- The Java Agent DEvelopment framework -

Principles and main functionalities

Cédric Herpson

Maître de conférences en intelligence artificielle

cedric.herpson@lip6.fr

@herpsonc

SCIENCES SORBONNE UNIVERSITÉ

LIP6

cnrs

https://jade-project.gitlab.io/

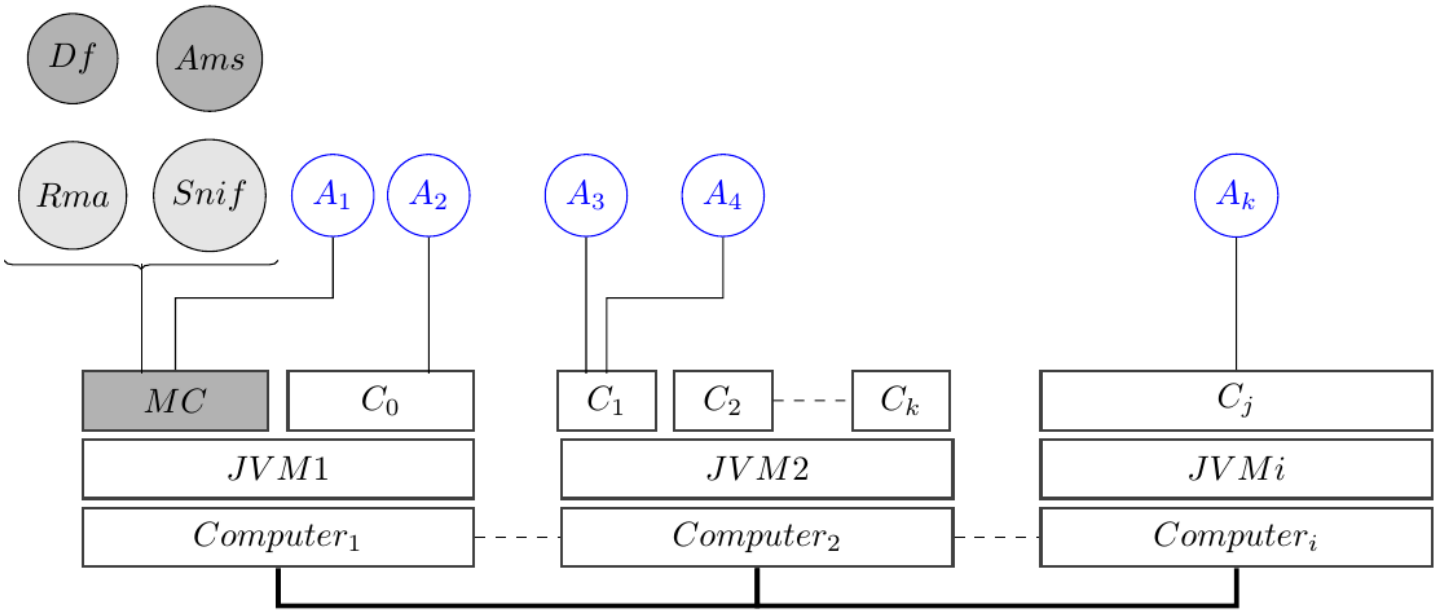
- A real distributed system
- Computation asynchronism
- Communication asynchronism
- FIPA compliant
- Support agent mobility
- Work on Android devices

- LGPL
- Initially backed by Telecom Italia
- Used in the Industry

Extensive documentation

- Programmer's guide
- Programming tutorial
- Book : Developing MAS with JADE (Wiley)

Jade's components



Mandatory ● DF : Directory Facilitator (Yellow page) ● AMS : Agent Management System (Platform control)
Optional ● RMA : Remote Management Agent (Gui for AMS) ● Sniffer : Catch messages on the fly (Debug)
Optional ● Ai : Your agents

1 Agent

- 1 Thread
- 1 unique identifier, the AID : AgentLocalName@platformName
- 1 mailbox
- K behaviours (add/suppress them dynamically)

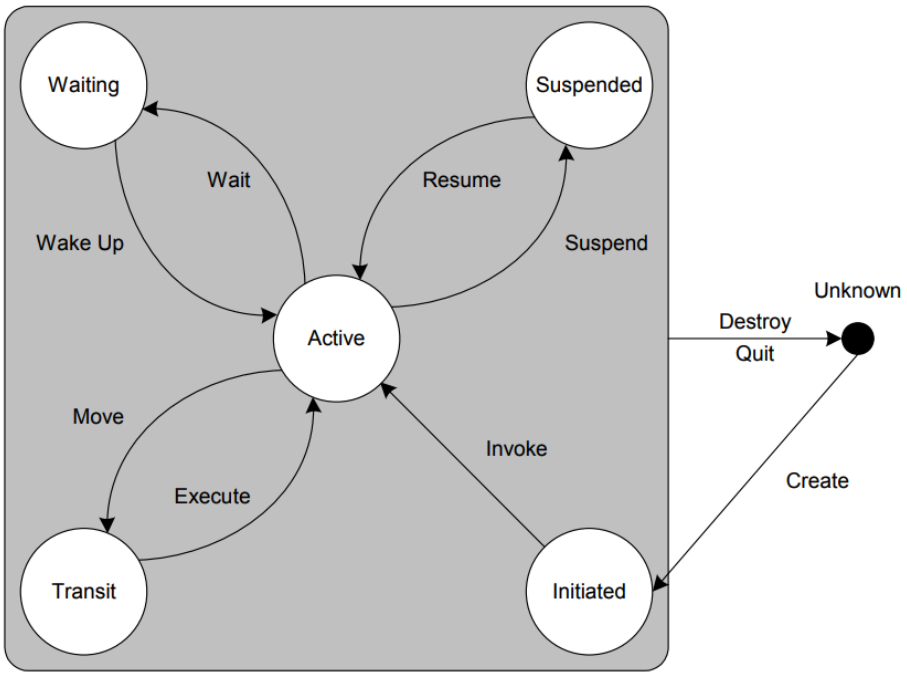
1 Agent

- 1 Thread
- 1 unique identifier, the AID : AgentLocalName@platformName
- 1 mailbox
- K behaviours (add/suppress them dynamically)

```
MyAgent extends Agent {

  //Automatically called when MyAgent created
  public void setup(){
    // variables Init + behaviours
    addBehaviour(new BehaviourA());
  }

  //Automatically called when MyAgent killed
  public void takeDown(){ }
}
```

