



Software agent computing

*1st laboratory activities
at Warsaw University of Technology*

Maciej Gawinecki

Systems Research Institute, Polish Academy of Sciences

maciej.gawinecki@ibspan.waw.pl

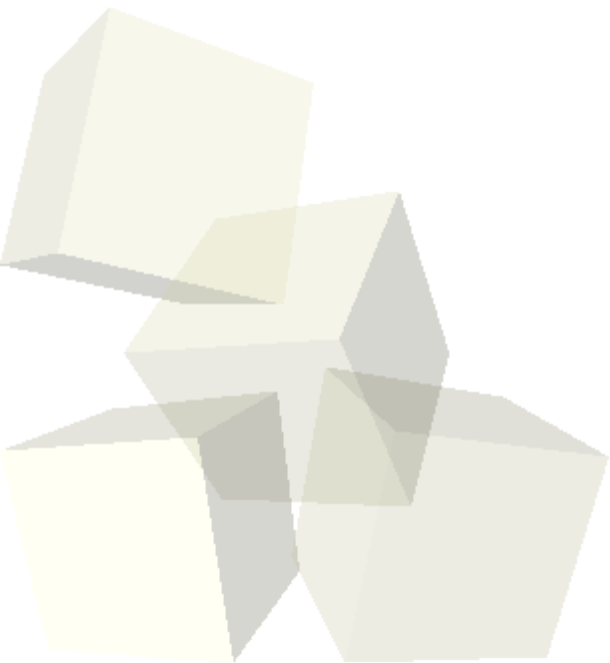
<http://www.ibspan.waw.pl/~gawinec>



4 h



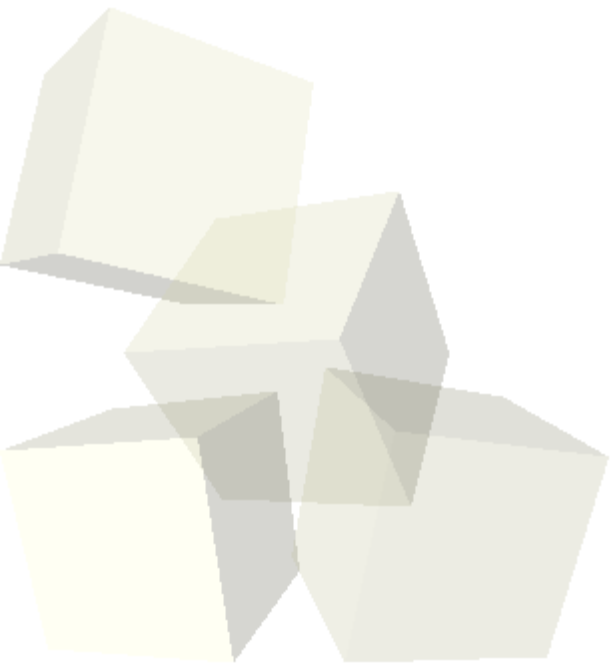
Part I: Motivation and aim





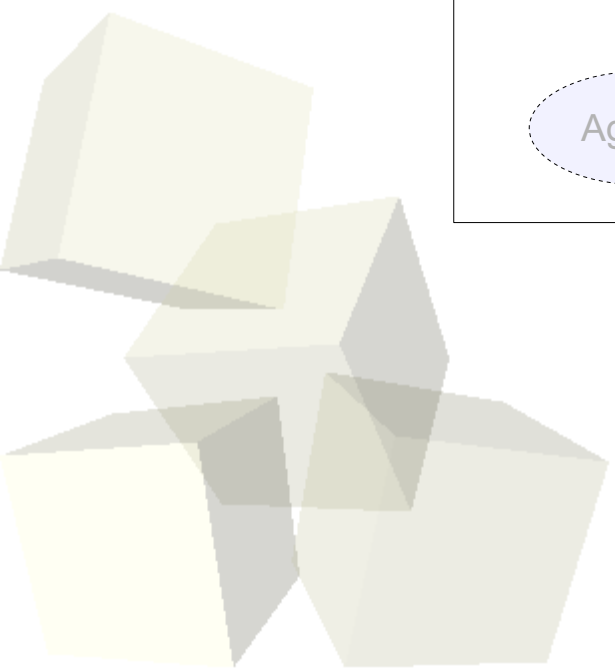
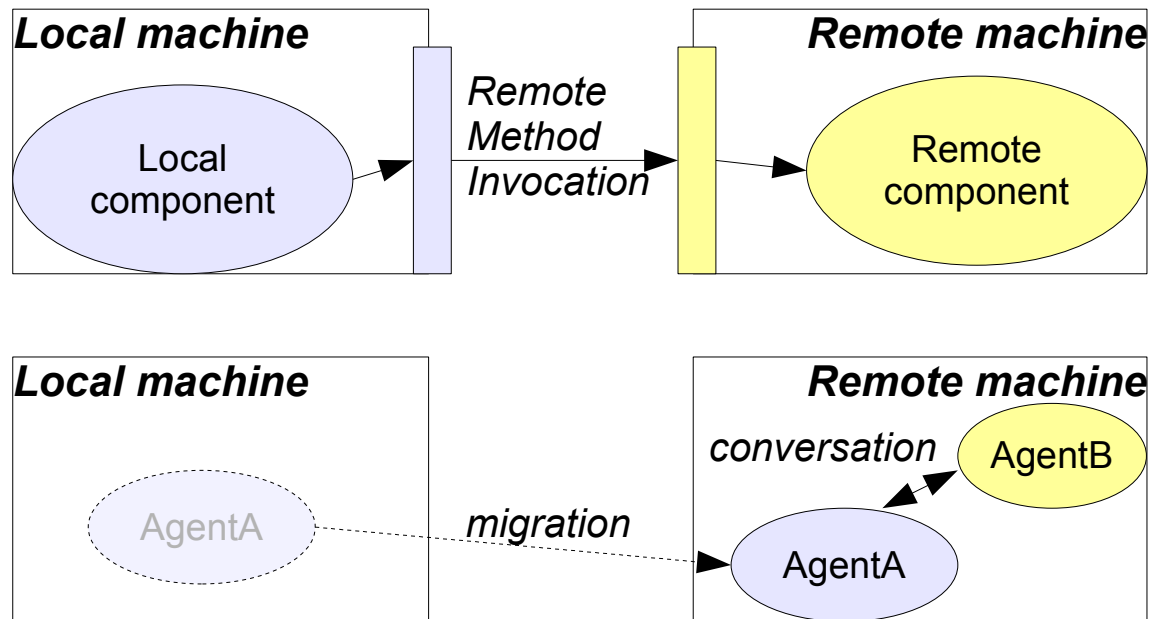
Aim of this laboratory

- Capture basic knowledge about programming in JADE
- Avoid confusing with vast specifactions
- Build up real agent-based application





Migration vs. RPC (RMI)





Mobile agents in logistics

Gives possibility to*:

- increase the **cost efficiency of the communication** between the logistical objects, e.g. as representatives of:
 - ✓ vehicles,
 - ✓ distribution centres,
 - ✓ packages
 - ✓ and other components
- enable new service solutions that have a need for **higher amounts of data to be transmitted**, e.g:
 - ✓ sensor data surveillance,
 - ✓ enriched route planning services

*Becker, M.; Singh, G.; Wenning, B.-L.; Görg, C.: *On Mobile Agents for Autonomous Logistics: An Analysis of Mobile Agents considering the Fan Out and sundry Strategies*. In: *International Journal of Services Operations and Informatics*, 1 (2007)



Migration – when use

*“In some cases agents need to **migrate** in order to accomplish their assigned tasks, as it would otherwise be impossible to transmit all the data needed for a specific task due to the **limitations imposed by the underlying communication network**. ”**

Example:

- ✓ **wireless networks**, which tend to have :
 - low and variable throughput,
 - high latency,
 - highly variable delays
 - and in some cases long connection establishment times.

*Becker, M.; Singh, G.; Wenning, B.-L.; Görg, C.: *On Mobile Agents for Autonomous Logistics: An Analysis of Mobile Agents considering the Fan Out and sundry Strategies*. In: *International Journal of Services Operations and Informatics*, 1 (2007)



Migration – when NOT use

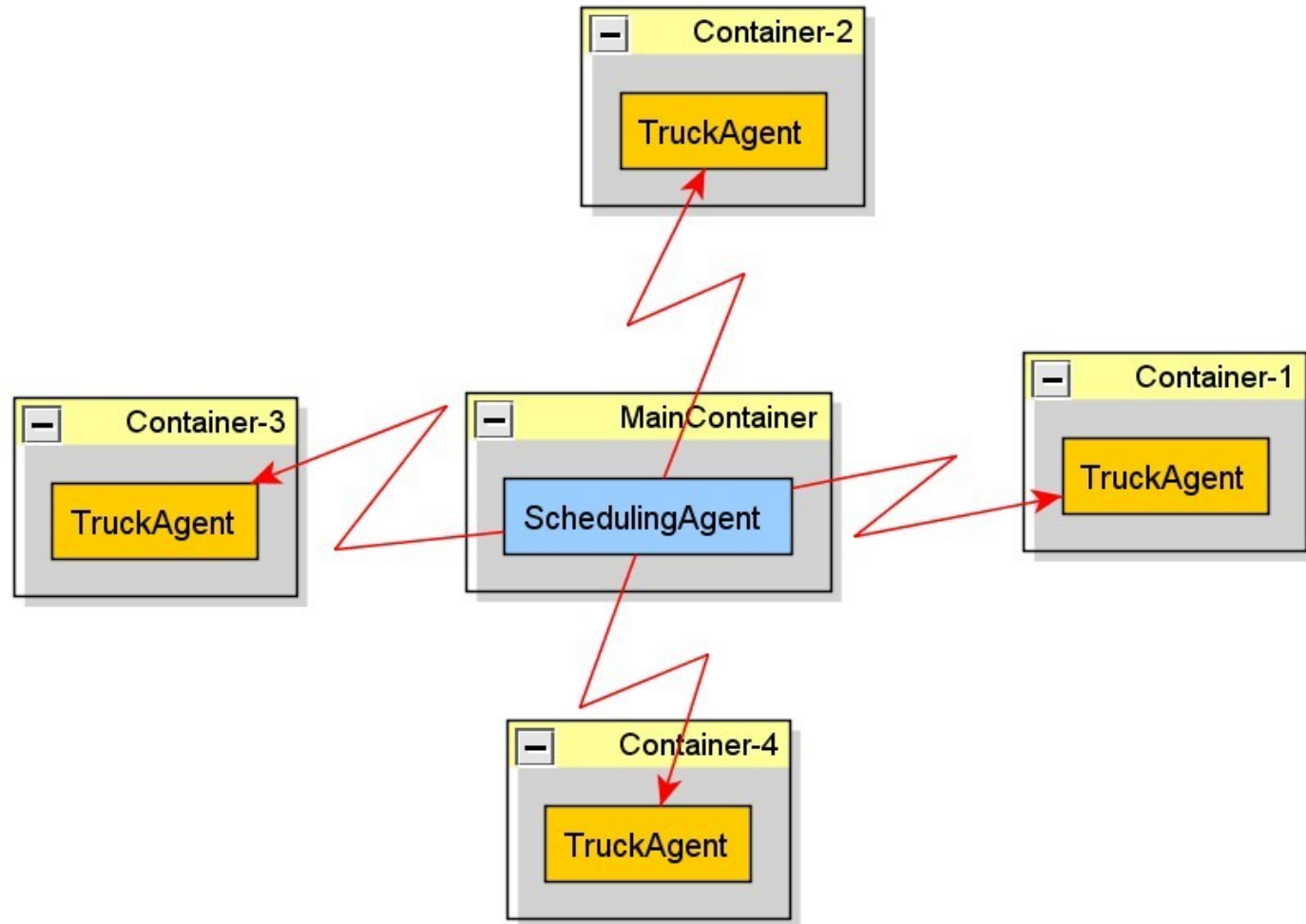
*“In other situations conventional communication (**moving the data to the code**) is faster because of a low amount of data to be transmitted and highly complex algorithms resulting in program code of considerable size working on the data. ”**

Example:

- ✓ **route planner** working on a small amount of update information about traffic jams with present geographical information with a **complex route finding algorithm**.

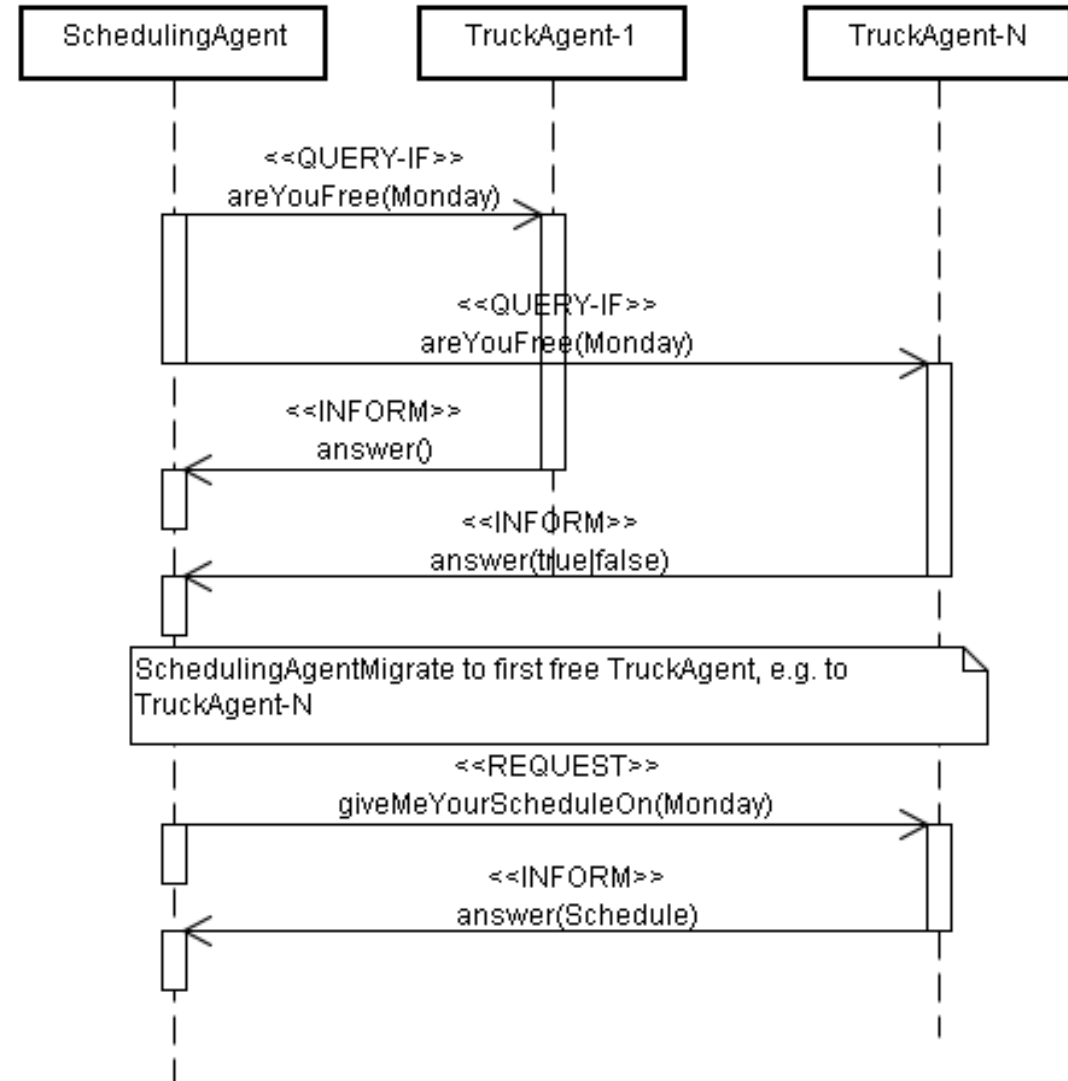
*Becker, M.; Singh, G.; Wenning, B.-L.; Görg, C.: On Mobile Agents for Autonomous Logistics: An Analysis of Mobile Agents considering the Fan Out and sundry Strategies. In: International Journal of Services Operations and Informatics, 1 (2007)

