Distributed Artificial Intelligence and Intelligent Agents

Exercise : Introduction to JADE

Aims of this exercise

- To get knowledge about Java-based programming environments for MAS
- To learn the JADE environment as a possible tool for implementing the course project.
- To create awareness of the types of systems that are available and generate ideas about how such systems can be created

JADE Platform

- JADE (Java Agent Development Environment)
  - http://jade.tilab.com/
- Version 4.3.3 (current version)
- Simple registration needed on website

JADE

JADE is a middleware that facilitates the development of multi-agent systems. It includes

- A **runtime environment** where JADE agents can "live" and that must be active on a given host before one or more agents can be executed on that host.
- A **library** of classes that programmers have to/can use (directly or by specializing them) to develop their agents.
- A suite of **graphical tools** that allows administrating and monitoring the activity of running agents.

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

JADE overview

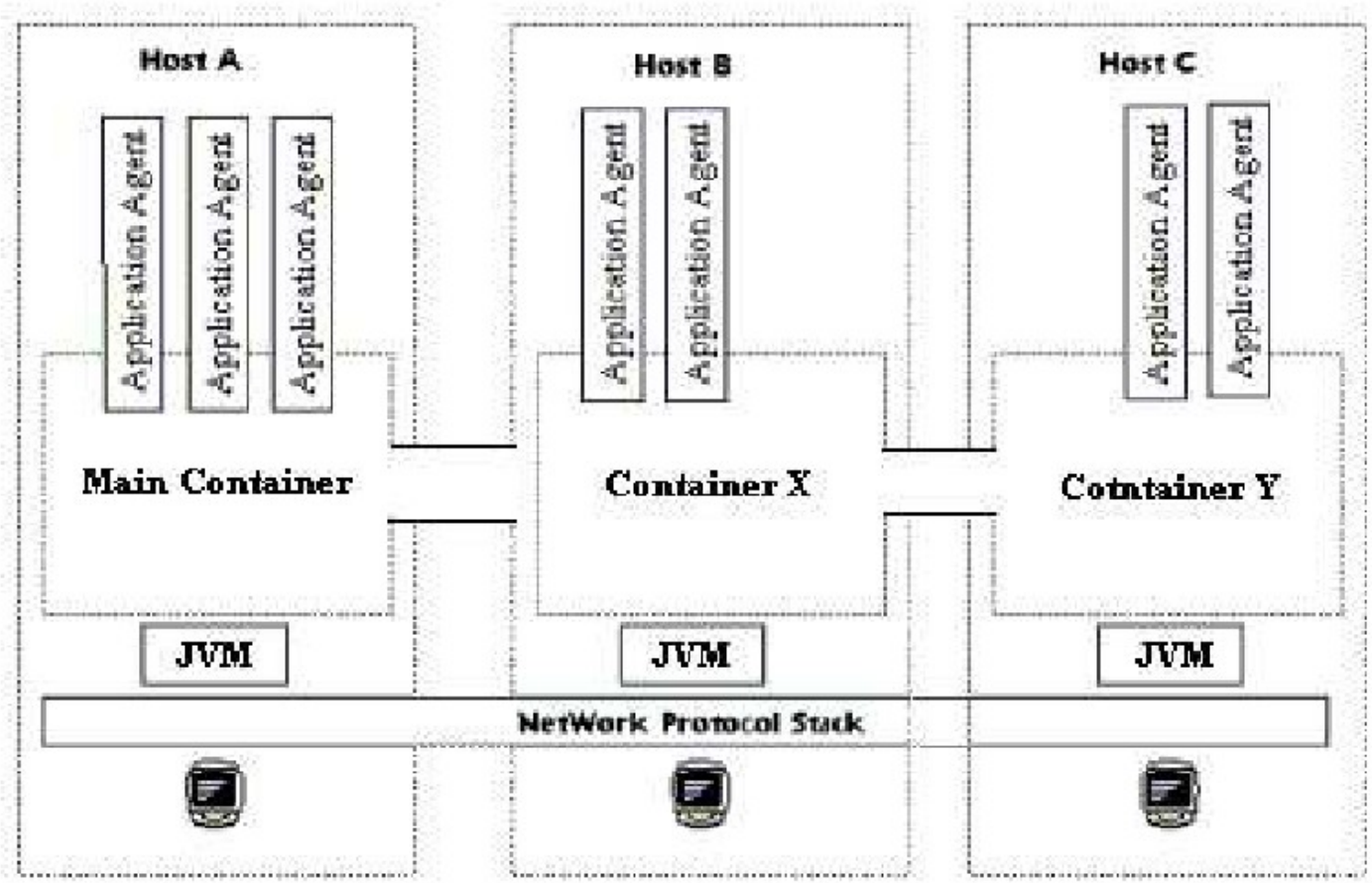
- ***Container*** - Each running instance of the JADE runtime environment (it can contain several agents).
- ***Platform*** - The set of active containers.
- A single special ***Main container*** must always be active in a platform and all other containers register with it as soon as they start.