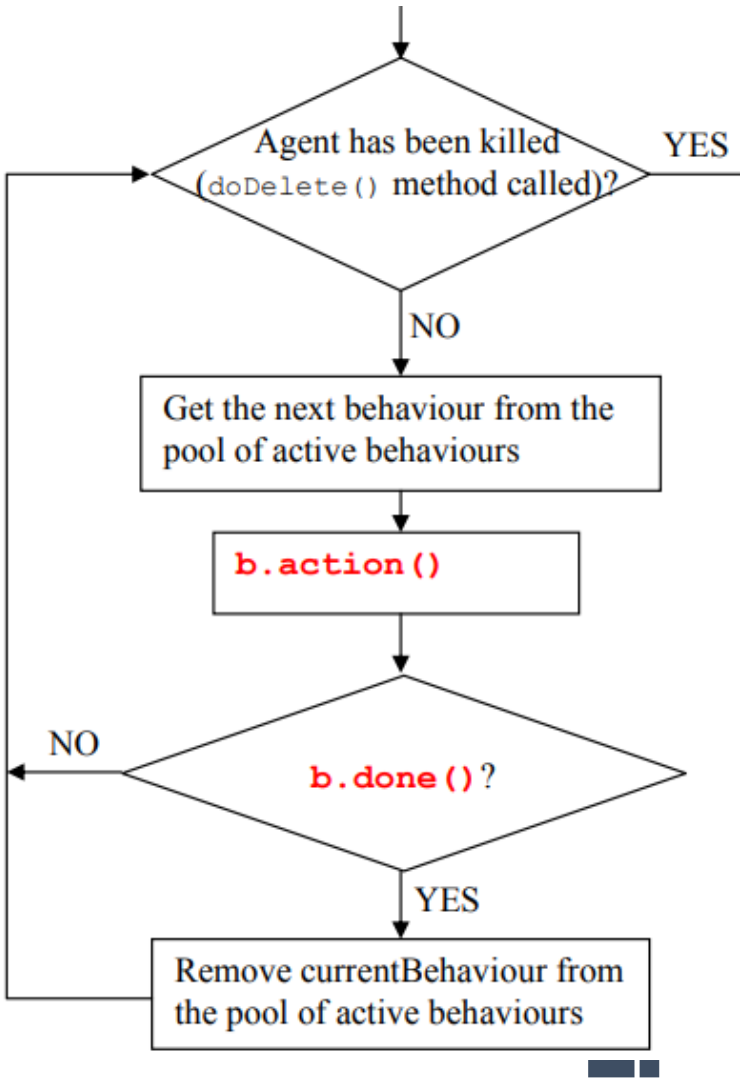
```
BehaviourA extends SimpleBehaviour {
  boolean b=false;
  public BeA(Agent a){
    // Init
  }
  //called if triggerable and choosen
  public void action(){
    //* Critical section *
    printf("A Little Nightmare Music");
  }

  //called after each action() call
  public boolean done(){
    return b;
  }
}
```

Agent's life's cycle



- Active : Agent computing.
- Waiting : Agent sleeping. Will wake up when a message is received
- Transit : Migration between two computers
- Initiated (not visible)
- Suspended (not visible)

Agent's abilities : Behaviours life's cycle



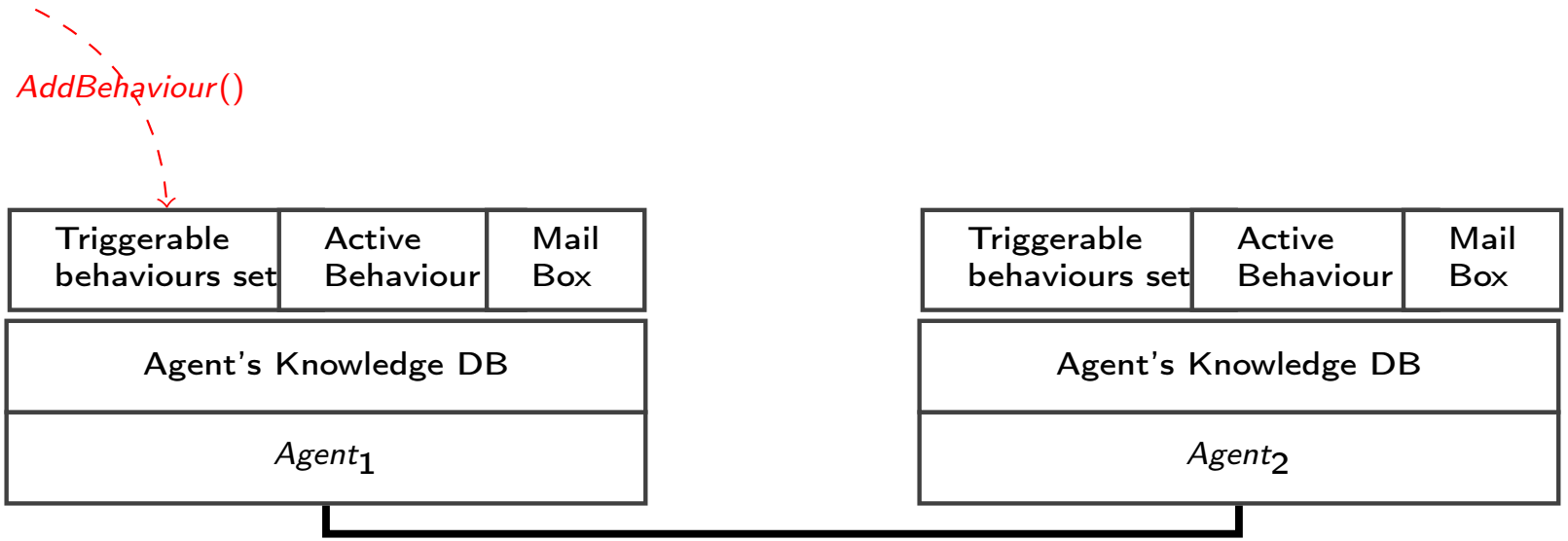
⚠ Behaviour scheduling is non-preemptive