Simple logistic app – architecture



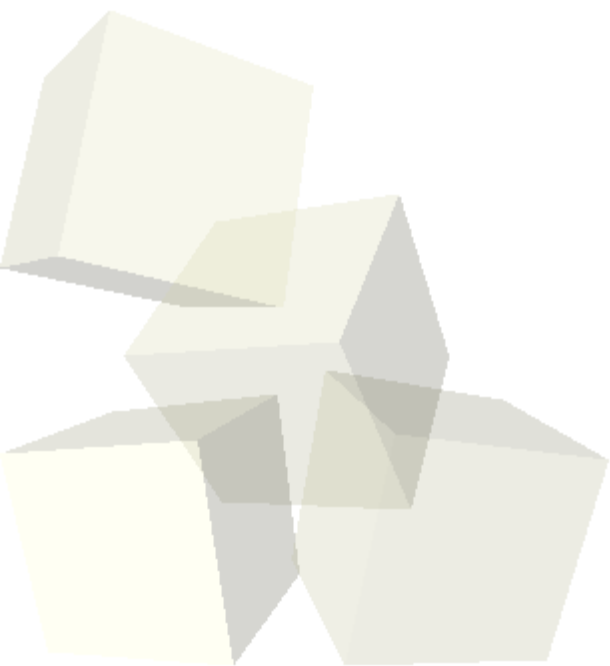


Simple logistic app – seq. diagram



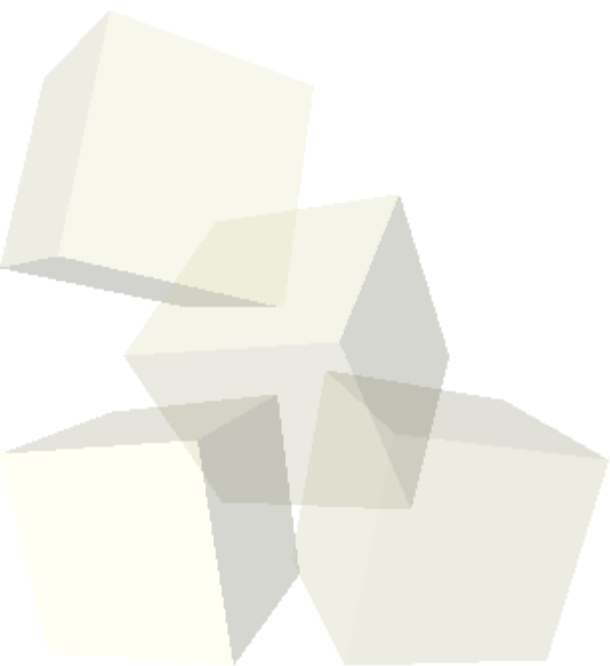


Questions ??





Part II: Creating agent



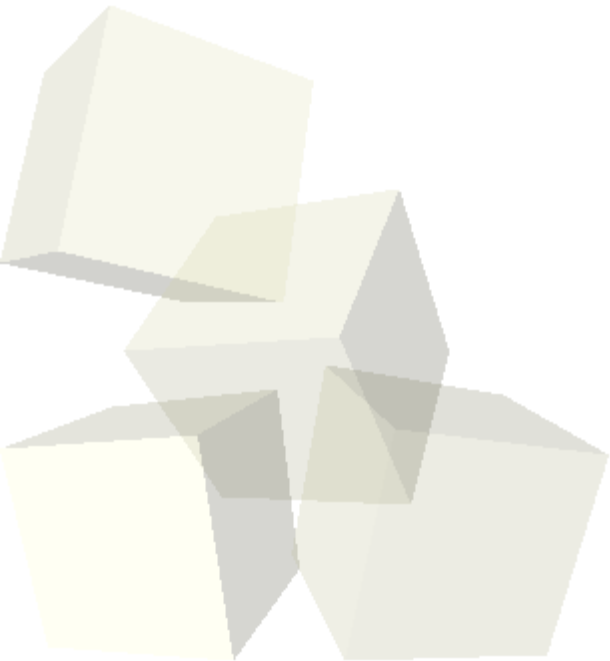


■ Software:

- ♦ JADE, [c:\Program Files\ABC\jade](#)
- ♦ Ant, [c:\Program Files\ABC\ant](#)

■ Code:

- ♦ <http://www.ibspan.waw.pl/~gawinec/>



***ABC:
Agent Based Computing***



JADE – how to start?

■ **JADE**, *Java Agent DEvelopment Framework*

- ♦ software and documentation
- ♦ complies with the FIPA specifications
- ♦ <http://jade.tilab.com>
- ♦ **register** at site and jade-dev@ mailing list !!!

■ **FIPA**, *Foundations of Intelligent Physical Agents*

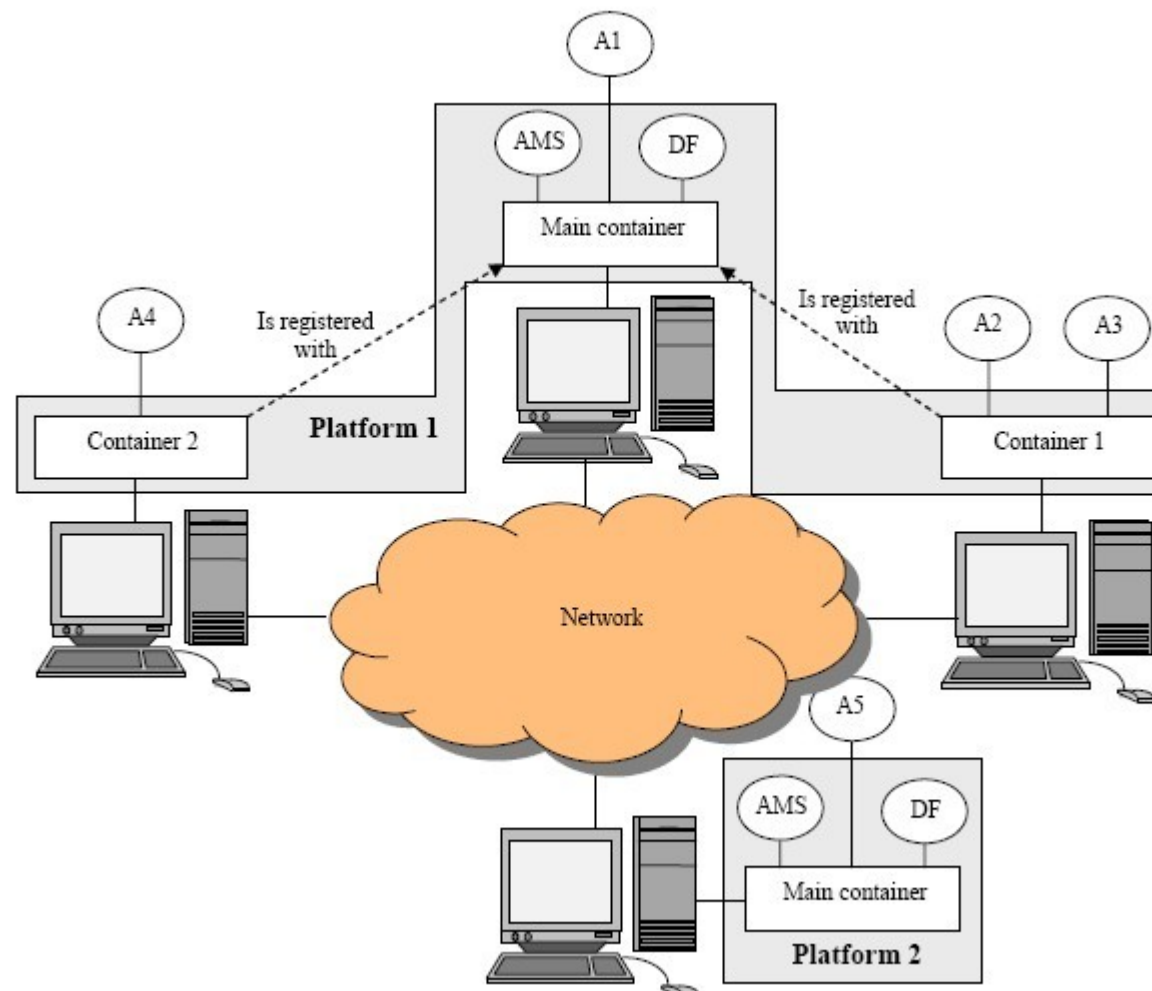
- ♦ Specifications
- ♦ <http://www.fipa.org>

■ Other documentation:

- ♦ Adam Łuszpaj, *JADE – materiały dydaktyczne*
<http://home.agh.edu.pl/~luszpaj/index.php?id=12>



Many platforms, many containers





Launching containers

Launching **Main-Container** with RMA

```
> java -cp $CLASSPATH  
    jade.Boot -gui
```

***RMA == Remote
Monitoring Agent
(GUI)***

Attaching **remote container**

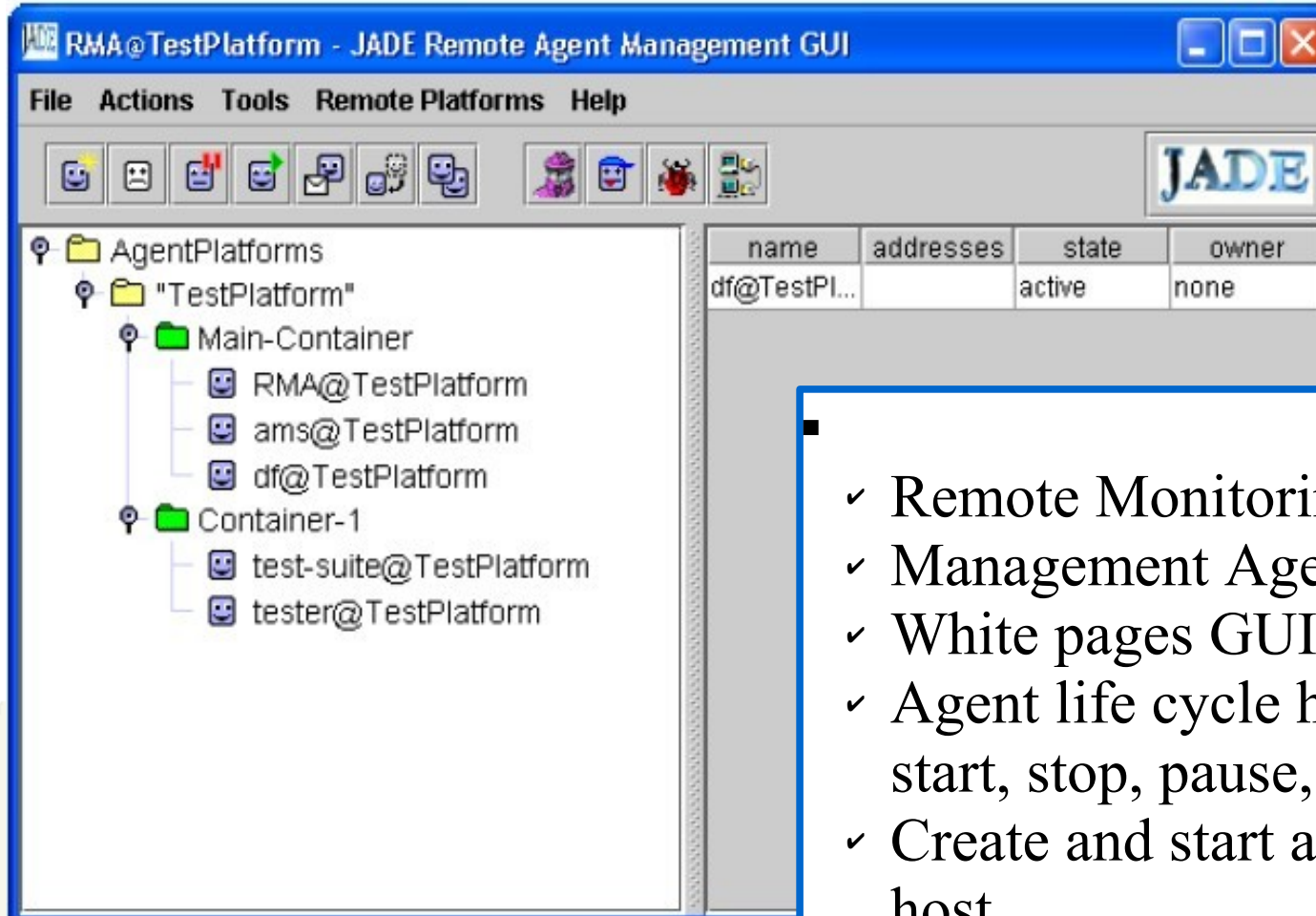
```
> java -cp $CLASSPATH  
    jade.Boot  
        -container  
        -host remoteMachine
```

Launching RMA at/from **remote host**

```
> java -cp $CLASSPATH jade.Boot  
    -container  
    -host remoteMachine  
    rma:jade.tools.rma.rma
```



Remote Agent Management



- ✓ Remote Monitoring Agent
- ✓ Management Agent
- ✓ White pages GUI – to find agents
- ✓ Agent life cycle handling allowing start, stop, pause, migrate, etc.
- ✓ Create and start agents on remote host
 - Assumes container already registered
- ✓ Naturally uses ACL for communication



Identifying an agent

- A type of agent is created by extending the `jade.core.Agent` class and redefining the `setup()` method.
- Each Agent instance is identified by an **AID** (`jade.core.AID`)
 - ✓ An AID is composed of a **unique name** plus **additional addresses**
 - ✓ An agent can retrieve its AID through the `getAID()` method of the **Agent** class

```
public class TruckAgent extends Agent {  
  
    @Override  
    protected void setup() {  
        System.out.println("My name is " +  
            getAID().getName());  
        System.out.println("I am free on Monday!");  
    };  
}
```