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

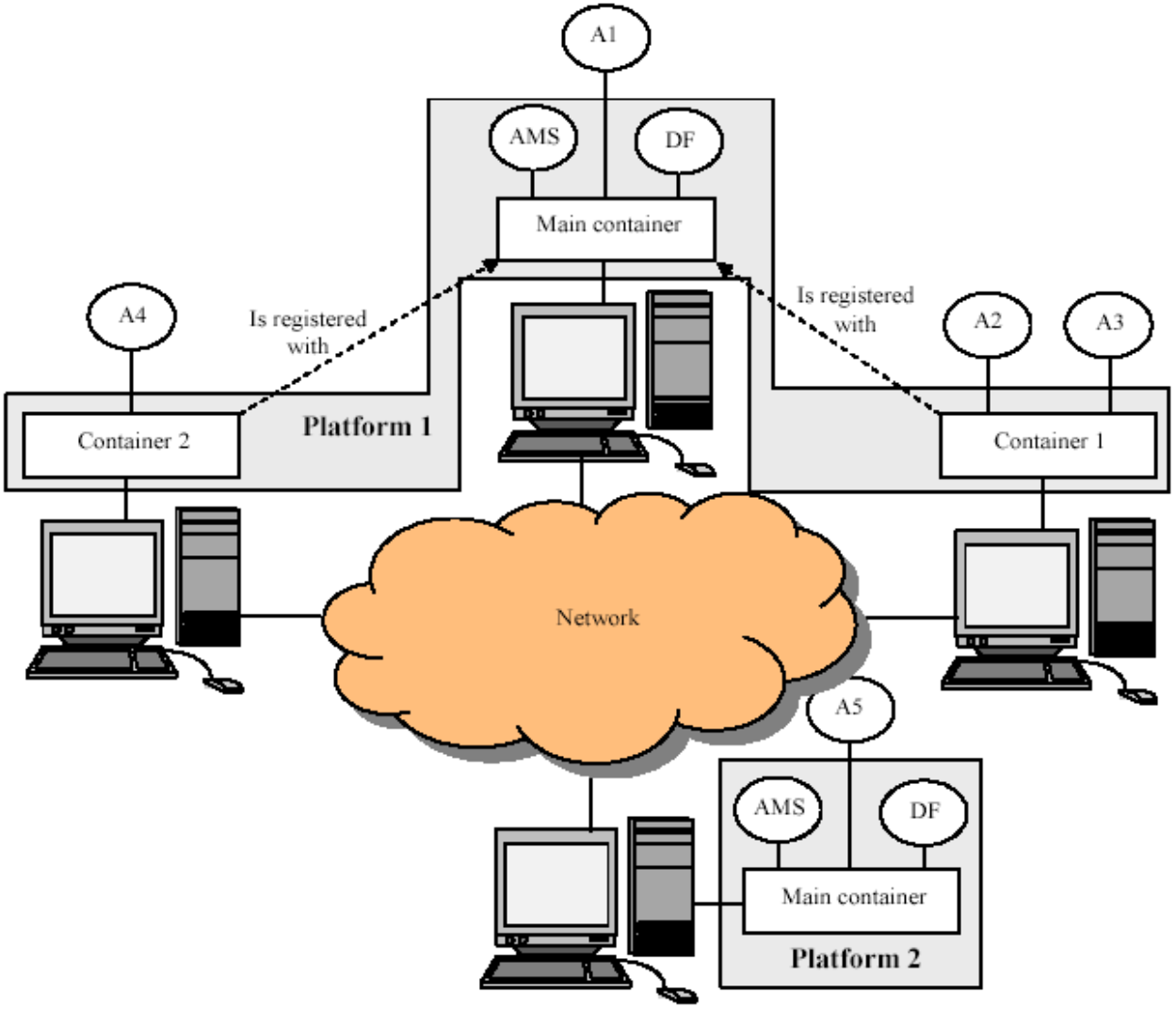
JADE overview

- Agent Management System (AMS) - white page service, the list of agents in the system
- Directory Facilitator (DF) - yellow page service, the list of service available from agents
- You can start, suspend and kill agents

JADE overview



JADE overview



From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Creating agents (BookTrading example)

```
package examples.bookTrading

import jade.core.Agent;
public class BookBuyerAgent extends Agent {
  protected void setup() {
     //  Printouta welcome message
          System.out.println(
               "Hallo! Buyeragent"
               + getAID().getName()
               +" is ready.");
     }
}
```

- The setup() method is intended to include agent initializations.
- The actual job an agent has to do is typically carried out within "behaviours"

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Agent identifiers

- Each agent is identified by an "agent identifier" represented as an instance of the jade.core.AID class.
- The **`getAID()`** method of the Agent class allows retrieving the agent identifier. An AID object includes a globally unique name plus a number of addresses.
- The name in JADE has the form <*nickname*>@<*platform-name*> so that an agent called *misha* living on a platform called *IMIT* will have

  misha@IMIT as globally unique name.

AID can be obtained knowing agent's nickname:

```
String nickname = "Peter";
AID id = newAID(nickname, AID.ISLOCALNAME);
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Running Agents

- Compilation

```
javac -classpath <JADE-classes>
    BookBuyerAgent.java
```

- Running

```
java -classpath <JADE-classes>;.
    jade.Boot buyer:BookBuyerAgent
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Passing arguments to an agent

```
java jade.Boot
     buyer:BookBuyerAgent(The-Lord-of-
     the-rings)
```