⚠ action() is a critical-section ⚠

If needed, see *ThreadedBehaviourFactory()* for blocking operations

Asynchronous communication

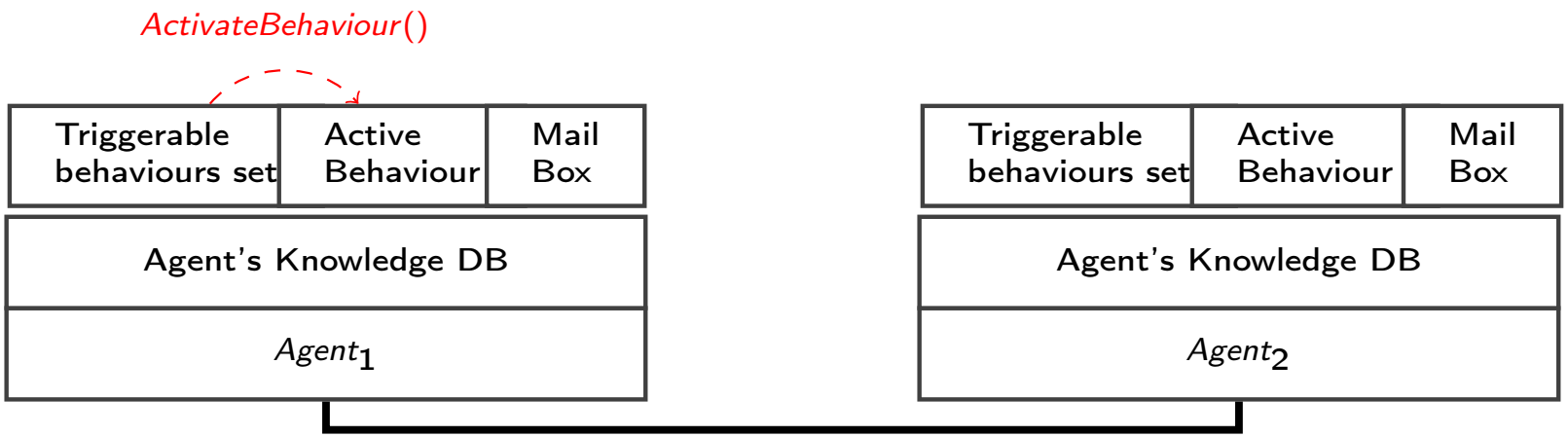
Reception ≠ Delivery



1. AddBehaviour
2. Activate & execute
3. Send(msg)
4. Transmission
5. Reception
6. Activate & execute
7. Delivery

Asynchronous communication

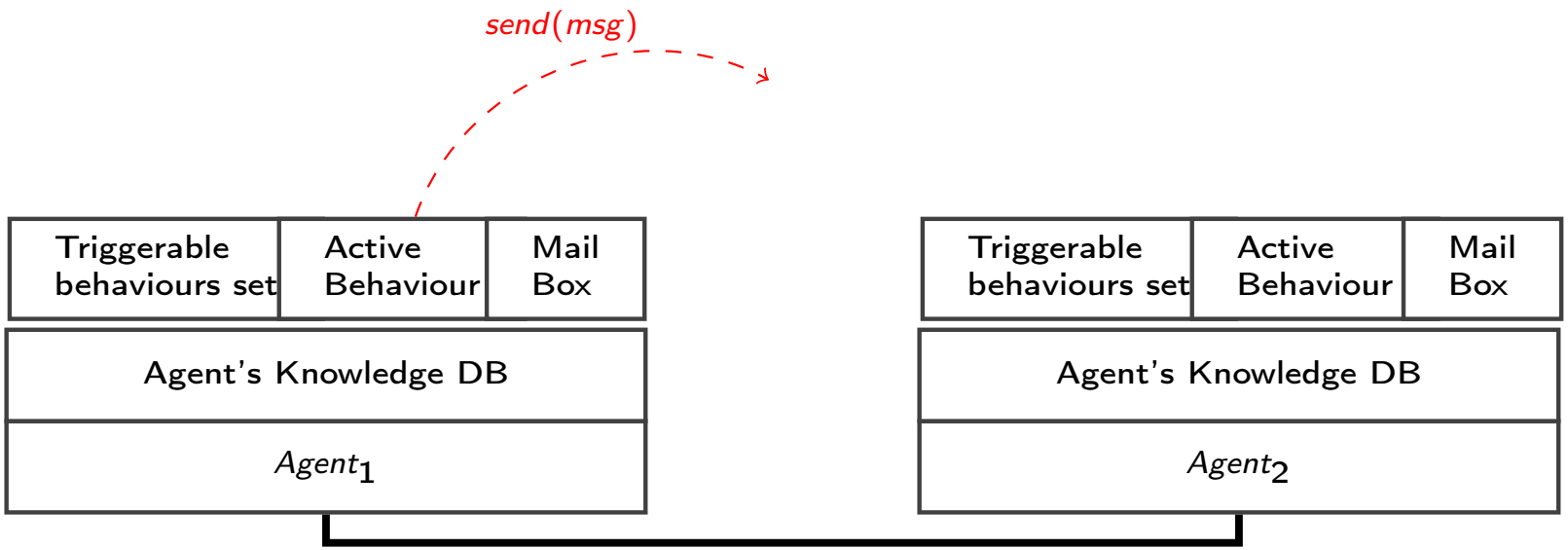
Reception ≠ Delivery



1. AddBehaviour
2. Activate & execute
3. Send(msg)
4. Transmission
5. Reception
6. Activate & execute
7. Delivery

Asynchronous communication

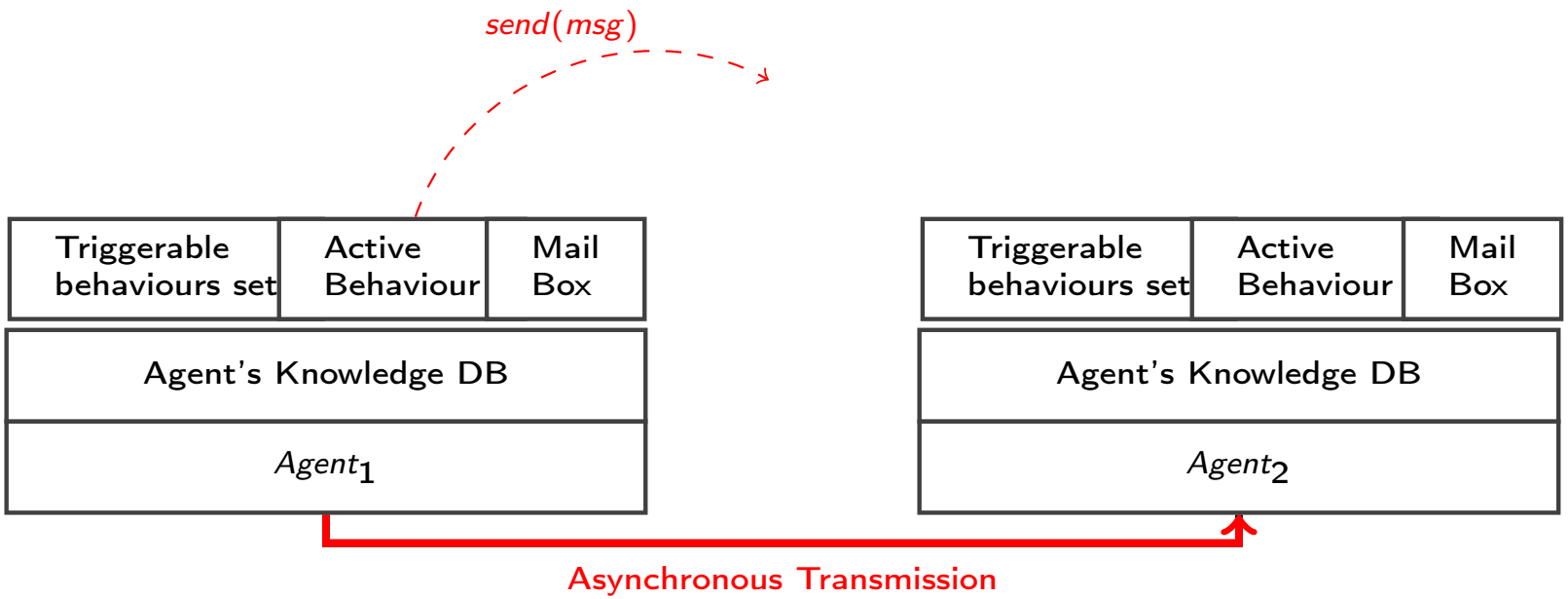
Reception ≠ Delivery



1. AddBehaviour
2. Activate & execute
3. Send(msg)
4. Transmission
5. Reception
6. Activate & execute
7. Delivery

Asynchronous communication

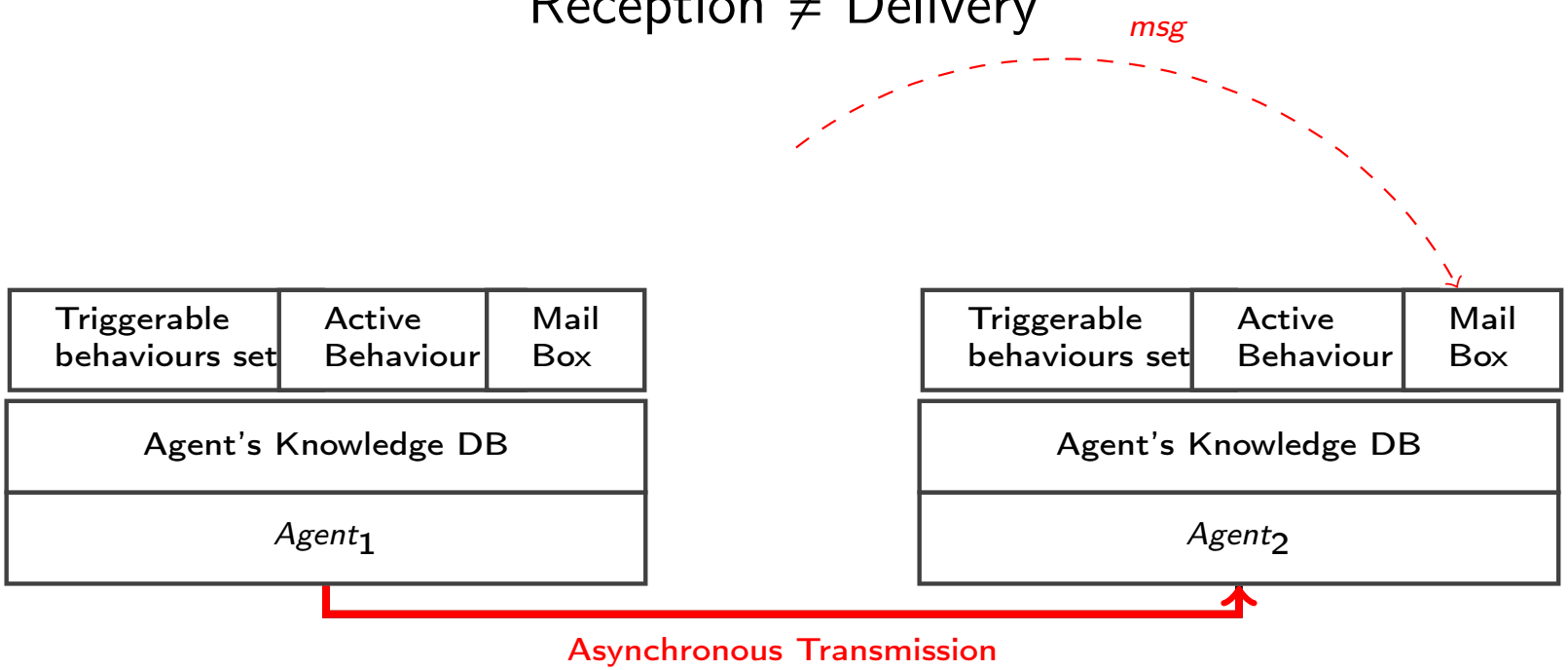
Reception ≠ Delivery



1. AddBehaviour
2. Activate & execute
3. Send(msg)
4. Transmission
5. Reception
6. Activate & execute
7. Delivery