➤ *Code :*
`ibspan.lab1.ex1`

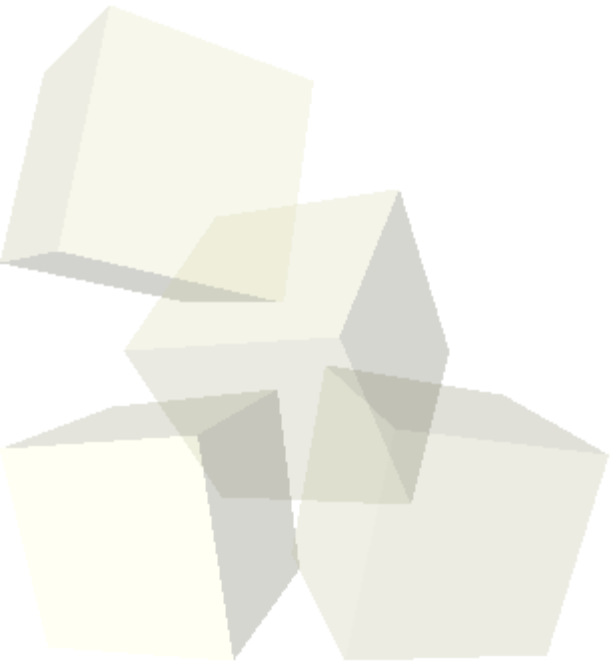


HelloWorld with agents

Launching an agent:

- from *GUI*
- from *command-line*

```
java  
  jade.Boot  
    hello:ibspan.lab1.ex1.TruckAgent
```





Local names, GUID and addresses

- Agent names are of the form **<local-name>@<platform-name>**
- The complete name of an agent must be **globally unique**.
- The **default** platform name is **<main-host>:<main-port>/JADE**
- The platform name can be set using the **-name** option
- Within a single JADE platform agents are referred through their names only.
- Given the name of an agent its AID can be created as
 - **AID id = new AID(localname, AID.ISLOCALNAME) ;**
 - **AID id = new AID(name, AID.ISGUID) ;**
- The addresses included in an AID are those of the platform MTPs and are **ONLY** used in communication between agents living on **different FIPA platforms**

GUID -Globally Unique Identifier

Passing arguments to an agent

- It is possible to pass arguments to an agent
 - `java jade.Boot a:myPackage.MyAgent (arg1 arg2)`
- The agent can retrieve its arguments through the `getArguments()` method of the **Agent** class

```
public class TruckAgent extends Agent {  
    private boolean isFree = false;  
    protected void setup() {  
        Object[] args = getArguments();  
        if (args.length >= 1)  
            isFree = Boolean.parseBoolean((String) args[0]);  
  
        System.out.println("I am " +  
            (isFree ? "free" : "not free") +  
            " !");  
    }  
}
```

➤ *Code:*
`ibspan.lab1.ex2`

```
java  
  jade.Boot  
    hello:ibspan.lab1.ex2.TruckAgent (free)
```



Behavioural programming (I)

- The actual job that an agent does is typically carried out within “**behaviours**”
- Behaviours are created by extending the **jade.core.behaviours.Behaviour** class
- To make an agent execute a task it is sufficient to create an instance of the corresponding **Behaviour** subclass and call the **addBehaviour()** method of the **Agent** class.
- Each **Behaviour** subclass must implement
 - ♦ **public void action()**: what the behaviour actually does
 - ♦ **public boolean done()**: whether the behaviour is finished



Behavioural programming (III)

➤ Code: `jade.core.behaviours`

```
public abstract class CyclicBehaviour
    extends SimpleBehaviour {
    /**
        Default constructor. It does not
        set the owner agent.
    */
    public CyclicBehaviour() {
        super();
    }

    /**
        This constructor sets the owner
        agent for this
        <code>CyclicBehaviour</code>.
        @param a The agent this behaviour
        must belong to.
    */
    public CyclicBehaviour(Agent a) {
        super(a);
    }
```

```
    /**
        This is the method that makes
        <code>CyclicBehaviour</code>
        cyclic, because it always returns
        <code>>false</code>.
        @return Always <code>>false</code>.
    */
    public final boolean done() {
        return false;
    }
}
```

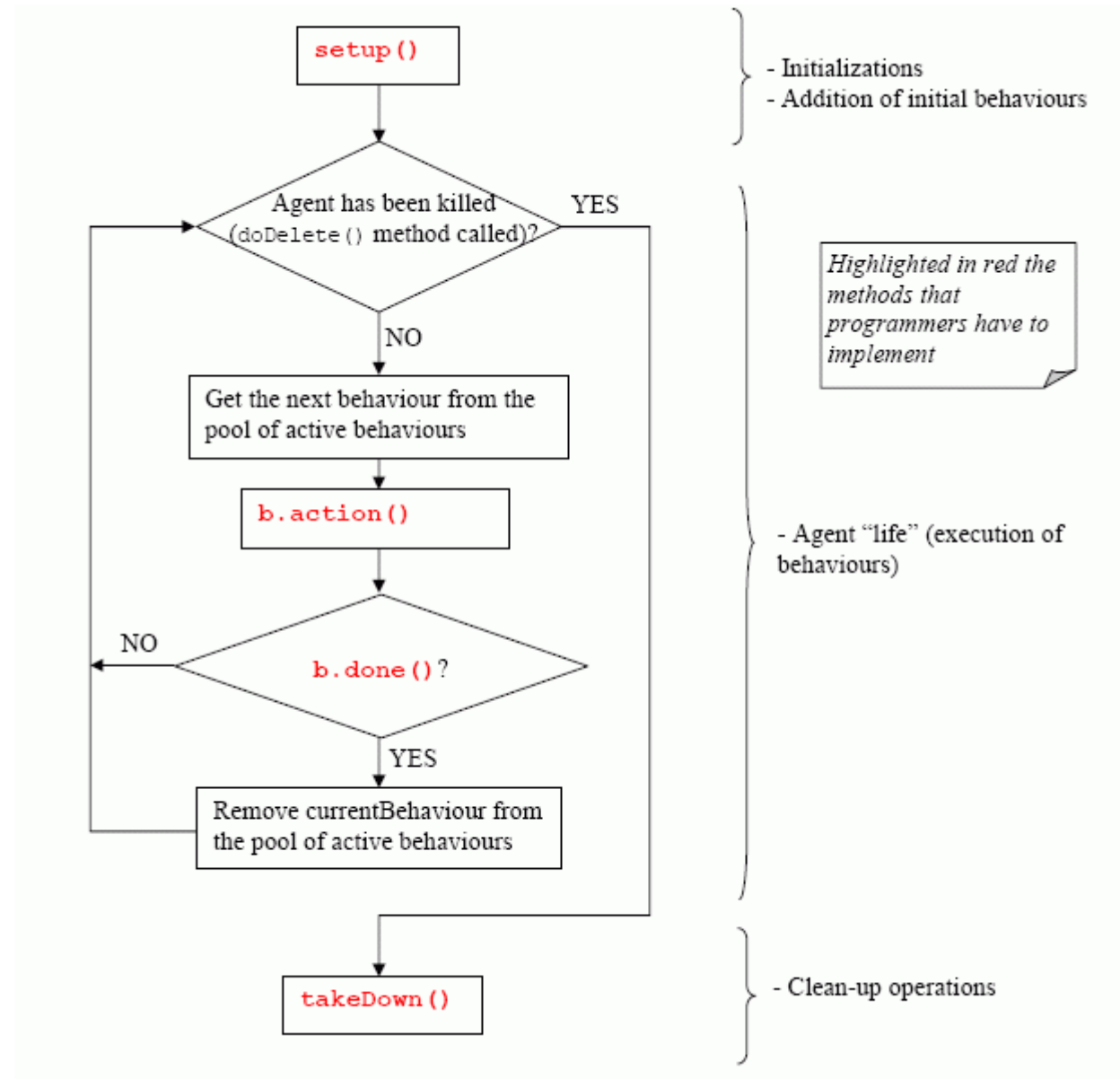
```
public abstract class OneShotBehaviour
    extends SimpleBehaviour {
    /**
        Default constructor. It does not
        set the owner agent.
    */
    public OneShotBehaviour() {
        super();
    }

    /**
        This constructor sets the owner
        agent for this
        <code>OneShotBehaviour</code>.
        @param a The agent this behaviour
        belongs to.
    */
    public OneShotBehaviour(Agent a) {
        super(a);
    }
```

```
    /**
        This is the method that makes
        <code>OneShotBehaviour</code>
        one-shot, because it always
        returns <code>>true</code>.
        @return Always <code>>true</code>.
    */
    public final boolean done() {
        return true;
    }
}
```



Agent execution thread path





Example of CyclicBehaviour

```
public class TruckAgent extends Agent {  
    private boolean isFree = false;  
    protected void setup() {  
        Object[] args = getArguments();  
        if (args.length >= 1)  
            isFree = Boolean.parseBoolean((String) args[0]);  
  
        Behaviour b = new OneShotBehaviour(this) {  
            public void action() {  
                System.out.println("I am " +  
                    (isFree ? "free" : "not free") +  
                    " !");  
            }  
        };  
        addBehaviour(b);  
    }  
}
```

➤ Code:
ibspan.lab1.ex3

Usage of anonymous classes
for introducing new
behaviours is common
pattern in JADE
programming



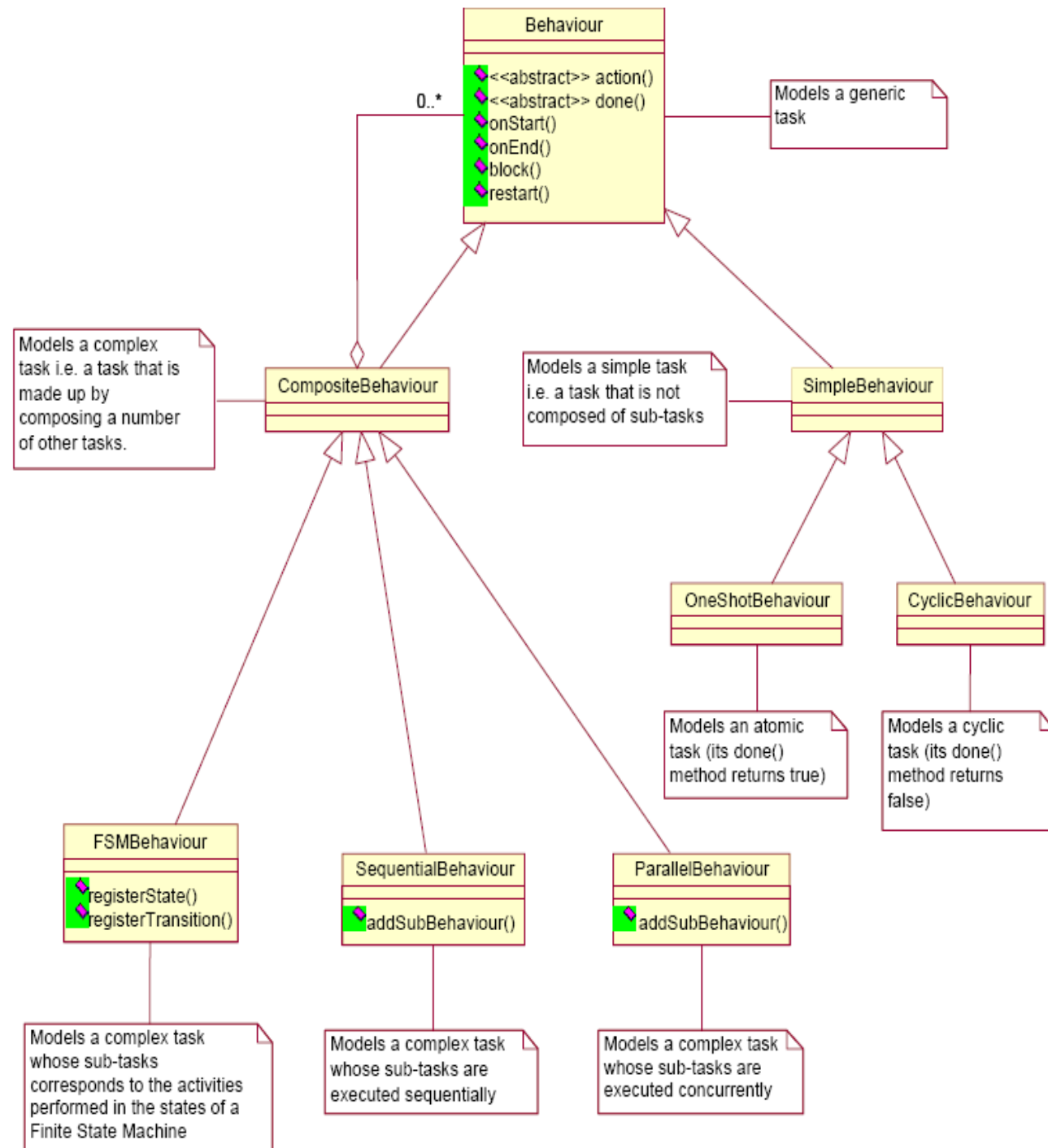
Agent termination

- An agent terminates when its **doDelete()** method is called.
- On termination the agent's **takeDown()** method is invoked (intended to include clean-up operations).

```
public class TruckAgent extends Agent {  
  
    private boolean isFree = false;  
  
    protected void setup() {  
        Object[] args = getArguments();  
        if (args.length >= 1)  
            isFree = Boolean.parseBoolean((String) args[0]);  
        else {  
            doDelete();  
            return;  
        }  
  
        Behaviour b = new CyclicBehaviour(this) {  
            public void action() {  
                System.out.println("I am " +  
                    (isFree ? "free" : "not free") +  
                    " !");  
            }  
        };  
        addBehaviour(b);  
    }  
  
    protected void takeDown() {  
        System.out.println("Bye...");  
    }  
}
```

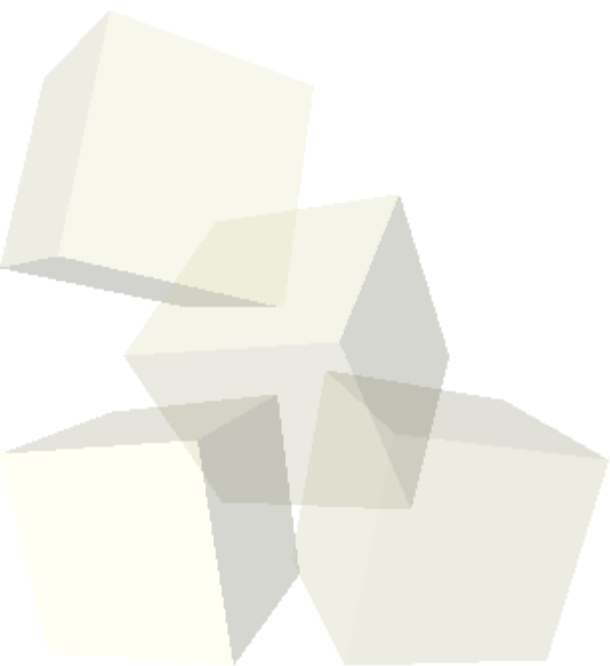


Behavioural programming (II)



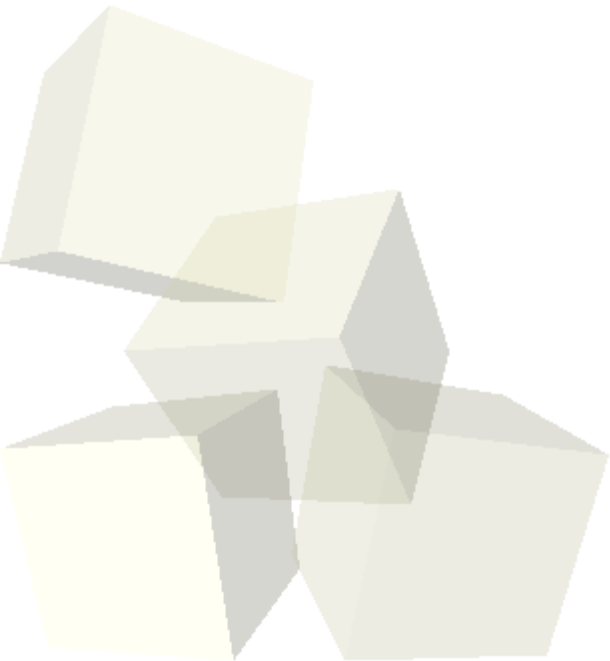


Questions ??