Passing arguments to an agent

```
import jade.core.Agent;
import jade.core.AID;
public class BookBuyerAgent extends Agent {
   // The title of the book to buy
private String targetBookTitle;
   // The list of known seller agents
private AID[] sellerAgents = {new AID("seller1", AID.ISLOCALNAME), new
   AID("seller2", AID.ISLOCALNAME)};
   // Put agent initializations here
protected void setup() {
   // Printout a welcome message
         System.out.println("Hallo! Buyer-agent "+getAID().getName()+" is ready");
    // Get the title of the book to buy as a start-up argument
    Object[] args = getArguments();
    if (args != null && args.length > 0) {
         targetBookTitle = (String) args[0];
         System.out.println("Trying to buy "+targetBookTitle);
   }
   else {
         // Make the agent terminate immediately
         System.out.println("No  book title specified");
         doDelete();
   }
}
   // Put agent clean-up operations here
protected void takeDown() {
   // Printout a dismissal message
   System.out.println("Buyer-agent "+getAID().getName()+" terminating.");
   }
}
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Behaviours

- The actual job an agent has to do is typically carried out within "behaviours".
- A behaviour represents a task that an agent can carry out and is implemented as an object of a class that extends **`jade.core.behaviours.Behaviour`**.
- Behaviour is added to agent by means of the **`addBehaviour()`** method of the Agent class.
- Behaviours can be added at any time
- Each class extending Behaviour must implement the **`action()`** method, that actually defines the operations to be performed when the behaviour is in execution and the **`done()`** method (returns a boolean value), that specifies whether or not a behaviour has completed and have to be removed from the pool of behaviours an agent is carrying out.
- An agent can execute several behaviours concurrently.
- When a behaviour is scheduled for execution its **`action()`** method is called and runs until it returns.

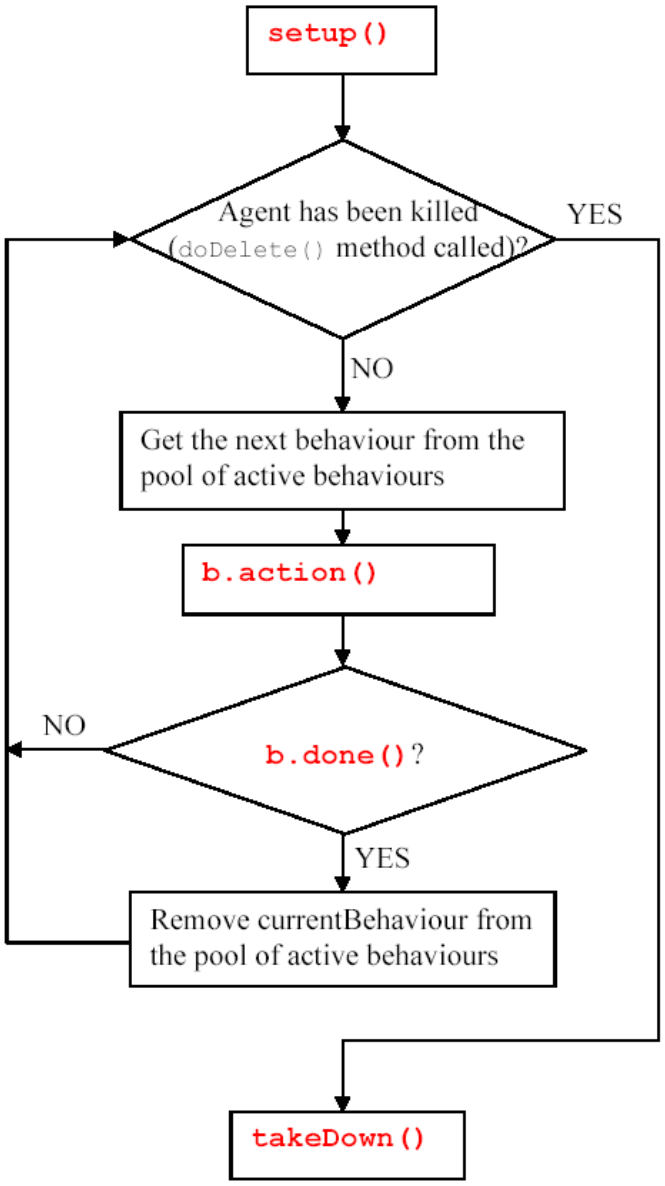
From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Behaviours

- Allows having a single Java thread per agent