Asynchronous communication

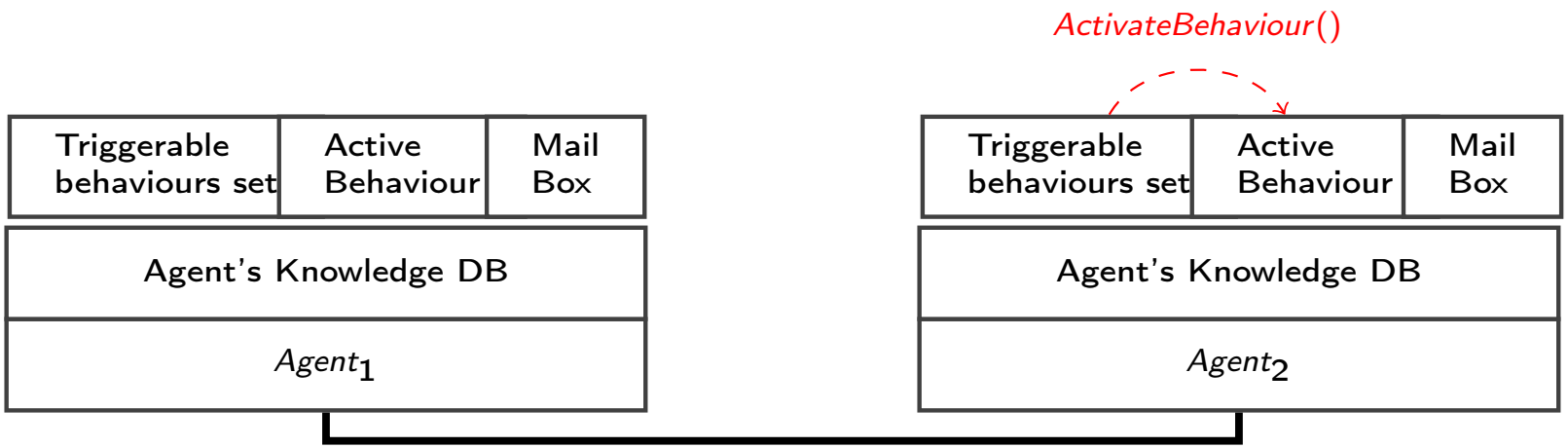
Reception ≠ Delivery



1. AddBehaviour
2. Activate & execute
3. Send(msg)
4. Transmission
5. Reception
6. Activate & execute
7. Delivery

Asynchronous communication

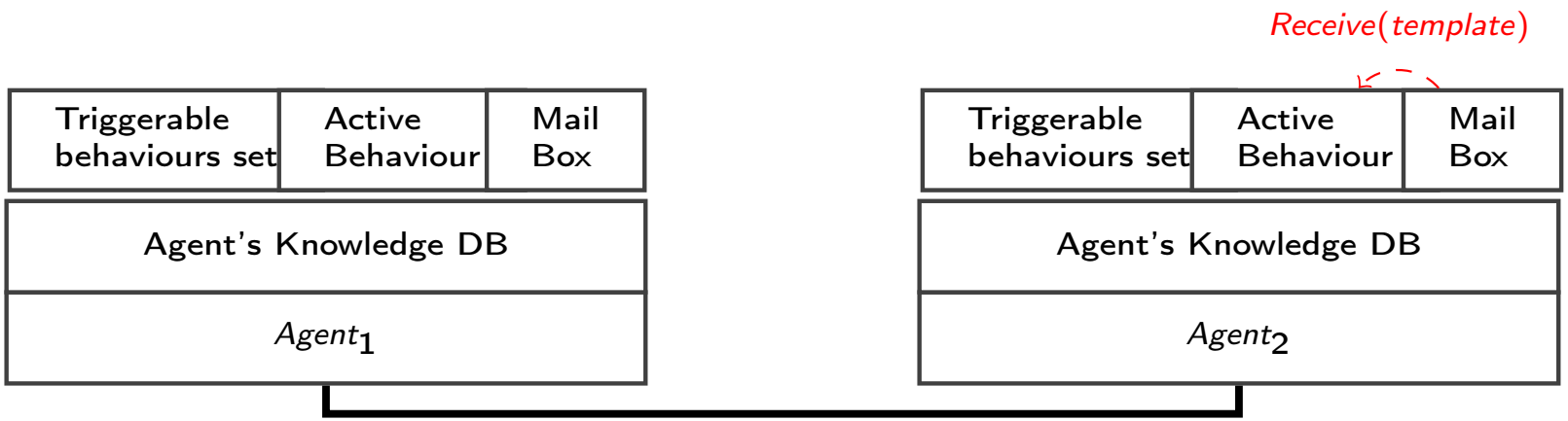
Reception ≠ Delivery



1. AddBehaviour
2. Activate & execute
3. Send(msg)
4. Transmission
5. Reception
6. Activate & execute
7. Delivery

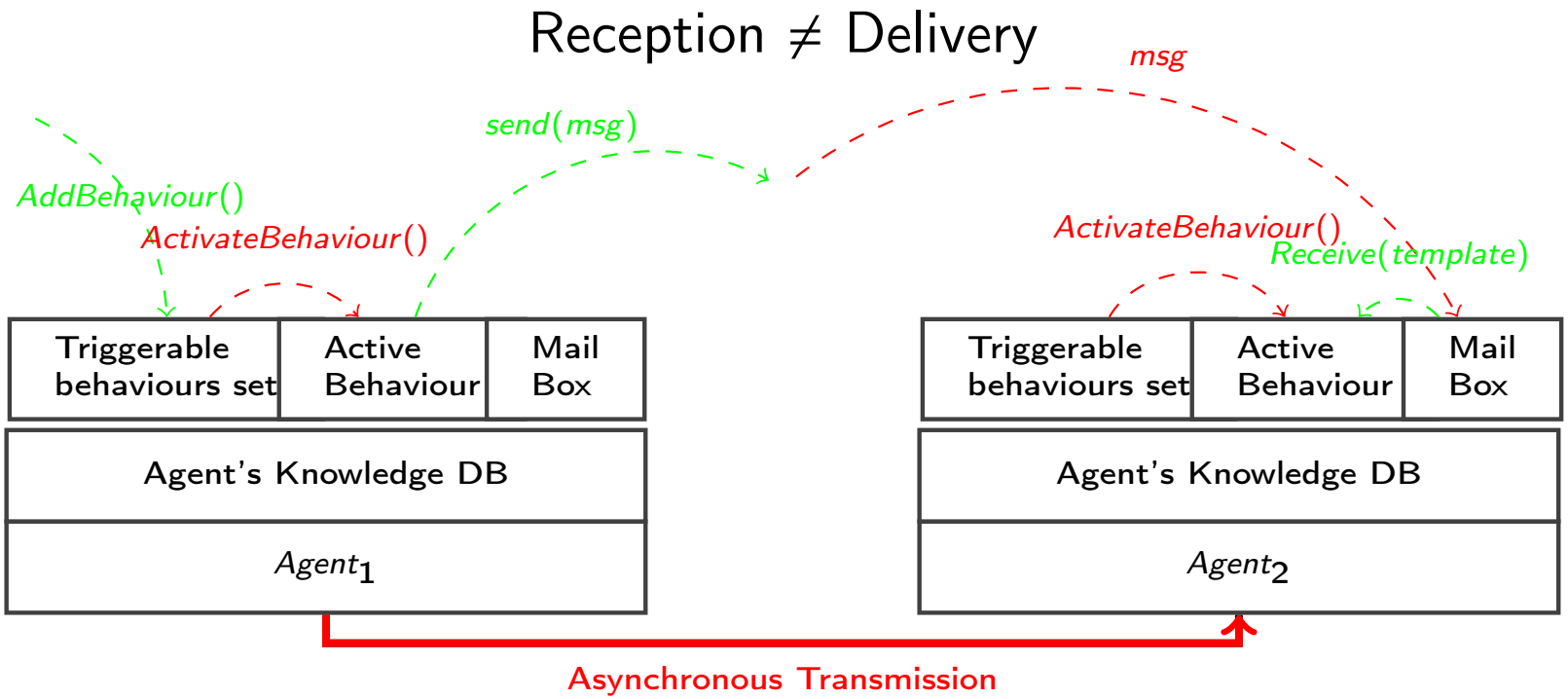
Asynchronous communication

Reception ≠ Delivery



1. AddBehaviour
2. Activate & execute
3. Send(msg)
4. Transmission
5. Reception
6. Activate & execute
7. Delivery

Asynchronous communication



1. AddBehaviour
2. Activate & execute
3. Send(msg)
4. Transmission
5. Reception
6. Activate & execute
7. Delivery

Communication

Send a message

```
ACLMessage msg=new
 ACLMessage(ACLMessage.INFORM);//FIPA
msg.setSender( this .myAgent.getAID());
msg.setProtocol(" UselessProtocol ");
msg.setContent("Hello World");
msg.setContentObject(/ Serializable
        object /);
msg.addReceiver(new
 AID("ReceiverName",AID.ISLOCALNAME));
msg.addReceiver(new
 AID("ReceiverName2",AID.ISLOCALNAME));
send(msg);
```

Receive a message

```
MessageTemplate template= MessageTemplate.and(
 MessageTemplate.MatchProtocol("UselessProtocol"),
  MessageTemplate.or(
   MessageTemplate.MatchPerformative(ACLMessage.INFORM)
   MessageTemplate.MatchPerformative(ACLMessage.REFUSE)
  )
 );
 // takes the message if the template is  verified
 ACLMessage msg=this.myAgent.receive(template);
if  (msg!=null){
  // Processing of the message
  String  textMessage=msg.getContent()
  Myobject o = (Myobject) msg.getContentObject();
} else{
 block();//the behaviour goes to sleep,  until  next
        msg
}
```

- Sleeping behaviours are woke up when a new message arrives
- You do not control the waking order. The computer do.