Part III: Concurrency to JADE





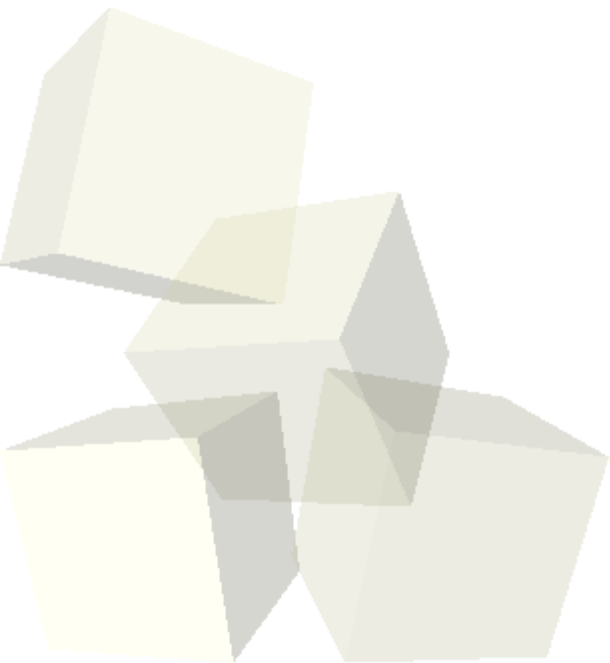
Concurrency in JADE

- Agent needs concurrency:
 - ♦ to engage in multiple simultaneous conversations
 - ♦ to execute several concurrent tasks
- Let us present it on example:
 - ♦ **AirConditioning** application, responsible for
 - Checking temperature outside
 - Checking temperature inside
 - ♦ Activities must be performed **concurrently** with **different frequency**
 - ♦ Let see different approaches in handling with concurrency
 - ♦ While putting common-used functionality into separate class: `ibspan.lab1.ex3.AirConditioning`



AirConditioning with threads (I)

- Threads are managed by JVM in **pre-emptive** manner
- This means threads works in **competitive** way





AirConditioning with threads (II)

Input:

```
final AirConditioning ac = new AirConditioning(36f,
    20f);

// Construct task checking temperature outside
Runnable task1 = new Runnable() {
    public void run() {
        while (true) {
            ac.checkTempOutside();
            try {
                Thread.currentThread().sleep(5000);
            } catch (InterruptedException ignore) {}
        }
    }
};

// Construct task checking temperature inside
Runnable task2 = new Runnable() {

    ...

// Starts runnable tasks as threads
new Thread(task1).start();
new Thread(task2).start();
```

Output:

```
Outside: 36.89621
  Inside: 20.098755
Outside: 36.962685
  Inside: 19.816002
Outside: 35.905853
Outside: 37.576965
  Inside: 19.861654
Outside: 39.236443
  Inside: 19.615541
Outside: 38.940544
Outside: 39.02758
```

➤ Code: `ibspan.lab1.ex3.AirConditioningThreads`



AirConditioning with agents (I)

- Each JADE agent is executed in a **single Java thread**
- Behaviours are managed by JADE in **non-re-emptive** manner
- This means behaviour works in **cooperative** way
 - ♦ Every behaviour must **release** the control to **allow** the other behaviours to be executed
 - ♦ *Behaviour switch occurs only when the **action ()** method of the currently scheduled behaviour returns.*
- JADE scheduler carries out a **round-robin** policy among all behaviours in the ready queue
- When the pool of active behaviours of an agent is empty the agent enters the **IDLE state** and its thread goes to sleep

AirConditioning with agents (II)

Input:

```
AirConditioning ac;

protected void setup() {
    ac = new AirConditioning(36f, 20f);

    // Construct behaviour checking temperature
    // outside
    Behaviour b1 = new SimpleBehaviour(this) {
        public void action() {
            while(true) {
                ac.checkTempOutside();
                block(5000);
            }
        }

        public boolean done() {
            return false;
        }
    };
    addBehaviour(b1);

    ...
    addBehaviour(b2);
}
```

Output:

```
Outside: 36.89621
Outside: 36.962685
Outside: 35.905853
Outside: 37.576965
Outside: 39.236443
Outside: 38.940544
Outside: 39.02758
```

*Infinite loop not
allowed in non-preemptive
sheduling!!!*

WRONG

➤ Code: `ibspan.lab1.ex3.AirConditioningAgent1`

AirConditioning with agents - :-)

Input:

```
AirConditioning ac;

protected void setup() {
    ac = new AirConditioning(36f, 20f);

    // Construct behaviour checking temperature
    // outside
    Behaviour b1 = new CyclicBehaviour(this) {
        public void action() {
            ac.checkTempOutside();
            block(5000);
        }
    };
    addBehaviour(b1);

    ...

    addBehaviour(b2);
}
```

CORRECT

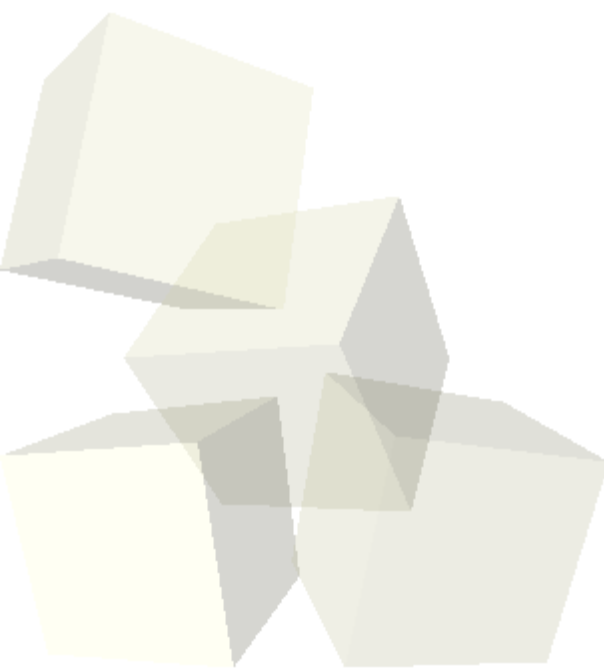
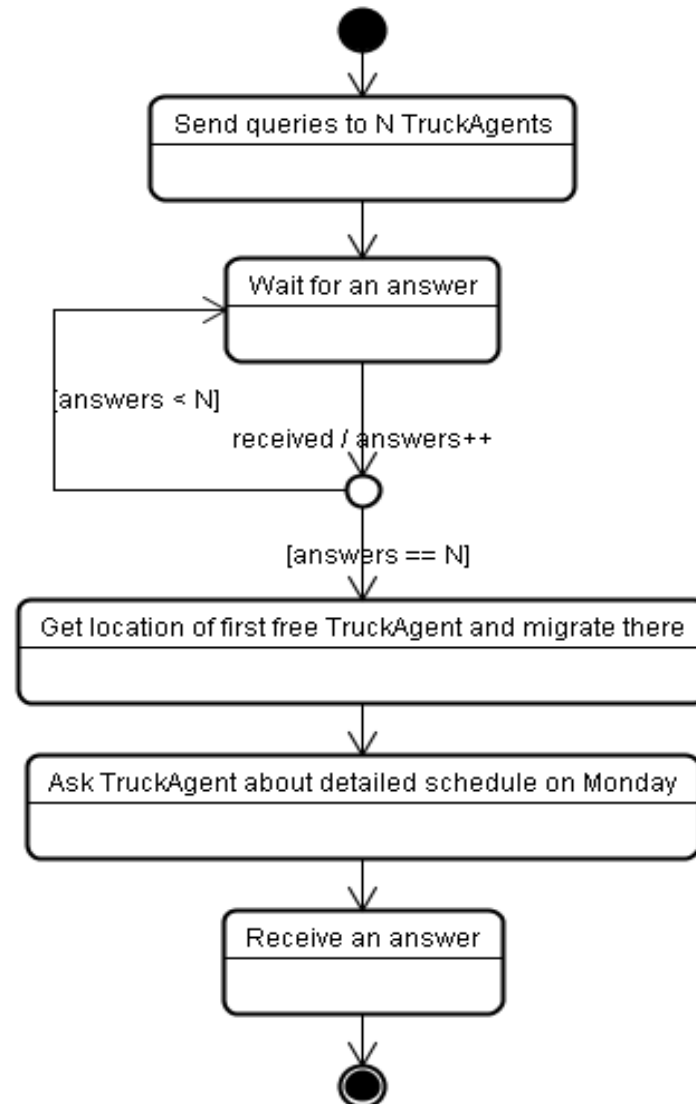
Output:

```
Outside: 36.89621
  Inside: 20.098755
Outside: 36.962685
  Inside: 19.816002
Outside: 35.905853
Outside: 37.576965
  Inside: 19.861654
Outside: 39.236443
  Inside: 19.615541
Outside: 38.940544
Outside: 39.02758
```

➤ Code: `ibspan.lab1.ex3.AirConditioningAgent2`



SchedulingAgent states' flow





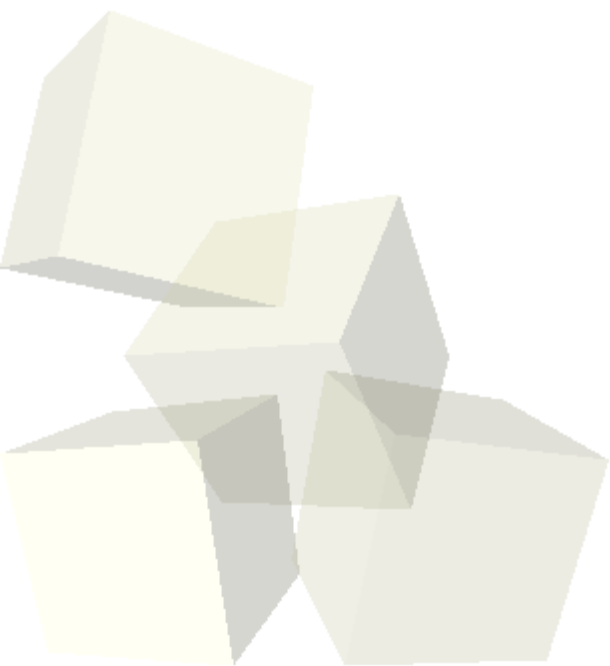
Saving stack

- Everytime a behaviour release control, it is the programmer who is responsible for saving a stack

```
Behaviour b = new SimpleBehaviour(this) {  
    private int step = 0;  
    public void action() {  
        switch (step) {  
            // Send queries, about which Truck Agent is free  
            case 0:  
                ...  
                step++; break;  
            // Collect all answers, cyclicly  
            case 1:  
                ...  
                // if (allAnswersCollected)  
                step++;  
                break;  
            // Get location of first free TruckAgent and migrate there  
            case 2:  
                ...  
                step++; break;  
            // Ask TruckAgent about detailed schedule on Monday  
            case 3:  
                ...  
                step++; break;  
            // Receive an answer  
            case 4:  
                ...  
                // if (received)  
                step++; break;  
        }  
  
        public boolean done() { return (step == 5); }  
    }  
};
```

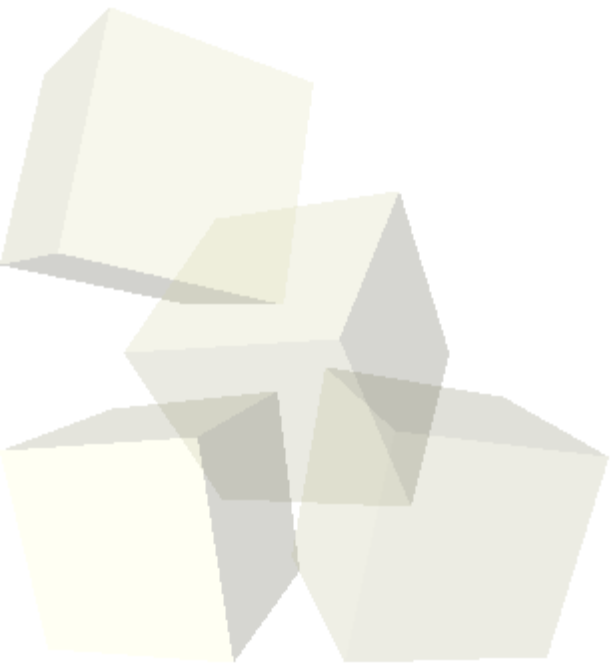


Questions ??