- Provides better performances
- Eliminates all synchronization issues between concurrent behaviours accessing the same resources
- When a behaviour switch occurs the status of an agent does not include any stack information and is therefore possible to take a “snapshot” of it.

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Agents execution



- Initializations
- Addition of initial behaviours

Highlighted in red the methods that programmers have to implement

- Agent "life" (execution of behaviours)

- Clean-up operations

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Behaviour examples

This behaviour never stops

```
public class OverbearingBehaviour extends
                    Behaviour {
  public void action() {
      while (true) {
            // do something
      }
  }
  public boolean done() {
      return true;
  }
}
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Available Behaviours

- **`jade.core.behaviours.Behaviour`** - base class for all behaviours; needed only in the case of special synchronisation needs
- **`jade.core.behaviours.CompositeBehaviour`** - holds a number of children behaviours; this class must be extended to provide the actual scheduling policy to apply when running children behaviours
- **`jade.core.behaviours.CyclicBehaviour`** - atomic behaviour that is executed forever

Available Behaviours

- **`jade.core.behaviours.FSMBehaviour`** - CompositeBehaviour that executes its children behaviours according to a finite state machine (FSM) defined by the user

- **`jade.core.behaviours.OneShotBehaviour`** - atomic behaviour that executes just once

- **`jade.core.behaviours.ParallelBehaviour`** - CompositeBehaviour with concurrent children scheduling; it terminates when a particular condition on its sub-behaviours is met i.e. when all children are done, N children are done or any child is done

Available Behaviours

- **`jade.core.behaviours.ReceiverBehaviour`** - behaviour for receiving an ACL message; this behaviour terminates when an ACL message is received

- **`jade.core.behaviours.SenderBehaviour`** - behaviour for sending an ACL message; this behaviour sends a given ACL message and terminates