List of the agents on the platform (using AMS)

```
/**
     *
     * @return The list of the (local)names of the agents currently within the platform
     */
    private List<String> getAgentsList(){
        AMSAgentDescription [] agentsDescriptionCatalog = null;
        List<String> agentsNames= new ArrayList<String>();

        try {
            SearchConstraints c = new SearchConstraints();
            c.setMaxResults ( new Long(-1) );
            agentsDescriptionCatalog = AMSService.search(this.myAgent, new
                    AMSAgentDescription (), c );
        }
        catch (Exception e) {
            System.out.println( "Problem searching AMS: " + e );
            e.printStackTrace();
        }

        for (int i=0; i<agentsDescriptionCatalog.length;i++){
            AID agentID = agentsDescriptionCatalog[i].getName();
            agentsNames.add(agentID.getLocalName());
        }
        return agentsNames;
    }
```

Start-Jade (1/2)

How to start the Jade Multi-agent platform from source code ?

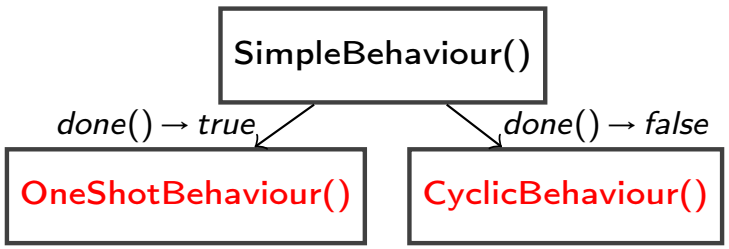
https://startjade.gitlab.io/

Classical behaviours

SimpleBehaviour()

```
BehaviourA extends SimpleBehaviour {
    boolean b=false;
    public BeA(Agent a){
      // Init
    }
    //called if triggerable and choosen
    public void action(){
      //* Critical section *
       printf ("A Little Nightmare Music");
    }

    //called after each action() call
    public boolean done(){
      return b;
    }
}
```

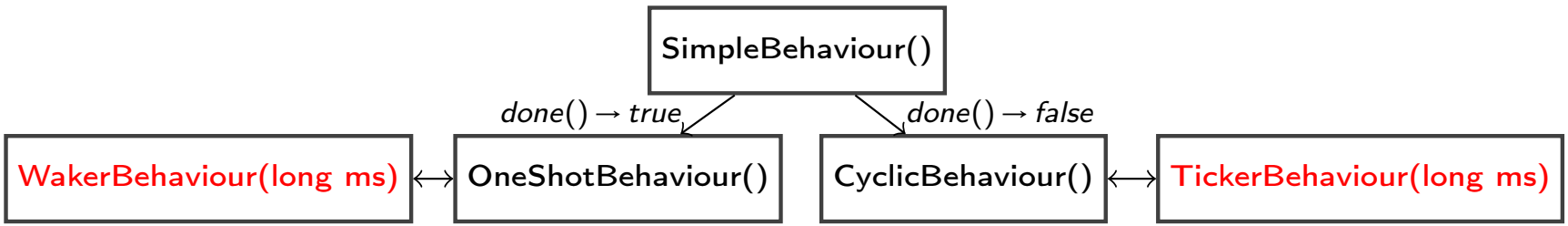
Classical behaviours




```
BehaviourA extends OneShotBehaviour {

  public BeA(Agent a){
    // Init
  }

  //Called only once. Behaviour then removed
  public void action(){
    //* Critical section *
    printf("A Little Nightmare Music");
  }
}
```

```
BehaviourA extends CyclicBehaviour {

  public BeA(Agent a){
    // Init
  }

  // Called an infinity of times
  // Release the proc after each call
  public void action(){
    //* Critical section *
    printf("A Little Nightmare Music");
  }
```

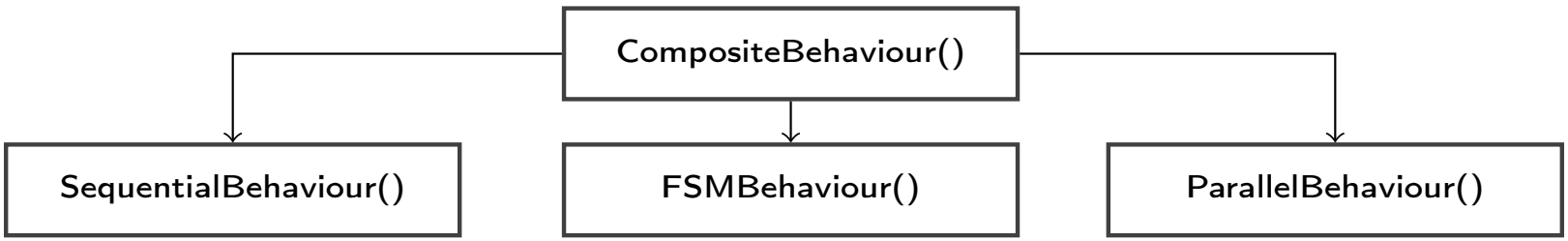
Classical behaviours



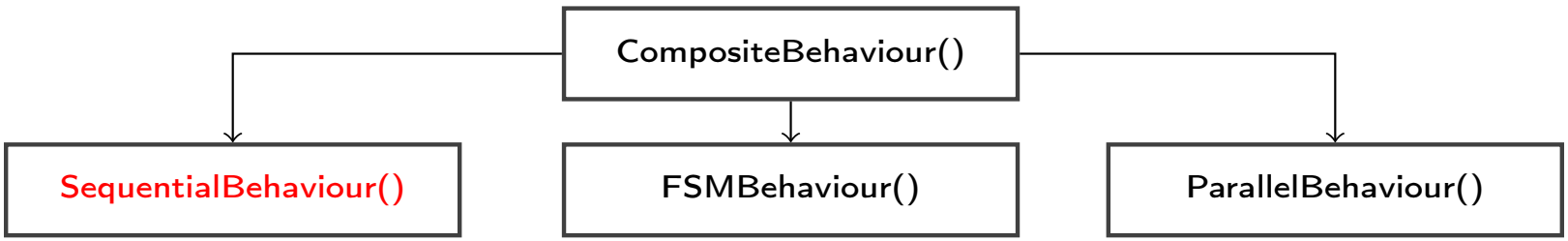

```
BehaviourA extends WakerBehaviour {

  public BeA(Agent a,long waitingTime){
    super(a,waitingTime);
  }

  // Called after waitingTime ms
  public void action(){
    //* Critical section *
    printf("A Little Nightmare Music");
  }
}
```

```
BehaviourA extends TickerBehaviour {

  /* period in ms */
  public BeA(Agent a, long period){
    super(a,period);
  }

  //Behaviour triggerable every period
  public void onTick(){
    //* Critical section *
    printf("A Little Nightmare Music");
  }
}
```