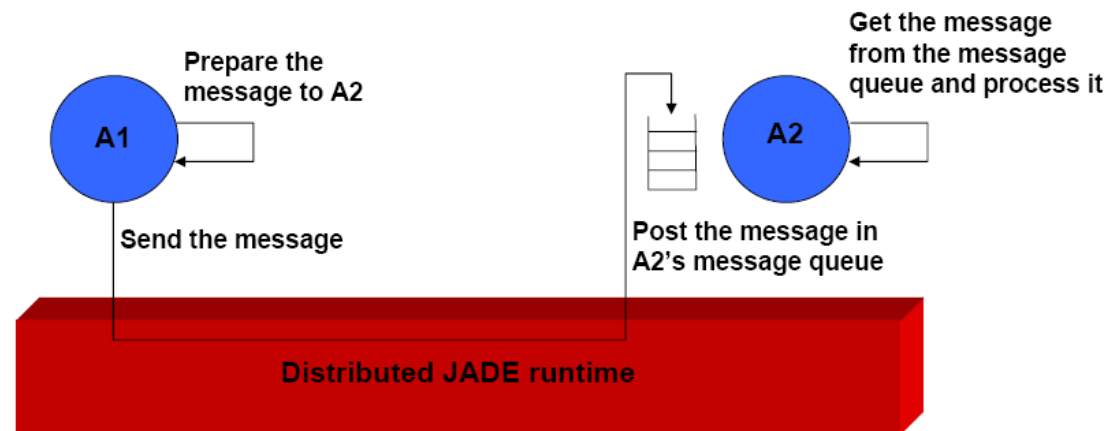


Part IV:
Agent
communication
in JADE



Communication model & subsystem

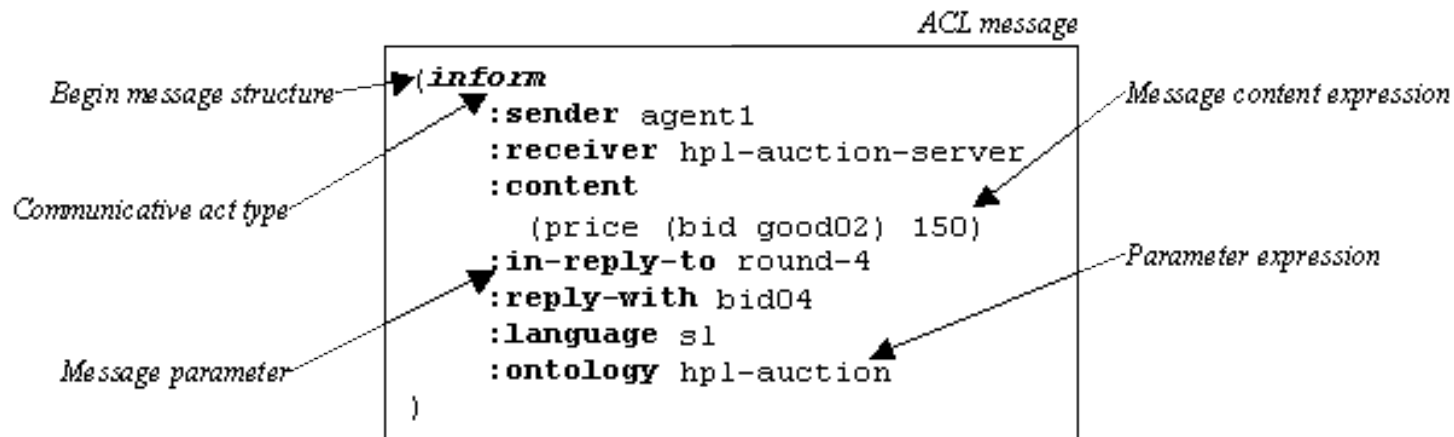
- Every agent has a private queue of ACL messages
- Communication based on **asynchronous** message passing
- Message format defined by the **ACL** language (FIPA)
- If you send a message to another agent and the sub-system can't find target, then it sends it to the AMS to handle
- Sending is completely transparent to where the agent resides (local/remote), the platform that takes care of selecting the most appropriate address and transport mechanism.





Agent Communication Language

- Structured message, targeted for flexible communication



- *Performative* = *communicative act* == *speech act* (*INFORM*, *QUERY*, *REFUSE*, ...)
- *Addressing*: *To*, *From*
- *ConversationID* – Used to link messages in same conversation
- *In reply to* – Sender uses to help distinguish answers
- *Reply with* – Another field to help distinguish answers
- *Reply by* – Used to set a time limit on an answer
- *Language* – Specifies which language is used in the content
- *Ontology* – Specifies which ontology is used in the content
- *Protocol* – Specifies the protocol
- *Content* – This is the main content of the message





The ACLMessage class

- Messages exchanged by agents are instances of the **jade.lang.acl.ACLMessage** class.
- Provide accessor methods to get and set all the fields defined by the ACL language
 - ♦ `get/setPerformative()` ;
 - ♦ `get/setSender()` ;
 - ♦ `add/getAllReceiver()` ;
 - ♦ `get/setLanguage()` ;
 - ♦ `get/setOntology()` ;
 - ♦ `get/setContent()` ;
 - ♦





Sending and receiving messages

- Sending a message is as simple as creating an **ACLMessage** object and calling the **send()** method of the **Agent** class

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);  
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME));  
msg.setLanguage("English");  
msg.setOntology("Weather-Forecast-Ontology");  
msg.setContent("Today it's raining");  
send(msg);
```

- Reading messages from the private message queue is accomplished through the **receive()** method of the **Agent** class.

```
ACLMessage msg = receive();  
if (msg != null) {  
    // Process the message  
}
```

Blocking / waiting for a message

- A behaviour that processes incoming messages does not know exactly when a message will arrive – It should poll the message queue by continuously calling **myAgent.receive()**.
- This of course would completely waste the CPU time.
- The **block()** method of the Behaviour class removes a behaviour from the agent pool and puts it in a blocked state.
- Each time a message is received all blocked behaviours are inserted back in the agent pool and have a chance to read and process the message.

```
public void action() {  
    ACLMessage msg = myAgent.receive();  
    if (msg != null) {  
        // Process the message  
    } else {  
        block();  
    }  
}
```

*This is the strongly
recommended pattern to
receive messages
within a behaviour*



Receiving a message

```
// Prepare behaviour responsible for cyclic reading
Behaviour b = new CyclicBehaviour(this) {

    public void action() {
        ACLMessage rcv = receive();
        if (rcv != null) {
            // Performative (communicative act type) is base
            // for choice of reaction.
            switch(rcv.getPerformative()) {
                case ACLMessage.QUERY_IF:
                    // We are assuming this is a question of SchedulingAgent
                    // about being free on Monday
                    ACLMessage response = rcv.createReply();
                    response.setPerformative(ACLMessage.INFORM);
                    response.setContent("" + isFree);
                    send(response);
                    break;
                ...
            }
        }
    }
}
```

➤ Code: `ibspan.lab1.ex5.TruckAgent1`



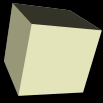
Selective reading messages

- ✓ The **receive()** method returns the first message in the message queue and removes it
- ✓ If there are two (or more) behaviours receiving messages, one may “steal” a message that the other one was interested in.
- ✓ To avoid this it is possible to read only messages with certain characteristics specifying a **jade.lang.acl.MessageTemplate** parameter in the **receive()** method.

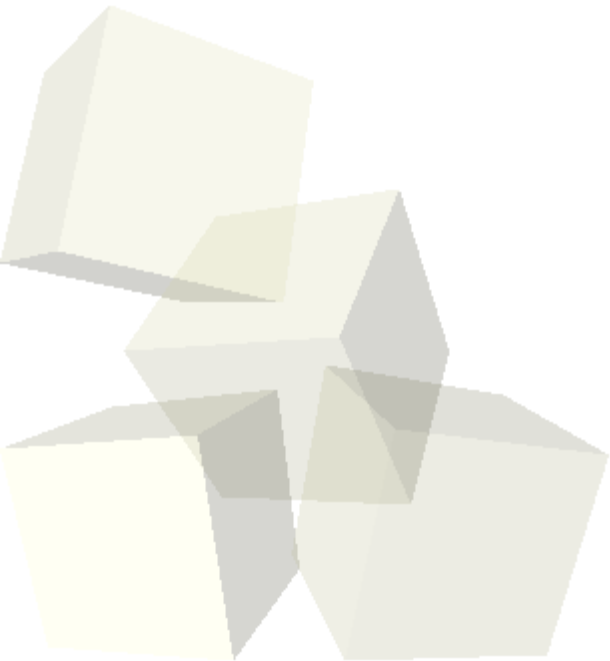
```
// Prepare behaviour responsible for cyclic reading
// a question of SchedulingAgent about being free on Monday
Behaviour b = new CyclicBehaviour(this) {
    public void action() {
        MessageTemplate mt =
            MessageTemplate.MatchPerformative(ACLMessage.QUERY_IF);
        ACLMessage rcv = receive(mt);
        if (rcv != null) {
            ACLMessage response = rcv.createReply();
            response.setPerformative(ACLMessage.INFORM);
            response.setContent("" + isFree);
            send(response);
        } else
            block();
    }
};
...
```

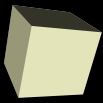
➤ Code: `ibspan.lab1.ex5.TruckAgent2`





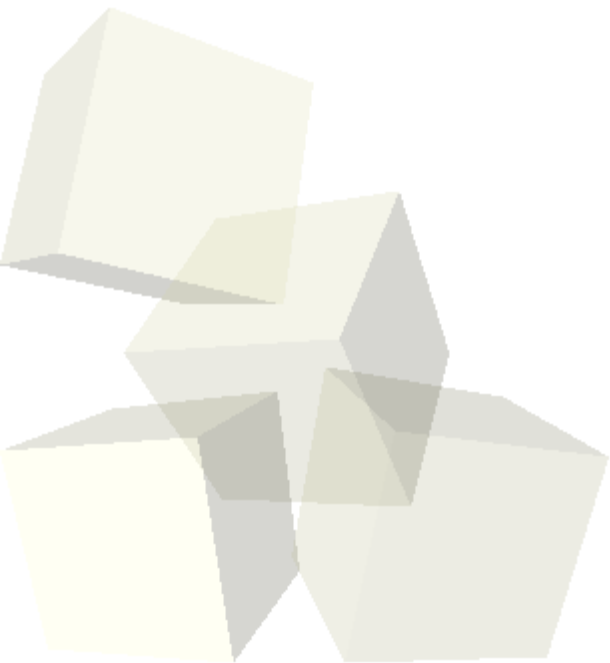
*Part IV:
Debugging
communication
in JADE*



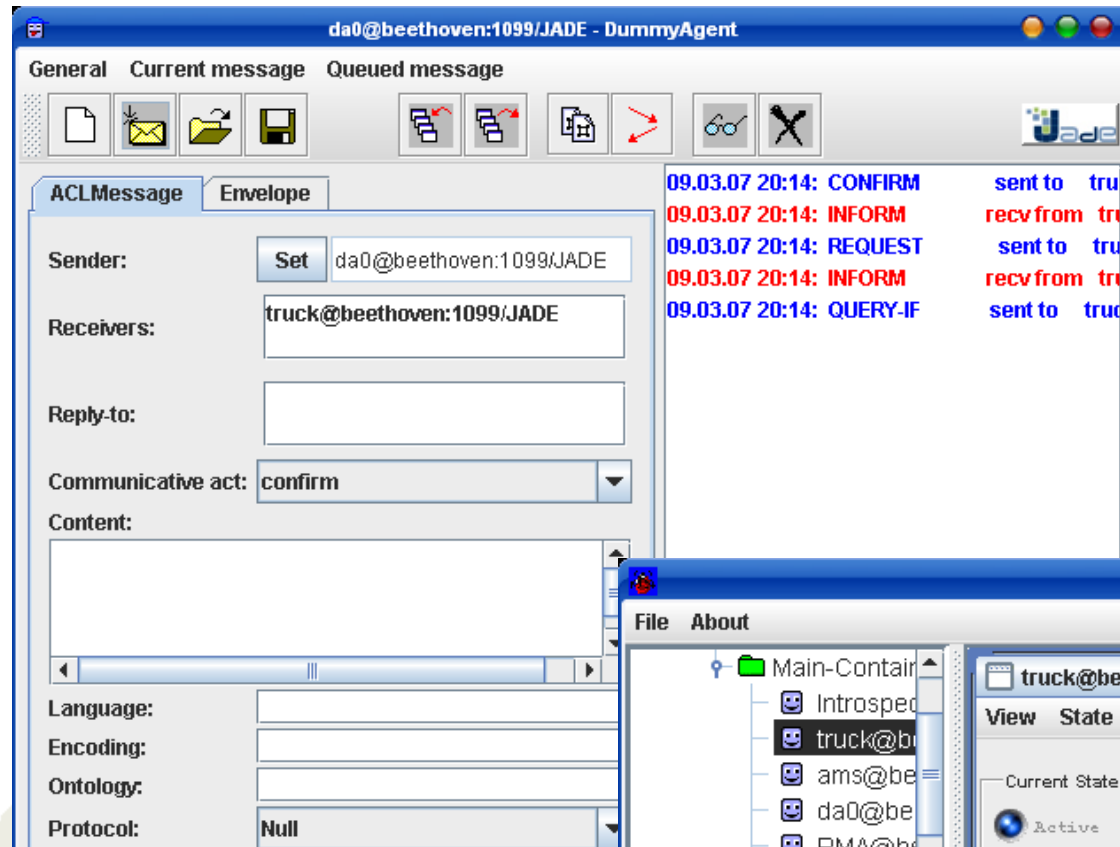


Debugging: documentation

- JADE Administrator's Guide



DummyAgent / IntrospectorAgent

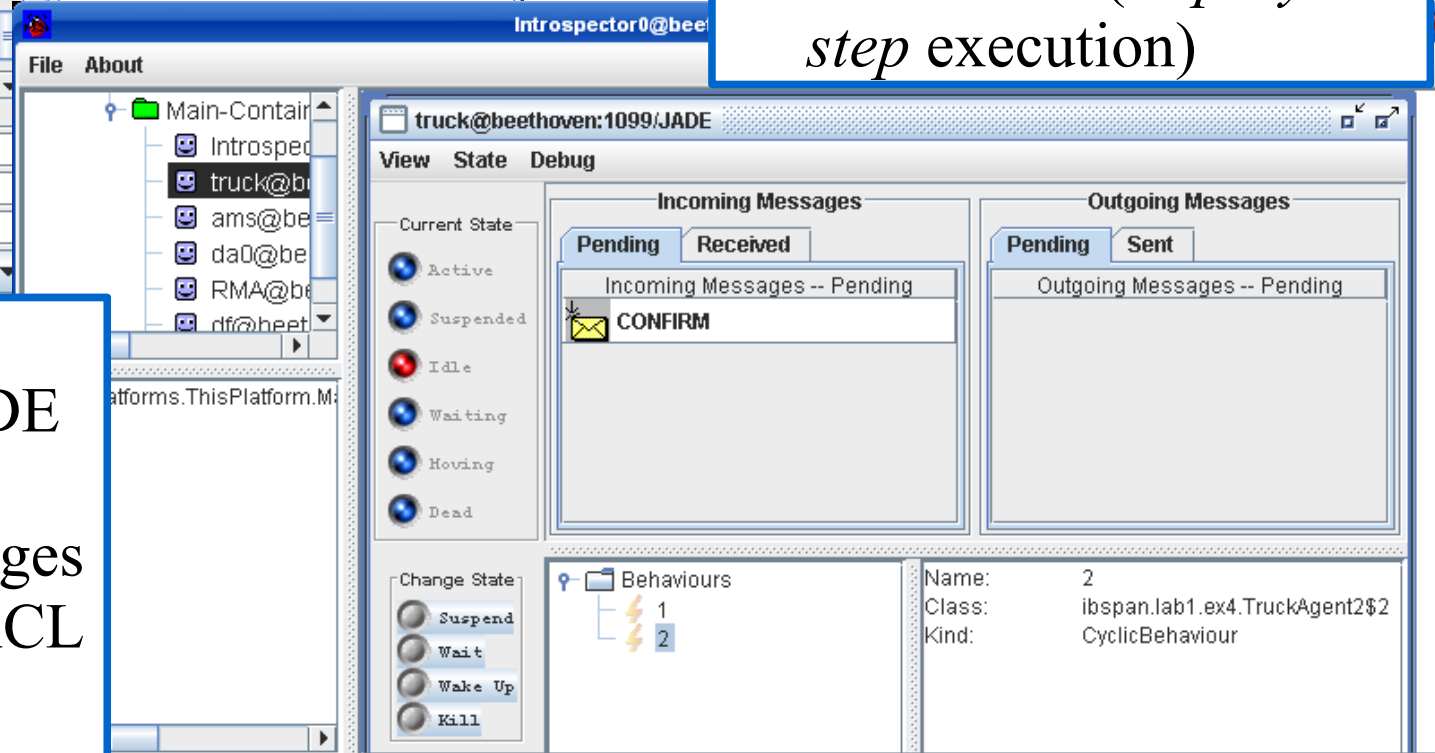


Introspector

- monitoring and controlling the *life-cycle* of agent
- monitoring agent's exchanged messages
- monitoring the queue of behaviours (*step-by-step* execution)

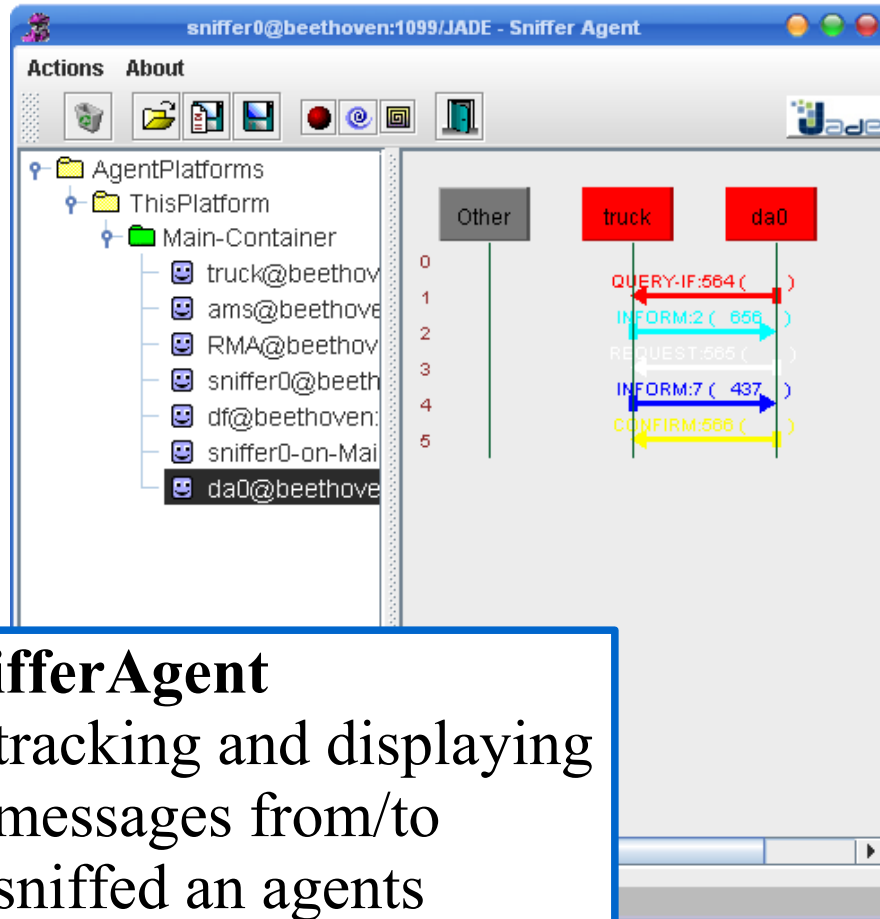
DummyAgent

- interacting with JADE agents
- sending ACL messages
- maintains a list of ACL messages sent and received





SnifferAgent



■ SnifferAgent

- tracking and displaying messages from/to sniffed agents
- saving tracked messages

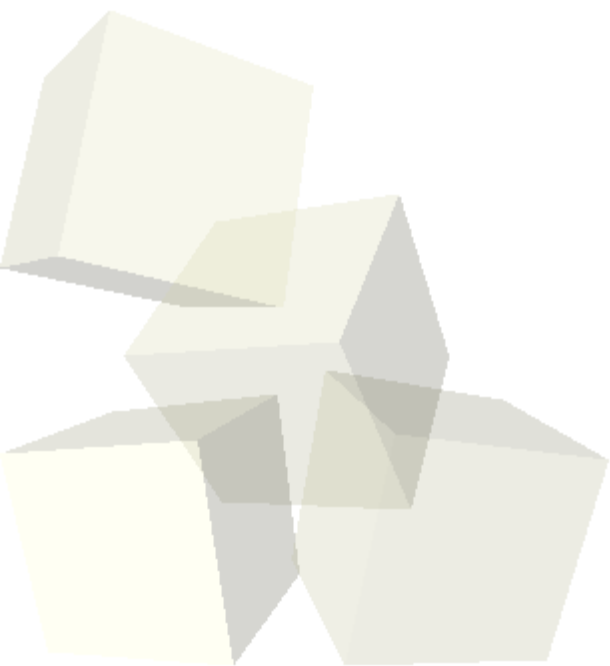
The 'ACL Message' dialog box has two tabs: 'ACLMessage' and 'Envelope'. The 'ACLMessage' tab is active, showing the following fields:

- Sender: View da0@beethoven:1099/JADE
- Receivers: truck@beethoven:1099/JADE
- Reply-to: (empty)
- Communicative act: query-if
- Content: (empty text area)
- Language: (empty)
- Encoding: (empty)
- Ontology: (empty)
- Protocol: Null
- Conversation-id: (empty)
- In-reply-to: (empty)
- Reply-with: (empty)
- Reply-by: View (empty)
- User Properties: (empty)

An 'OK' button is located at the bottom right.

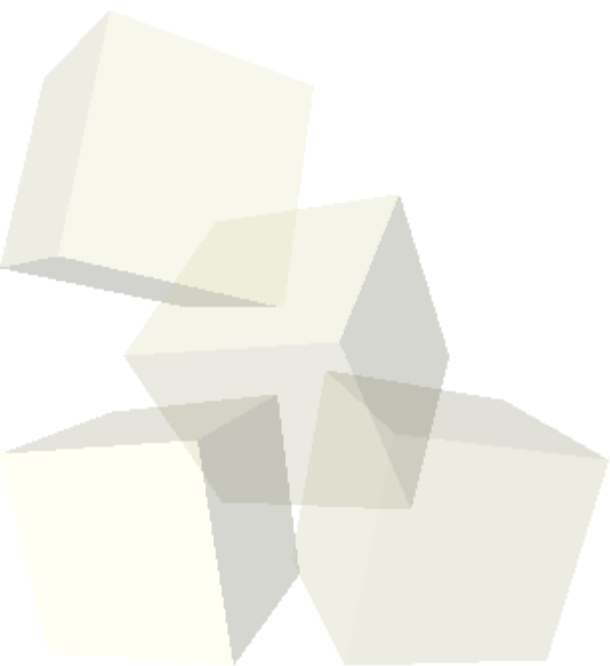


Questions ??





Part V: Planning mobility – roadmap





Mobility – seemingly easy