- **`jade.core.behaviours.SequentialBehaviour`** - CompositeBehaviour with sequential children scheduling; executes its children behaviours in sequential order, and terminates when its last child has ended

Available Behaviours

- **`jade.core.behaviours.SimpleBehaviour`** - atomic behaviour, which models behaviours that are made by a single, monolithic task and cannot be interrupted

- **`jade.core.behaviours.TickerBehaviour`** - behaviour that periodically executes a user-defined piece of code

- **`jade.core.behaviours.WakerBehaviour`** - **`OneShotBehaviour`** that is executed only once just after a given timeout is elapsed

Creating Behaviours

"One-shot" behaviour

```
public class MyOneShotBehaviour extends
       OneShotBehaviour{
   public void action() {
        // perform operation X
   }
}
```

"Cyclic"

```
public class MyCyclicBehaviour extends CyclicBehaviour{
   public void action() {
        // perform operation Y
   }
}
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Creating Behaviours

- Generic behaviour

```
public class MyThreeStepBehaviour extends Behaviour {
   private int step = 0;
   public void action() {
        switch (step) {
        case 0:
                // perform operation X
                step++;
                break;
        case 1:
                // perform operation Y
                step++;
                break;
        case 2:
                // perform operation Z
                step++;
                break;
        }
   }
   public boolean done()
        {  return step ==
        3;
   }
}
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Adding behaviours

```
public class MyAgent extends Agent {
   protected void setup() {
       System.out.println("Adding waker behaviour");
       addBehaviour(new WakerBehaviour(this, 10000) {
             protected void handleElapsedTimeout() {
                   // perform operation X
             }
       } );
   }
}
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Adding behaviours

```
public class MyAgent extends Agent {
  protected void setup() {
      addBehaviour(
            new TickerBehaviour(this, 10000){
            protected void onTick() {
                  // perform operation Y
            }
      } );
  }
}
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Book-buyer agent behaviours

```
protected void setup() {
   // Printout a welcome message
   System.out.println("Hallo! Buyer-agent
        "+getAID().getName()+" is ready.");
   // Get the title of the book to buy as a start-up argument
   Object[] args = getArguments();
   if (args != null && args.length > 0) {
        targetBookTitle = (String) args[0];
        System.out.println("Trying to buy "+targetBookTitle);
   // Add a TickerBehaviour that schedules a request to seller
   // agents every minute
   addBehaviour(new TickerBehaviour(this, 60000)
        {  protected void onTick() {
              myAgent.addBehaviour(new RequestPerformer());
        }
   } );
}
else {
   // Make the agent terminate
   System.out.println("No target book title specified");
   doDelete();
   }
}
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

Book-seller agent behaviours

```
import jade.core.Agent;
import jade.core.behaviours.*;
import java.util.*;
public class BookSellerAgent extends Agent {
   // The catalogue of books for sale (maps the title of a book
   to //its price)
   private Hashtable catalogue;
   // The GUI by means of which the user can add books in the
   //catalogue
   private BookSellerGui myGui;
   // Put agent initializations here
   protected void setup()
        // Create {  the
        catalogue catalogue
        // Create = new Hashtable();
        myGui = new BookSellerGui(this);
        and show the GUI
        myGui.show();
        // Add the behaviour serving requests for offer from
   buyer       //agents
        addBehaviour(new OfferRequestsServer());
        // Add the behaviour serving purchase orders from buyer
        //agents
        addBehaviour(new PurchaseOrdersServer());
   }
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

