3.4\JadeStart.bat –gui projectagent1:Project.Project1 projectagent2:Project.Project2 projectagent3:Project.Project3`
- **Loading agents through RMA**

  It is also possible to load agents using the RMA tool. This method is more user friendly. This is done as follows

  1. Open the RMA using the previous command jadetsart.bat
  2. Expand the menu till the main container sub folder appears

  3. Then right click on the main container and choose start "new agent option"
After choosing start new agent button, a window named insert start parameters will appear that requires some input.

- For the agent name option insert any name like project
- For the Class name insert your class path for the agent like Project.Project
- Leave the rest of the parameters for the time being.
- After inserting the required parameters press ok.
- If the inserted parameters are not correct for some reason, a window will appear informing you that there is a problem with the agent like the following.
If the inserted parameters are correct the Jade environment will load the agent as follows.
5.4. Create more containers

Till this point we were able to create our first platform and out first main container. In the next section, we will learn how to create extra containers either local to the platform host or remote on another computer. In order to be able to add new containers we will have to use some command prompt commands. We have two cases; the first is if we want to add a container to the platform holding the main container. The second case is if we wanted to add a container to another remote computer and register it with the main container.

5.4.1. Add a container besides the main container

In this case, the host will hold two containers one is the main container holding the DF and the AMS and another container holding some agents. In order to the container do the following.

• The jade environment as mentioned previously identifies the agents through its nickname and its hosting computer. The hosting computer is usually identified by its name. For instance, my computer name is "sharlauc". Thus, an agent named "agent1" that runs on my system would have the following name "agent1@sharlauc:1099" where 1099 is the RMI port number. In order to change your computer name right click on my computer then click properties then choose the tab named computer name.

• After changing your computer name you should start your platform as mentioned earlier. However, in this case we will slightly change the command to the following

C:\Jade-all-3.4\jadestart.bat –gui –host <you computer name>

• The –host parameter when accompanied with –gui parameter determines the host name. Then we have our platform with the main container set.
A normal container is added through another command as follows:

```
C:\JADE-all-3.4\Jadestart.bat –host <computername> -container <agent>
```

In the previous command the –host parameter has a different terminology than the one used earlier. –host command in this case specifies to what platform the container should be added. The <computername> is the name of the computer that will host the container.

Moreover, the –container command specifies that the agent should run in a new container. The <agent> is the agent nickname followed by the agent class path. In my case, the command would look like this:

```
C:\JADE-all-3.4\Jadestart.bat –host sharlauc -container agent1:Project.Project
```

The result of running this command would look like the following. The container-3 is added to the platform and as we can see the agent is added to my computer which means that the agent is running on this computer.
5.4.2. **Add a container to a remote computer**

At the current stage we were able to start our platform, add the main container and another normal one. However, till this point we have not used the capacity of JADE. Jade was designed with distributed agent environment in mind. JADE enables different agents not only on different computers, bit on different domains as well to communicate. In this section we will learn how to start a distributed JADE environment. This can be done as follows:

- Start your platform with the main container as previously mentioned with the jadestart.bat command.
- Prepare the machine by following the instructions in the Install Jade section numbered 5.1.
- Go to the second machine and setup the machine name as done with the first one and lets call it "remote".
- Then we write the following command.
C:\JADE...\jadestart -host sharlauc –container agent1:Project.Project

- The previous command tells the JADE environment to start a new container that hold agent named "agent1" at machine "remote" with DHCP IP 10.0.0.2. Then instructs it to register that agent to machine "sharlauc" such that machine "remote" cannot control either the container no the agent. On the contrary, the "sharlauc" machine the one with the main container can control the containers with the agents running on it. This can be depicted as follows.

5.5. Handling containers

A word of advice, while you are testing your new agents, you'd better load them first in their new container. After making sure of the functionality of your agent you can add it to other populated containers. This is due to that once and agent class is added to JADE, it is not possible to overwrite the class with the new one. In this case, you will have to kill the container then reload it with the new agent code.
5.6. How to code an agent

As mentioned earlier, JADE contains set of packages that can be used to implement agents. These set of packages are divided into different categories each used to satisfy a certain function in your agent application.

5.6.1. Agent Lifecycle

![Agent Lifecycle Diagram]

- Initializations
  - Addition of initial behaviours
- Agent "life" (execution of behaviours)
  - Clean-up operations
5.6.2. Agent initialization

In order to start coding your agent you have to initialize your agent and prepare its basic structure. This is done through the sub package jade-core-Agent. This package includes some methods that are implemented by the developer. The most important methods in this class includes some methods that initialize an empty agent like setup() or instruct the agent to commit a certain action through AddBehavior(). The complete list of such methods can be found in the JADE API documentation.

5.6.3. Agent Behaviors

After the agent is initialized we need to instruct the agent to perform a certain action. In JADE this is called behavior. These behaviors are implemented inside a subclass called jade.core.behaviors. The most important methods inside this class that you need to know are action() that instruct the agent to run the behavior, the done() method which checks whether the behavior is still running or finished, and the block() method which instructs the agent to block the behavior. The most important things to know about behaviors is that JADE implemented different types of sub behaviors that differ in their execution and that might become beneficial in the course of developing your application. The different types of behavior packages are as follows.

5.6.3.1. The SimpleBehavior class

This is the simplest form of behaviors that execute once based upon the agent request and then terminates.

```java
public class MyOneShotBehaviour extends Behaviour {

    public void action() {
        //
    }
```
5.6.3.2. The CyclicBehavior class

This is different than the simple behavior in that it runs indefinitely unless interrupted by the agent itself.

```java
public class MyOneShotBehaviour extends CyclicBehaviour {
    public void action()
    {// }
}
```

5.6.3.3. The CompositeBehavior class

This behavior includes more than one child behaviors that are executed based upon the agent request.

```java
public class MyOneShotBehaviour extends CompositeBehaviour {
    public void action()
    {// switch statement }
}
```

5.6.3.4. The WakerBehavior class

This behavior is designed such that it executes whenever a specified time has elapsed.

```java
addBehaviour(new WakerBehaviour(this, 10000);
```

5.6.3.5. The TickerBehavior class

This behavior is designed such that it executes repetitively after specified amount of ticks that can be specified through the constructor.

```java
addBehaviour(new TickerBehaviour(this, 10000);
```

5.6.3.6. The SequentialBehavior class

This is very similar to the CompositeBehavior but for that it is designed to run number of child behaviors in sequential order.

For More information about the Behaviors please check the API documentation.
5.7. Agent Communication

This is done through two methods the first one is called dodelete() and the second one is called takedown(). The first one deletes the agent by setting its state to deleted state. The second one is for the user to implement some cleanup actions for the deleted agent.

5.8. Agent Termination

This is done through two methods the first one is called dodelete() and the second one is called takedown(). The first one deletes the agent by setting its state to deleted state. The second one is for the user to implement some cleanup actions for the deleted agent.