Advanced behaviours : *Composite*

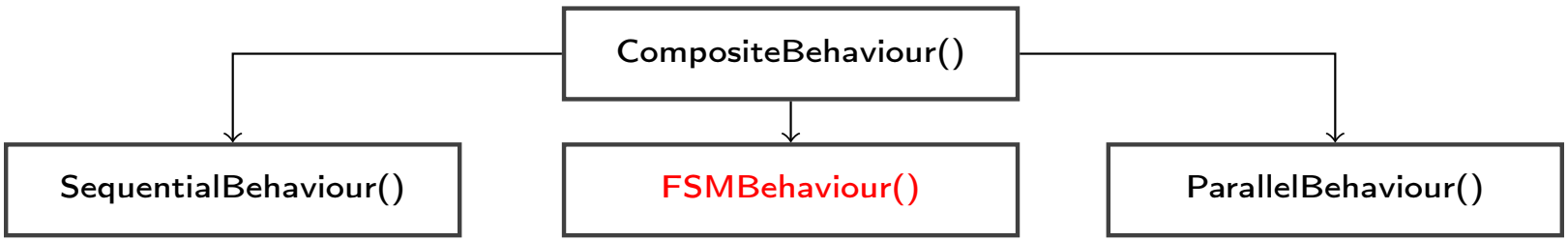


Advanced behaviours : *Sequential*

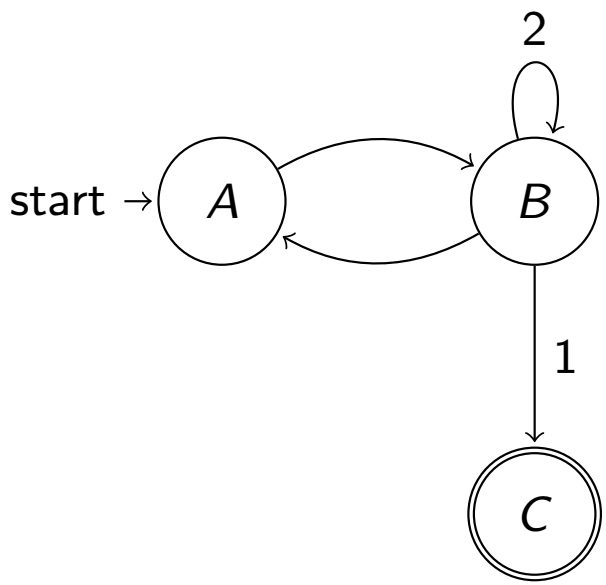



```
public class AgentA extends Agent {
    protected void setup() {
    SequentialBehaviour myBehaviour1 = new
            SequentialBehaviour(this);

    myBehaviour1.addSubBehaviour(new
            SingleStepBehaviour(this, "1.1"));
    myBehaviour1.addSubBehaviour(new
            SingleStepBehaviour(this, "1.2"));

    SequentialBehaviour myBehaviour2 = new
            SequentialBehaviour(this);
    SequentialBehaviour myBehaviour2_1 =
            new SequentialBehaviour(this);

    myBehaviour2_1.addSubBehaviour(new
            SingleStepBehaviour(this, "2.1.1"));
    myBehaviour2_1.addSubBehaviour(new
            SingleStepBehaviour(this, "2.1.2"));
    myBehaviour2.addSubBehaviour(myBehaviour2_1);
    myBehaviour2.addSubBehaviour(new
            SingleStepBehaviour(this, "2.2"));
    myBehaviour2.addSubBehaviour(new
            SingleStepBehaviour(this, "2.3"));

    addBehaviour(myBehaviour1);
    addBehaviour(myBehaviour2);
  }
}
class SingleStepBehaviour extends
        OneShotBehaviour {
  private String myStep;

  public SingleStepBehaviour(Agent a, String
          step) {
    super(a);myStep = step;
  }
  public void action() {
    System.out.println("Step "+myStep);
  }
}
```

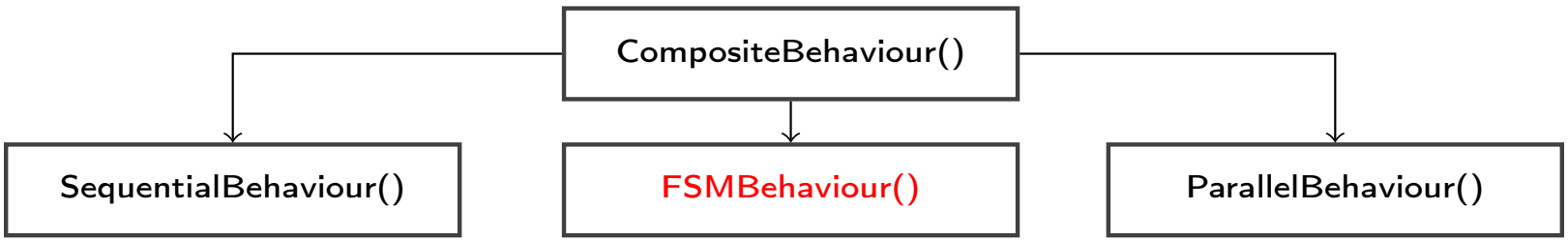
Advanced behaviours : *FSM*




```
public class AgentFsm extends Agent {
 // State names
  private static final String A = "A";
  private static final String B = "B";
  private static final String C = "C";
  protected void setup() {
   FSMBehaviour fsm = new FSMBehaviour(this);
   // Define the different states and behaviours
   fsm. registerFirstState (new StateBeha2(), A);
   fsm. registerState (new StateBeha(5), B);
   fsm. registerLastState (new StateBeha2(), C);
   // Register the transitions
   fsm. registerDefaultTransition (A,B);//Default
   fsm. registerDefaultTransition (B,A);//Default
   fsm. registerTransition (B,B, 2);//Cond 2
   fsm. registerTransition (B,C, 1);//Cond 1
   addBehaviour(fsm);
  }
}
```



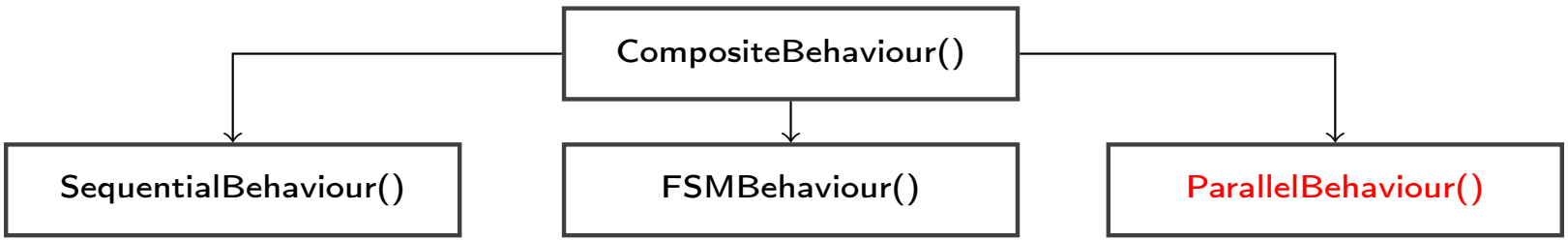
Advanced behaviours : *FSM*




```
public class AgentFsm extends Agent {
 // State names
 private static final String A = "A";
 private static final String B = "B";
 private static final String C = "C";
 protected void setup() {
  FSMBehaviour fsm = new FSMBehaviour(this);
  // Define the different states and behaviours
  fsm. registerFirstState (new StateBeha2(), A);
  fsm. registerState (new StateBeha(5), B);
  fsm. registerLastState (new StateBeha2(), C);
  // Register the transitions
  fsm. registerDefaultTransition (A,B);//Default
  fsm. registerDefaultTransition (B,A);//Default
  fsm. registerTransition (B,B, 2);//Cond 2
  fsm. registerTransition (B,C, 1);//Cond 1
  addBehaviour(fsm);
 }
}
```

```
public class StateBeha extends
      OneShotBehaviour {
   private int exitValue;
   public StateBeha(int max) {
    super(); exitValue = max;
   }
   public void action() {
    exitValue = (int) (Math.random() *
          exitValue);
    System.out. println ("Val : "+exitValue);
   }
  public int onEnd() {return exitValue;}
}
public class StateBeha2 extends
      OneShotBehaviour {
 public StateBeha2() { super();}
 public void action() {
  System.out. println ("Start/end behav");
 }
}
```

Advanced behaviours : *Parallel*



A misleading name

It does **NOT** allow to execute different behaviour in different threads.