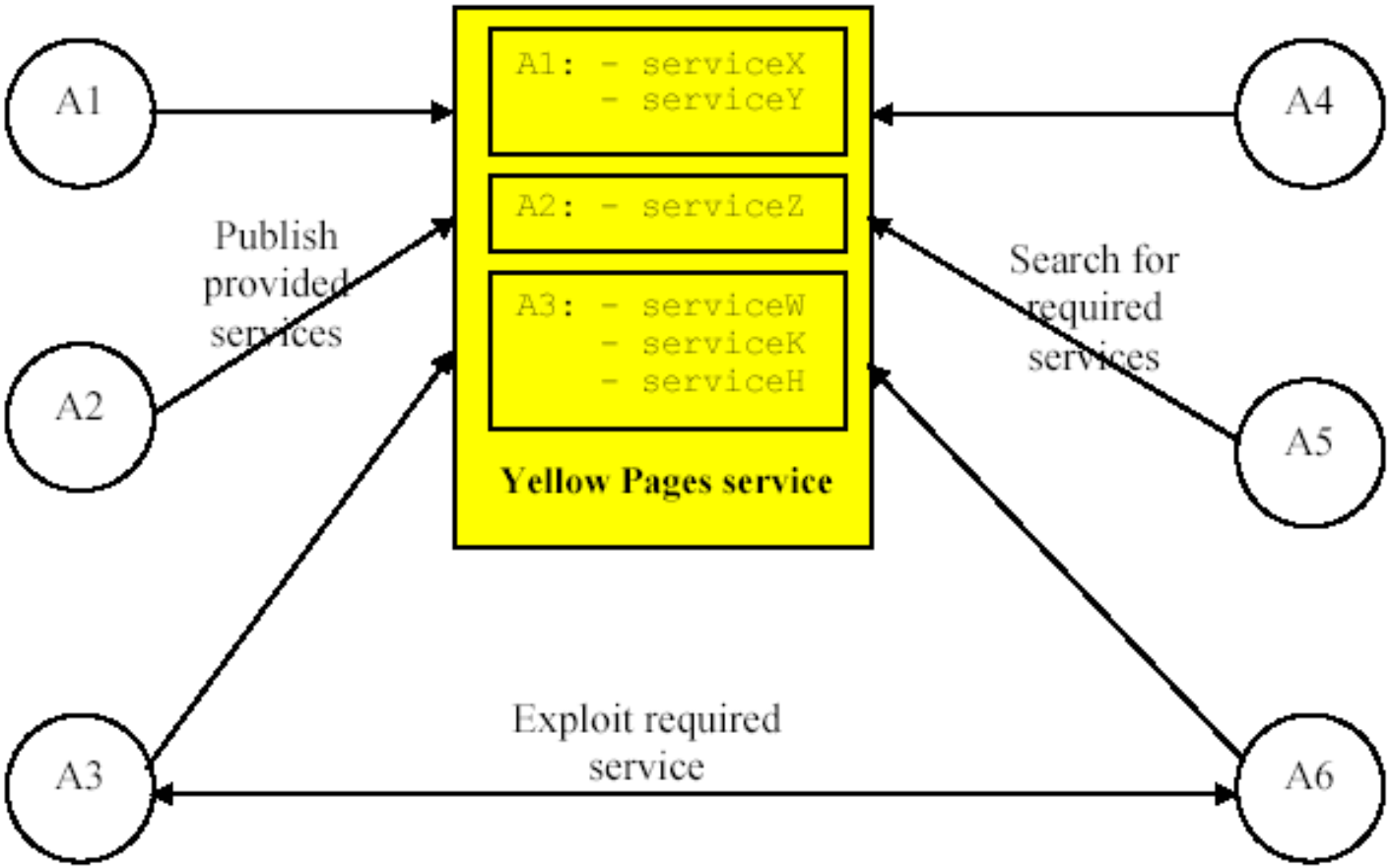
Book-seller agent behaviours

```
   // Put agent clean-up operations here
   protected void takeDown() {
        // Close the GUI
        myGui.dispose();
        // Printout a dismissal message
        System.out.println("Seller-agent
                "+getAID().getName()+" terminating.");
   }

/**
This is invoked by the GUI when the user adds a new book for sale
*/
   public void updateCatalogue(final String title, final int price){
        addBehaviour(new OneShotBehaviour() {
                public void action() {
                        catalogue.put(title, new Integer(price));
                }
        } );
   }
}
```

From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

THE DF SERVICE CLASS



From **JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS**
Author: Giovanni Caire (TILAB, formerly CSELT)

DFAgentDescription

```
DFAgentDescription
  Name: AID // Required for registration
  Protocols: set of Strings
  Ontologies: set of Strings
  Languages: set of Strings
  Services: set of {
      { Name: String // Required for each
                     //service specified
      Type: String   // Required ...
      Owner: String Protocols: set of Strings
      Ontologies: set of Strings
      Languages: set of Strings
      Properties: set of
            { Name: String
              bValue: String
            }
      }
```