```
Location remoteDestination = ...; // !@#$%^& ??!  
doMove(remoteDestination);
```

*How can I **know** the name of
remoteDestination ?*

*Maybe I should **type it myself** ?*





JavaDoc about migration

- ✓ **jade.core.Location** is an abstract interface, so application agents are not allowed to create their own locations.
- ✓ They must **ask the AMS**:
 - ✓ for the list of the **available locations** and choose one
 - ✓ or **where** (at which location) another **agent lives**.

```
doMove  
public void doMove(Location destination)
```

Make this agent move to a remote location. This method is intended to support agent mobility and is called either by the Agent Platform or by the agent itself to start a migration process. It should be noted that this method just changes the agent state to AP_TRANSIT. The actual migration takes place asynchronously.

NOT available in MIDP

Parameters:

destination - The Location to migrate to.

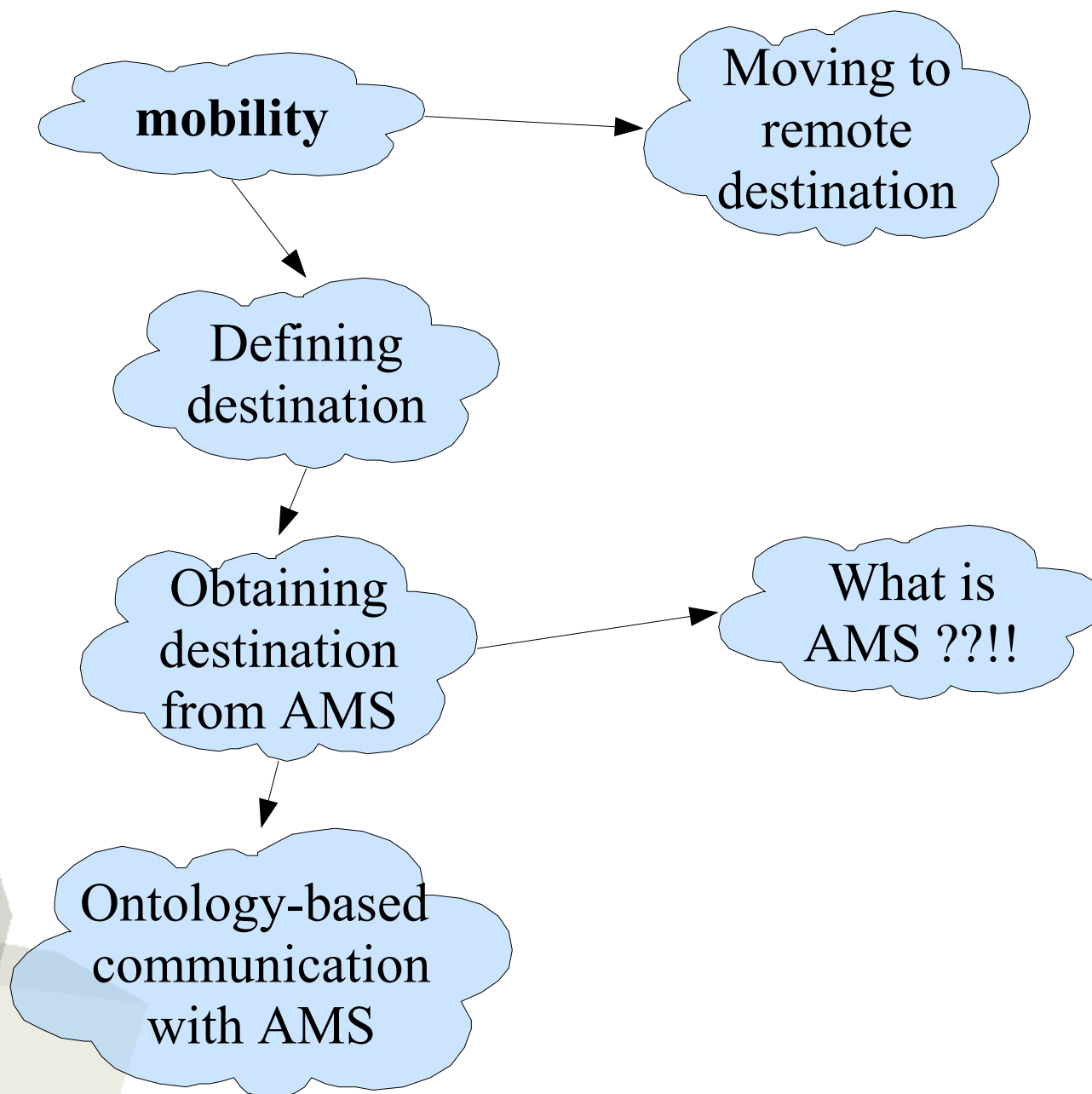
```
public interface Location extends Serializable, Concept
```

Abstract interface to represent JADE network locations. This interface can be used to access information about the various places where a JADE mobile agent can migrate.



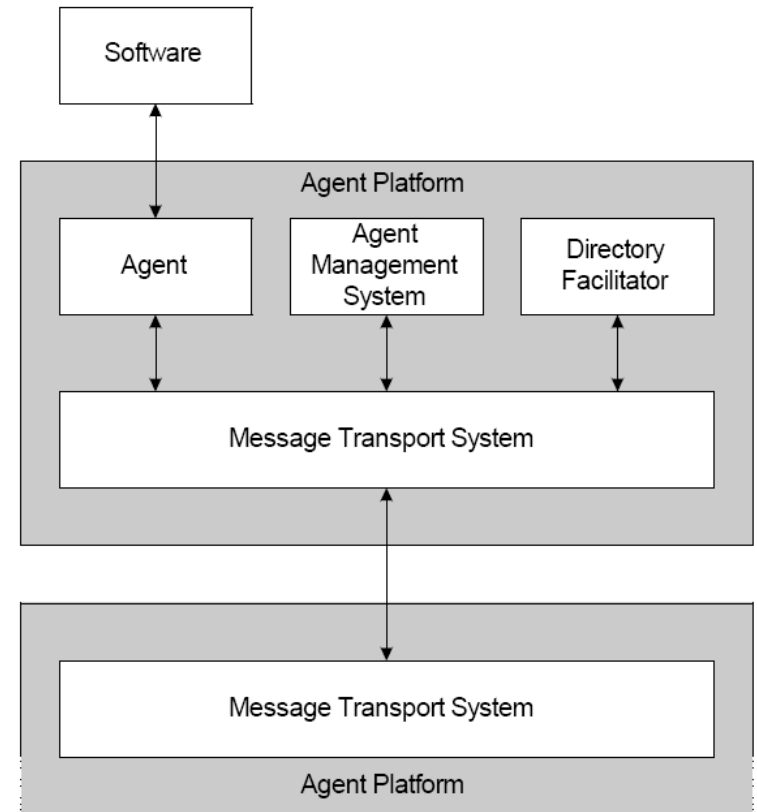


Mobility MindMap



Agent Management Reference Model

- An **Agent Management System (AMS)**:
 - ♦ mandatory component of the AP.
 - ♦ exerts supervisory control over use of the AP.
 - ♦ only one in a single AP
 - ♦ maintains a directory of AIDs
 - ♦ **offers white pages services to other agents.**
 - ♦ each agent must register with an AMS in order to get a valid AID
- An **Message Transport Service (MTS)** is the default communication method between agents on different AP
- An **Agent Platform (AP)** provides the physical infrastructure in which agents can be deployed.





AMS as White Pages service

AMS answers with the list of the available locations – **scenario 1.**

1. *Agent A* asks AMS about all possible containers in the platform.
2. *Agent A* receives set of locations from AMS.
3. *Agent A* migrates to the randomly chosen location.

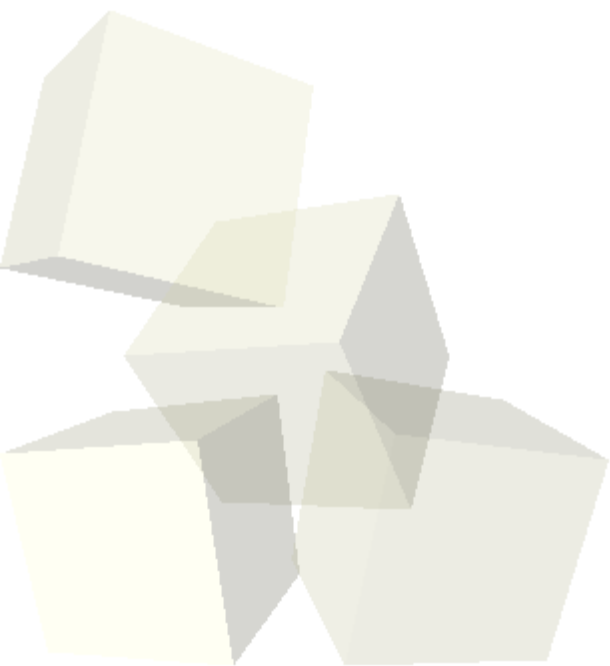
AMS answers with the location, where another agent lives – **scenario 2.**

1. *Agent A* takes as an argument name of *agent B*.
2. *Agent A* asks AMS about location (container) of *agent B*.
3. *Agent A* receives response from AMS about the location.
4. *Agent A* migrates to the location of *agent B*.

but...
AMS speaks in ontology !

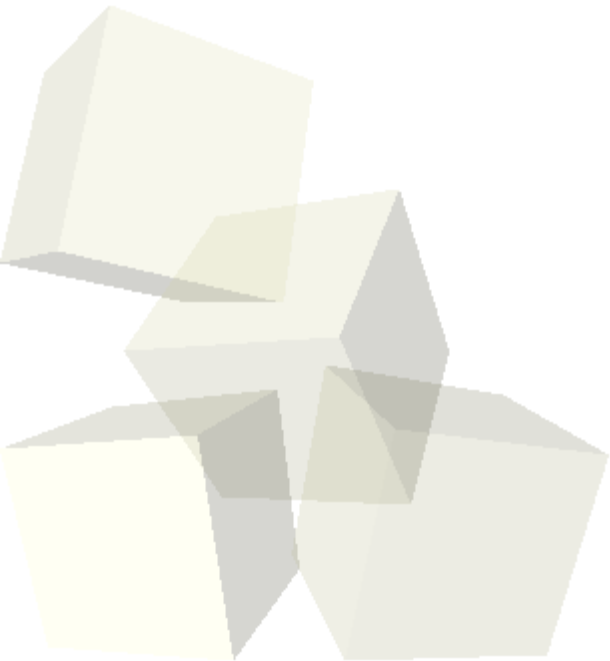


Questions ??





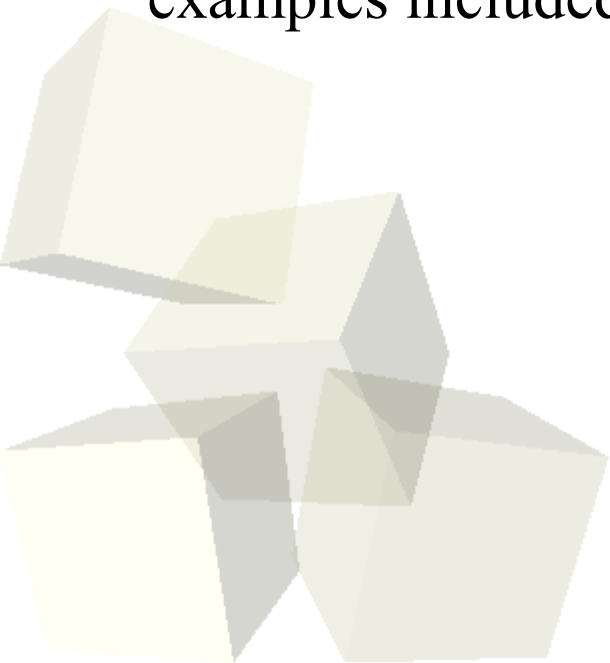
Part VI:
Introduction
to semantics of
communication
in JADE





Ontologies: Documentation

- A complete tutorial is available on the JADE site
JADE Tutorial Application-defined Content Languages and Ontologies,
<http://jade.tilab.com/doc/CLOntoSupport.pdf>
- API documentation (javadoc): **jade.content** package and subpackage
- Sample code: **examples.content** package in the examples included in the JADE distribution.





Communicative act type: Defines semantical structure of message content.

Content: That part of a communicative act which represents the domain dependent component of the communication.