- There is still only one thread that control the agent's behaviours.
- This is a behaviour composed of a set of subbehaviours trigerable *at the same time*.
- The main behaviour can be instructed to terminate when :
  - ALL of its sub-behaviours have completed or,
  - ANY sub-behaviour completes

real Parallel behaviours are possibles

```
Public class AgentMultithread extends Agent{
    private ThreadedBehaviourFactory tbf;

  protected void setup() {
     //0) Create the thread factory
     tbf= new ThreadedBehaviourFactory();
     // 1) You create the wanted behaviours
     Behaviour myBehaviour= new
           OneShotBehaviour(this) {
        public void action() {
           for (int i=1;i<100000;i++) {
              System.out.println(i);
           }}};

     //2) You add it in its own thread.
     this.addBehaviour(tbf.wrap(myBehaviour));
  }
```

Agent's abilities and services

Expose agents abilities as (web) services
Call (web) services

Locally : The Yellow page agent (DF)

Offering agent

```
DFAgentDescription dfd = new
        DFAgentDescription();
dfd.setName(getAID()); // The agent AID
 ServiceDescription  sd = new
        ServiceDescription();
sd.setType( "SUM" ); // You have to give a
        name to each service your agent offers
sd.setName(getLocalName());//(local)name of
        the agent
dfd.addServices(sd);

//Register the service
try {
  DFService.register(this, dfd );
} catch (FIPAException fe) {
        fe.printStackTrace(); }
```

Agent wishing to use a service

```
DFAgentDescription dfd = new
        DFAgentDescription();
 ServiceDescription  sd = new
        ServiceDescription();
sd.setType( "SUM" ); // name of the service
dfd.addServices(sd);
DFAgentDescription[] result =
        DFService.search(this, dfd);

//You get the list of all the agents (AID)
        offering this service
System.out.println(result.length + "
        results" );
if (result.length>0)
 System.out.println(result[0].getName());
```

Remotely : Agent's abilities use web-services

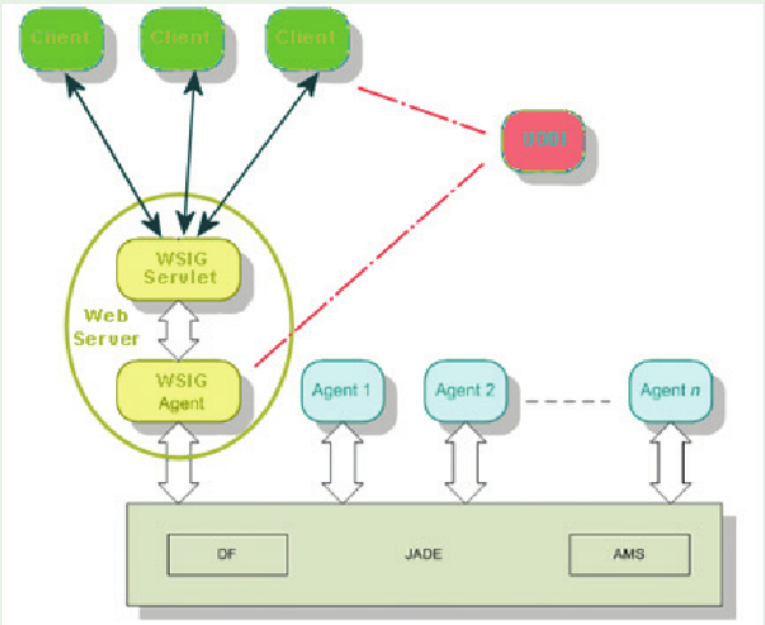
Use standard REST or SOAP (SAAJ) libraries

- REST : Functionalities seen as resources, URI
  → HTTP, simple keywords (post/get/put/delete)
  → Point to point connection, easy to learn, lightweight

- WS-* : Functionalities seen as services, Yellow-page like
  → Web Services Description Language (WSDL) and SOAP
  → Distribution friendly, complex, error management

Remotely : Agent's abilities as web-services

Use Jade's WSIG plugin (SOAP and REST)



→ WSIG agent will be the interface with external clients

→ WSIG agent will expose the services known by the DF

Mobile agents

Intra-platform or Inter-platforms

Intra-platform mobility

```
MyAgentA extends Agent{
  protected void setup(){
   super.setup();
   addBehaviour(new BeA(this));
  }
  protected void beforeMove(){
   super.beforeMove();
   // e.g. kill Gui
  }
  protected void AfterMove(){
   super.afterMove();
   addBehaviour(
   new BehaviourX());
   //e.g. restart Gui
  }}
```

```
BeA extends OneShotBehaviour {
  public BehaviourA(Agent a){
    // Init
  }

  public void action(){
   // Whatever your agent shoud do, then :
   ContainerID cID= new ContainerID();
   cID.setName("AnotherContainer1"); //Destination container
   cID.setAddress(ip); //IP of the host of the container
   cID.setPort(port); //port associated with Jade
   this.myAgent.doMove(cID);// LAST method to call in a
         behaviour, ALWAYS
  }
}
```

The doMove() execution will trigger :

1. Agent state to switch from active to transit (see slide 5)
2. Call of beforeMove() –> release here any local resource used, and store what is necessary
3. Serialization of the agent and its behaviours on the origin container
4. Deserialization of the agent and its behaviours on the targeted container
5. Call of afterMove() –> set here the state in which the agent should resume, and addbehaviourS()
6. Reception of the messages (if any), and the agent to resume

Start-Jade (2/2)

How can one agent migrate between containers (computers) ?

https://startjade.gitlab.io/

Inter-platform mobility

- Not natively supported by JADE
- Possible using the IPMS add-on

  Developed by the Universita Autonoma de Barcelona

  → Download IPMS 1.5 here

```
AID remoteAMS = new AID("ams@remotePlatform",
    AID.ISGUID);
remoteAMS.addAddresses("http://remotePlatformaddr:7778/acc");
PlatformID destination = new PlatformID(remoteAMS);
myAgent.doMove(destination);
```

The Dedale project

https://dedale.gitlab.io/

The Dedale project

https://dedale.gitlab.io/

Create your team of AI agents.

Make them collectively explore, hunt, collect treasures and survive.

Too easy ?

We'll see..