Interacting with the DF (Publishing services)

```
protected void setup() {
   ...
   // Register the book-selling service in the yellow pages
   DFAgentDescription dfd = new DFAgentDescription();
   dfd.setName(getAID());
   ServiceDescription sd = new ServiceDescription();
   sd.setType("book-selling");
   sd.setName("JADE-book-trading");
   dfd.addServices(sd);
   try {
       DFService.register(this, dfd);
   }
   catch (FIPAException fe) {
       fe.printStackTrace();
   }
   ...
}
```

This simple example and we do not specify any language, ontology or service-specific property.

Interacting with the DF (Publishing services)

```
void register( ServiceDescription sd)
{ DFAgentDescription dfd = new
     DFAgentDescription();
  dfd.setName(getAID());
  dfd.addServices(sd);
  try
                                    dfd );
  }  {  DFService.register(th
  catch(FIPAException fe) {
  fe.printStackTrace(); }
}
```

Usage:

```
ServiceDescription sd = new
     ServiceDescription();
sd.setType( "buyer" );
sd.setName( getLocalName() );
register( sd );
```

Interacting with the DF (de-registrating)

```
protected void takeDown() {
  // Deregister from the yellow pages
  try {
     DFService.deregister(this);
  }
  catch (FIPAException fe)
     {  fe.printStackTrace()
     ;
  }
  //  Close  theGUI
     myGui.dispose();
  //   Printouta dismissal message
     System.out.println("Seller-agent
     "+getAID().getName()+"terminating.");
}
```

Interacting with the DF (Searching for services)

```
public class BookBuyerAgent extends Agent
    {   private String targetBookTitle; // The title of the book to buy
    private AID[] sellerAgents;          // The list of known seller agents
    // Put agent initializations here
    protected void setup() {
          // Printout a welcome message
          System.out.println("Hallo! Buyer-agent "+getAID().getName()+" is ready.");
          // Get the title of the book to buy as a start-up argument
          Object[] args = getArguments();
          if (args != null && args.length > 0) {
               targetBookTitle = (String) args[0];
               System.out.println("Trying to buy "+targetBookTitle);
               // Add a TickerBehaviour that schedules a request to seller agents every minute
               addBehaviour(new TickerBehaviour(this, 60000) {
                    protected void onTick() {
                         // Update the list of seller agents
                         DFAgentDescription template = new DFAgentDescription();
                         ServiceDescription sd       = new ServiceDescription();
                         sd.setType("book-selling");
                         template.addServices(sd);
                         try {
                              DFAgentDescription[] result = DFService.search(myAgent,template);
                              sellerAgents = new AID[result.length];
                              for (int i = 0; i < result.length; ++i)
                                        {  sellerAgents[i] =
                                        result.getName();
                              }
                         }
                         catch (FIPAException fe)
                              {  fe.printStackTrace
                              ();
                         }
                         // Perform the request
                         myAgent.addBehaviour(new RequestPerformer());
                    }
```

DF Utilities

```
AID getService( String service )
   {
       DFAgentDescription dfd = new
              DFAgentDescription();
       ServiceDescription sd = new
              ServiceDescription();
       sd.setType(service );
       dfd.addServices(sd);
       try
       {
              DFAgentDescription[] result =
                     DFService.search(this, dfd);
                     if (result.length>0) return
                            result[0].getName() ;
       }
       catch (FIPAException fe) { fe.printStackTrace();}
       return null;
}
```

DF Utilities

```
AID [] searchDF( String service )
{
   DFAgentDescription dfd = new DFAgentDescription();
   ServiceDescription  sd = new ServiceDescription();
   sd.setType( service );
   dfd.addServices(sd);
   SearchConstraints ALL = new SearchConstraints();
   ALL.setMaxResults(new Long(-1));
   try
   {
       DFAgentDescription[] result =
              DFService.search(this, dfd, ALL);
       AID[] agents = new AID[result.length];
       for (i=0; i<result.length; i++)
              agents[i] = result[i].getName() );
       return agents;
   }
   catch (FIPAException fe) { fe.printStackTrace(); }
   return null;
}
```