```
(REQUEST
  :sender (agent-identifier :name dal@Zadig:1099/JADE)
  :receiver (set (agent-identifier :name ams@Zadig:1099/JADE))
  :content (( action
    (agent-identifier :name ams@Zadig:1099/JADE)
    (where-is-agent (agent-identifier :name Peter@Zadig:1099/JADE))
  ))
  :language FIPA-SLO
  :ontology JADE-Agent-Management
  :protocol fipa-request
)
```

Language: Denotes the encoding scheme of the content of the action.

Brings representation of message, e.g FIPA SL or it subsets (SLO, SL1).

Ontology: Denotes the ontology which is used to give a meaning to the symbols in the content expression. *An ontology gives meanings to symbols and expressions within a given domain language.*



REQUEST

- ✓ **Summary:** *The sender **requests** the receiver to **perform** some **action**.*
- ✓ **Message content:** *An **action description**.*
- ✓ **Description:** *The action can be any action the receiver is capable of performing: pick up a box, book a plane flight, change a password etc. An important use of the request act is to build composite conversations between agents, where the actions that are the object of the request act are themselves communicative acts such as **inform**.*

(REQUEST

...

```
:content (("action"
  (ActorOfAction)
    (ActionName (ActionParameters))
  ))
```

...

)

```
(REQUEST
:sender (agent-identifier :name dal@Zadig:1099/JADE)
:receiver (set (agent-identifier :name ams@Zadig:1099/JADE))
:content (( action
  (agent-identifier :name ams@Zadig:1099/JADE)
    (where-is-agent (agent-identifier :name Peter@Zadig:1099/JADE))
  ))
:language FIPA-SL0
:ontology JADE-Agent-Management
:protocol fipa-request
)
```



- ✓ **Summary:** *The sender **informs** the receiver that a given **proposition is true**.*
- ✓ **Message content:** *A **proposition** (i.e. predicate, here “result”)*

```
(INFORM
  :sender (agent-identifier :name ams@Zadig:1099/JADE)
  :receiver (set (agent-identifier :name dal@Zadig:1099/JADE))
  :content ((result
    (action
      (agent-identifier :name ams@Zadig:1099/JADE)
      (where-is-agent (agent-identifier :name Peter@Zadig:1099/JADE))
    )
    (set (location
      :name Container-1
      :protocol JADE-IPMT
      :address Zadig:1099/JADE.Container-1
    ))
  ))
```

(INFORM

```
..
:content (("result"
  ("action"
    (ActorOfAction)
    (ActionName (ActionParameters))
  )
  (ResultOfAction)
))
```

...

)



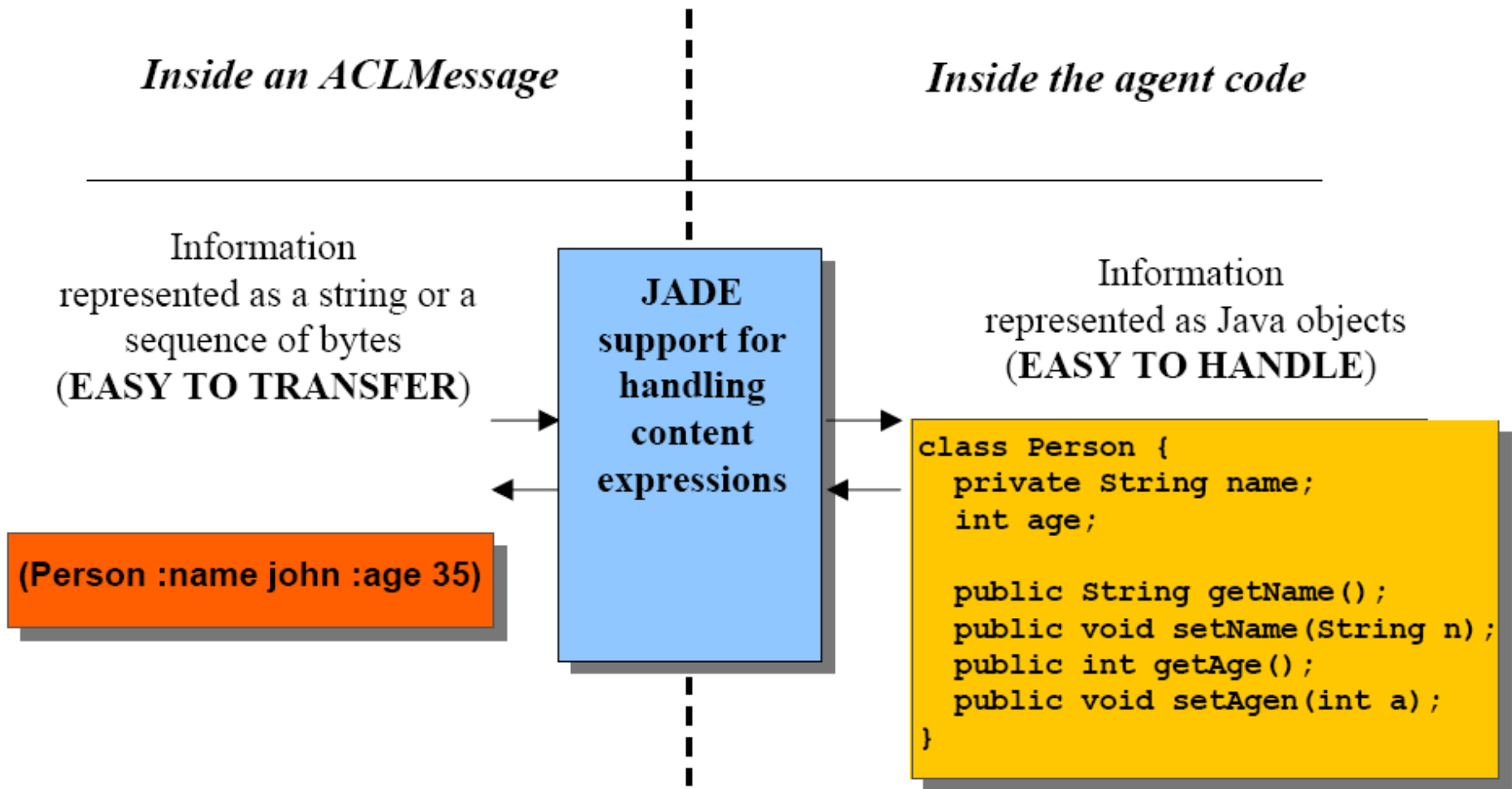


Working with the AMS – summary

- The AMS (Agent Management System) represents the authority in a JADE platform.
- All platform management actions (creating/killing agents, killing containers...) are under the control of the AMS.
- Other agents can request the AMS to perform these actions by using
 - ♦ The **Fipa-Request Interaction Protocol**
 - ♦ The **SL** language
 - ♦ The **JADE-Agent-Management ontology** and related actions
- The AID of the AMS can be retrieved through the **getAMS ()** method of the **Agent** class



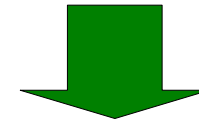
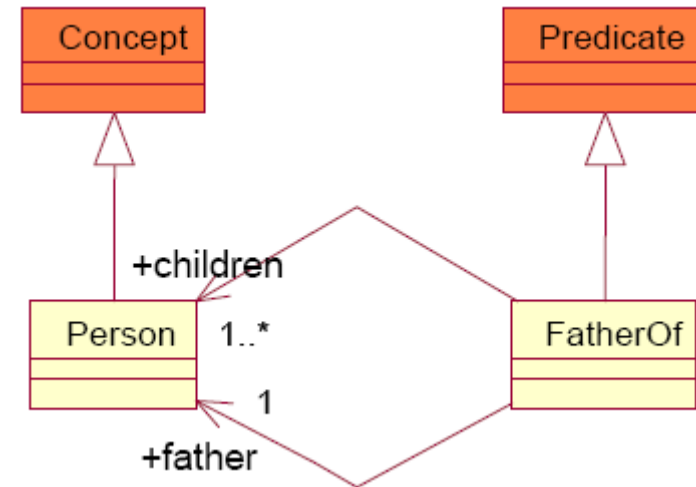
Handling content expressions



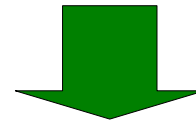


How it works

- ✓ Creating the Ontology (domain specific)
 - ✓ *Defining the schemas ontology elements*
 - ✓ *Defining the corresponding Java classes*
- ✓ Handling content expressions as Java objects
- ✓ Using the **ContentManager** to fill and parse message contents



```
Person john = new Person("John", 35);
Person bill = new Person("Bill", 67);
FatherOf f = new FatherOf();
f.setFather(bill);
f.addChildren(john);
```



```
(father-of
 (person :name Bill :age 67)
 (set (person :name John :age 35) ) )
```



Registering language and ...

```
// register the SLO content language
getContentManager().registerLanguage(new SLCodec(),
    FIPANames.ContentLanguage.FIPA_SLO);

// register the mobility ontology
getContentManager()
    .registerOntology(MobilityOntology.getInstance());
```





Creating request to AMS

```
private ACLMessage prepareRequestToAMS(AID agent) {
    ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
    request.addReceiver(getAMS());
    request.setLanguage(FIPANames.ContentLanguage.FIPA_SLO);
    request.setOntology(MobilityOntology.NAME);
    request.setProtocol(FIPANames.InteractionProtocol
        .FIPA_REQUEST);

    // creates the content of the ACLMessage
    Action act = new Action();
    act.setActor(getAMS());

    WhereIsAgentAction action = new WhereIsAgentAction();
    action.setAgentIdentifier(agent);
    act.setAction(action);

    try {
        getContentManager().fillContent(request, act);
    } catch (CodecException ignore) {}
    } catch (OntologyException ignore) {}

    return request;
}
```

➤ Code:
ibspan.lab1.ex6
.AgentA

```
public class WhereIsAgentAction
    implements AgentAction {
    private AID agentName;
    public WhereIsAgentAction() {}
    public void setAgentIdentifier(AID id)
        { agentName = id; }
    public AID getAgentIdentifier()
        { return agentName; }
}
```

```
(REQUEST
:sender (agent-identifier :name dal@Zadig:1099/JADE)
:receiver (set (agent-identifier :name ams@Zadig:1099/JADE))
:content (( action
    (agent-identifier :name ams@Zadig:1099/JADE)
    (where-is-agent (agent-identifier :name Peter@Zadig:1099/JADE))
))
:language FIPA-SLO
:ontology JADE-Agent-Management
:protocol fipa-request
)
```





Parsing answer from AMS

```
(INFORM
:sender (agent-identifier :name ams@Zadig:1099/JADE)
:receiver (set (agent-identifier :name dal@Zadig:1099/JADE))
:content ((result
  (action
    (agent-identifier :name ams@Zadig:1099/JADE)
    (where-is-agent (agent-identifier :name Peter@Zadig:1099/JADE))
  )
  (set (location
    :name Container-1
    :protocol JADE-IPMT
    :address Zadig:1099/JADE.Container-1
  ))
))
:reply-with dal@Zadig:1099/JADE
:language FIPA-SL
:ontology JADE-Agent
:protocol fipa-re
)
```

➤ Code:
ibspan.lab1
.ex6.AgentA

```
private Location parseAMSResponse(ACLMessage response) {
    Result results = null;
    try {
        results =
            (Result)getContentManager()
                .extractContent(response);
    } catch (UngroundedException e) {
    } catch (CodecException e) {
    } catch (OntologyException e) {}

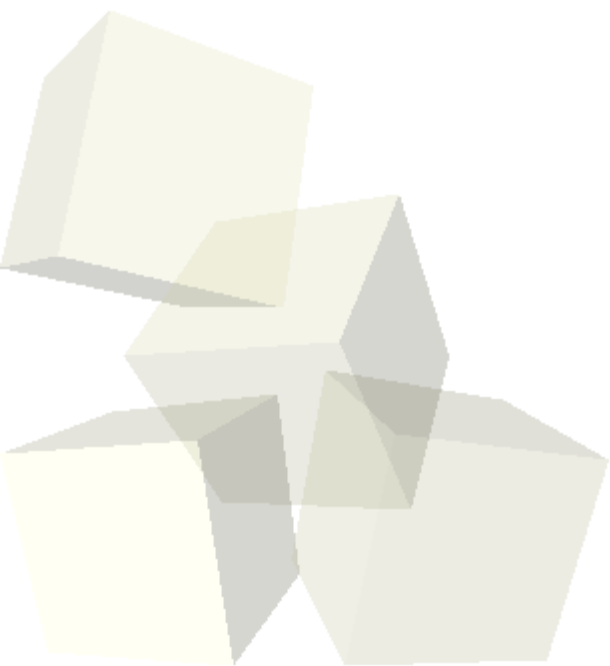
    Iterator it = results.getItems().iterator();

    Location loc = null;
    if (it.hasNext())
        loc = (Location) it.next();

    return loc;
}
```

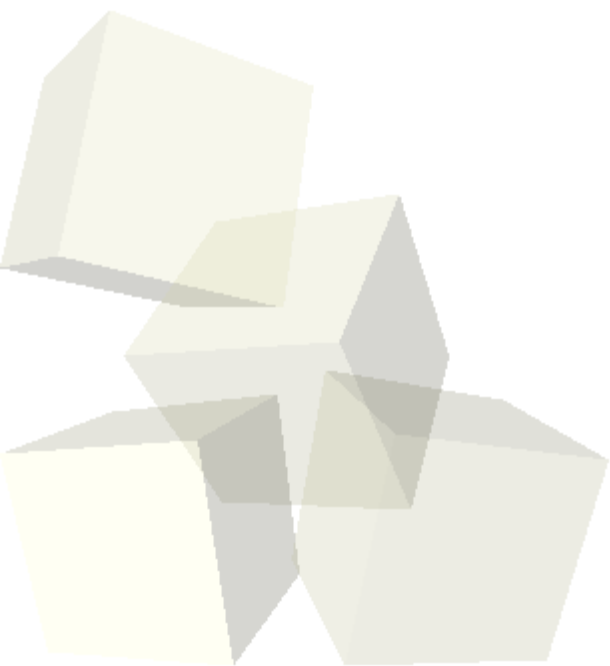


Questions ??





*Part VII:
Agent
mobility
in JADE*





Mobility: documentation

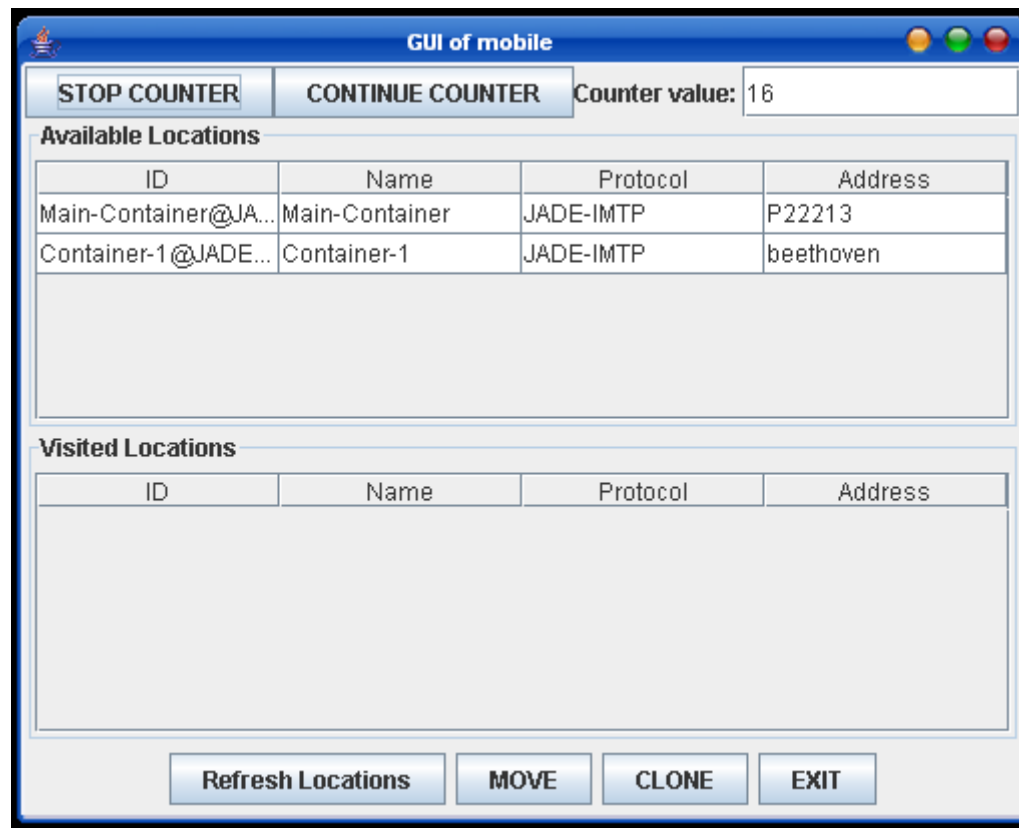
Documentation

- Chapter 3.7 in the **Programmers guide** included in the JADE distribution provides a detailed explanation of the mobility support
- API documentation (javadoc): **jade.core.Agent** class, **jade.core.Location** interface and **jade.domain.mobility** package
- Sample code: **examples.mobile** package in the examples included in the JADE distribution.
- In **FIPA Agent Management Support for Mobility Specification**
<http://www.fipa.org/specs/fipa00087/index.html>



Mobility example from JADE

➤ Code: `examples.mobile` in JADE package





1st mobile application

1. *Agent A* takes as an argument name of *agent B*.
2. *Agent A* asks AMS about location (container) of *agent B*.
3. *Agent A* receives response from AMS about the location.
4. *Agent A* migrates to the location of *agent B*.

➤ Code: `ibspan.lab1.ex4`





2nd mobile application

1. *Agent A* asks AMS about all possible containers in the platform.
2. *Agent A* receives set of locations from AMS.
3. *Agent A* migrates to the randomly chosen location.

➤ Code: `ibspan.lab1.ex5`





JADE – weak or strong mobility ?

- We usually say JADE supports *"not-so-weak mobility"*:
 - ✓mobility of **status**:
 - ✓JADE does not support **mobility of Java stacktrace**, as in e.g. Nomad system*
 - ✓In respect to non-preemptive behaviour scheduling: before migration **agent have to wait** before currently scheduled behaviour finishes its cycle.
 - ✓JADE agents can migrate in the middle of a conversation and a programmer can exploit the behaviour composition capability in order to define an arbitrarily fine-grained sequence of states in which an agent is allowed to migrate
 - ✓mobility of **code**:
 - ✓If the code of the moving agent is not available on the destination container it is automatically **retrieved on demand**.



Intra- and inter-platform mobility

- ✓ **Intra-platform mobility** is that, mobile agent can navigate across different agent containers but it is confined to a single JADE platform
- ✓ **Inter-platform mobility** provides agent with navigating among Agent Platforms
- ✓ FIPA defines extensions that are necessary to the **AMS to support mobility**.
- ✓ JADE containers are not **FIPA compliant** (FIPA does not specify them)
 - ✓ *“It should be noted that the concept of an AP does not mean that all agents resident on an AP have to be co-located on the same host computer.”**
 - ✓ Therefore FIPA does not specify intra-platform migration
- ✓ **JADE supports intra-platform** migration by adaptation of inter-platform migration specification
- ✓ **JADE does not support inter-platform mobility**, by **Migration Add-On** does: <http://jade.tilab.com>

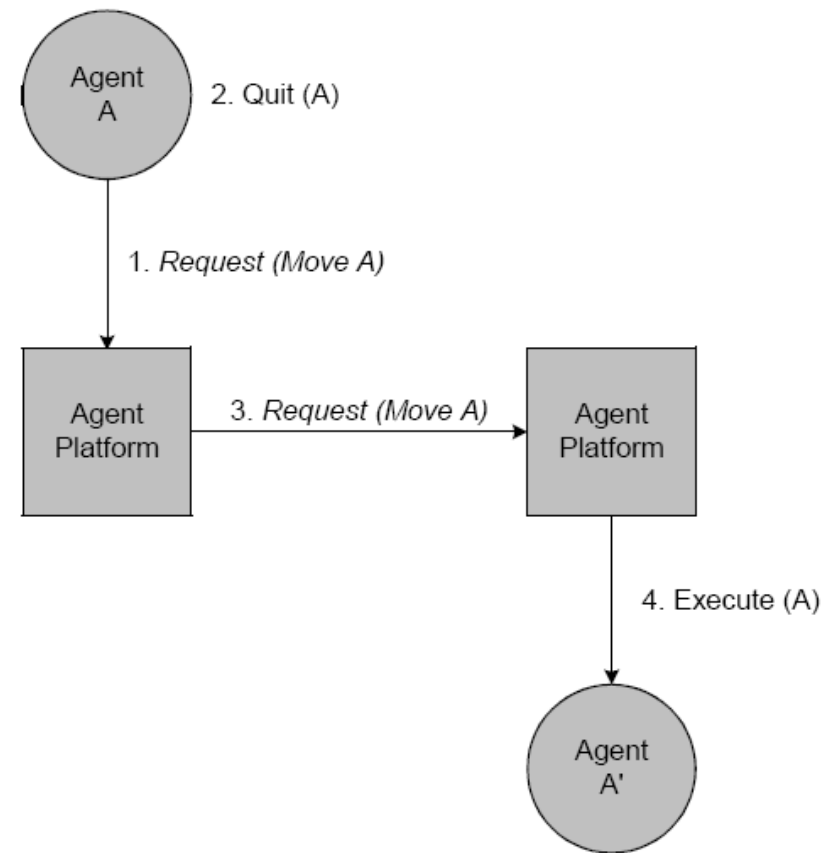
* FIPA Agent Management Specification, <http://www.fipa.org/specs/fipa00023/SC00023K.pdf>





Guidelines for mobility

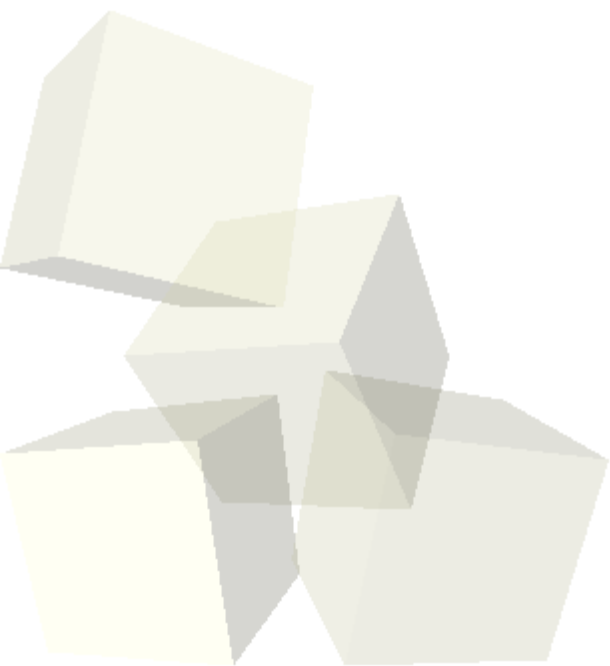
- In order to be able to move, an agent must be **Serializable**
- Mobility can be
 - ♦ self-initiated through the **doMove()** method of the **Agent** class
 - ♦ forced by the **AMS** (following a request from another agent)
- Besides mobility also agent cloning is available (method **doClone()**)



*Simple Mobility Protocol
(example)*



Questions ??

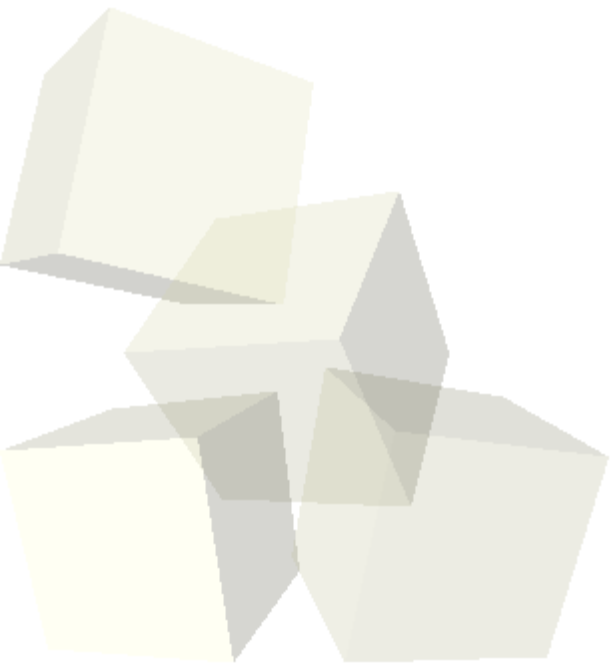




Let us experiment with our app!

➤ Code: `ibspan.lab1.ex7` package

➤ Code: `ibspan.lab1.ex8` package

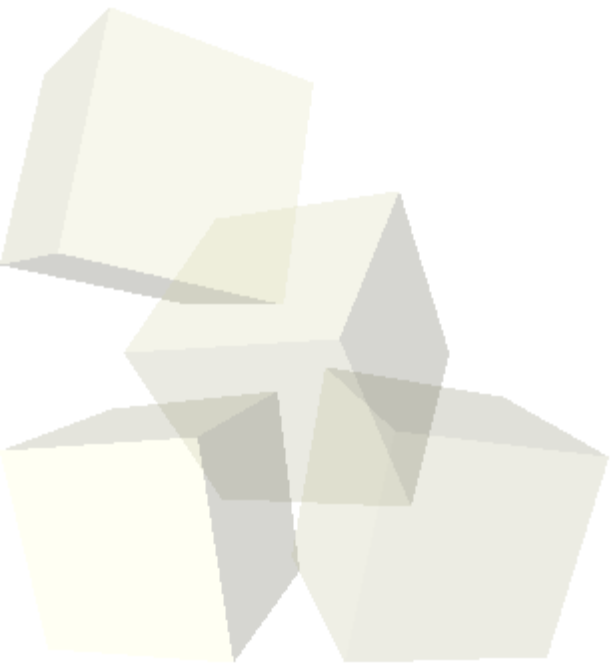




Homework

- Re-implement presented application by use of any of complex behaviours (**SequentialBehaviour**, **FSMBehaviour**)
 - ♦ Documentation
 - **examples.behaviours.ComplexBehaviourAgent**, **examples.behaviours.FSMAgent** classes in JADE package
 - *JADE Programmer's Guide*, <http://jade.tilab.com>
- Think about agents modelling some phenomenon from real world

Questions ??





Acknowledgements

- Michał Szymczak, Paweł Kaczmarek and Mateusz Kruszyk for proposing tutorial content
- Maria Ganzha, Paweł Kobzdej and Marcin Paprzycki for verifying content and lots of comments
- Martin L. Griss and Robert R. Kessler for their tutorial *'Making Java Agents and JBuilder Work for You'*

