AGENT-ORIENTED SOFTWARE ENGINEERING

Intelligent Agents Course

OUTLINE

- Multiagent Platforms
- The Java Agent Development Environment (JADE)
- Agent-oriented Software Engineering
  - The GAIA Methodology
  - Engineering JADE Agents with the Gaia Methodology

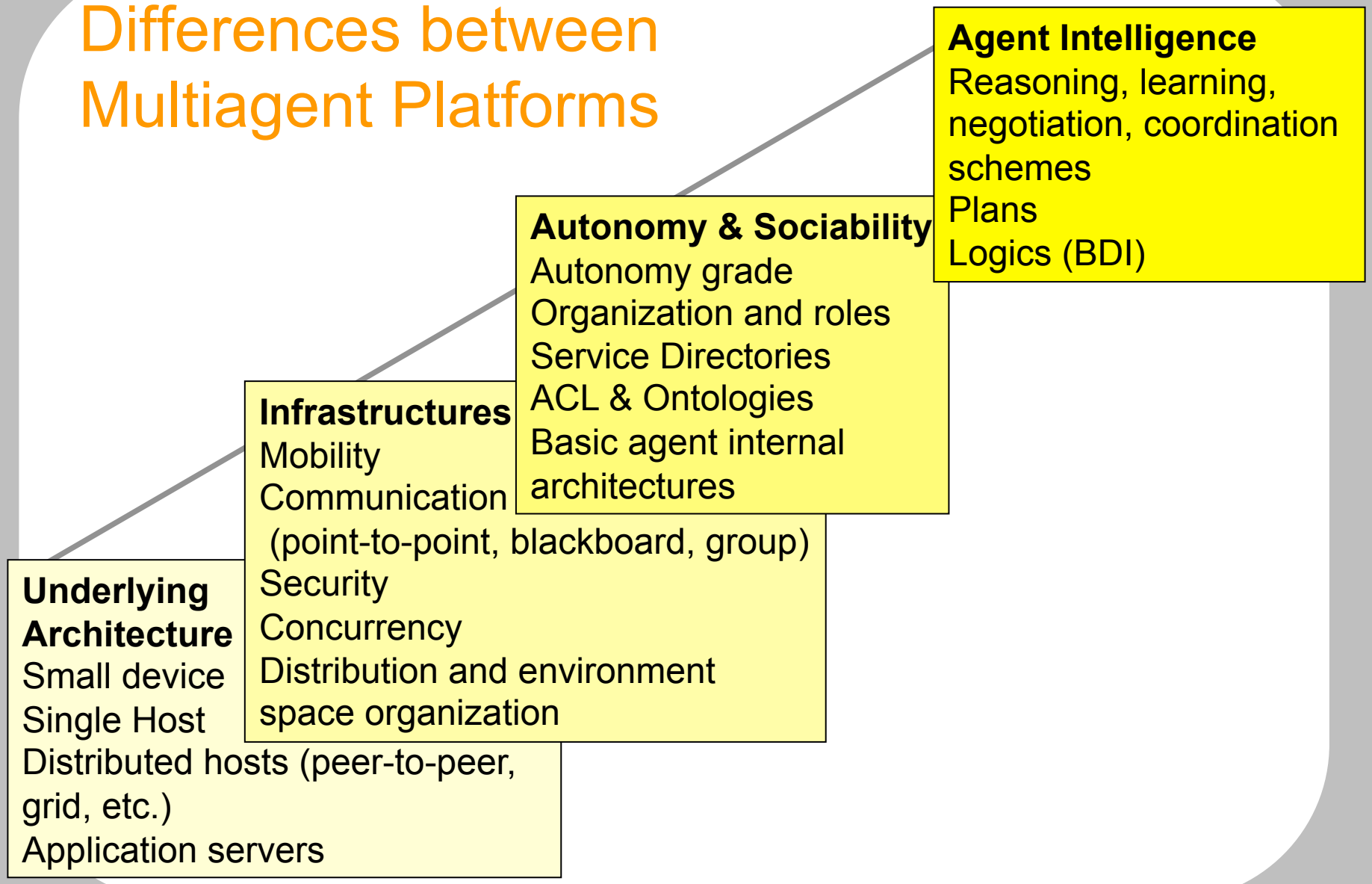
Multiagent Platforms

Platform Types and Examples

- Many experimental multiagent platforms
- Multiagent platforms have not yet reached status of wide spread middleware and methodology (still a research area)

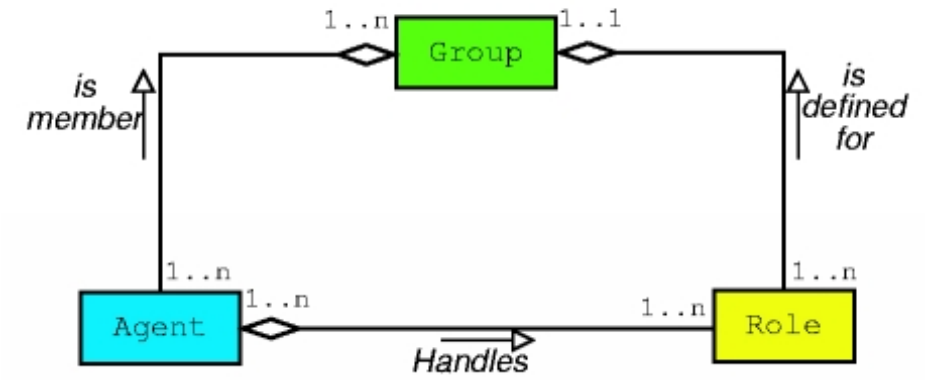
PLATFORM EXAMPLES	
Simulation-oriented	RePast 3
Organization-oriented	MadKit 4
Event-oriented	Cybele
Belief Desire Intention (BDI) Agents	JACK 5
Agent Application Servers	Living Systems Technology Suite
Middleware-oriented	JADE 3.4

Differences between Multiagent Platforms



MadKit

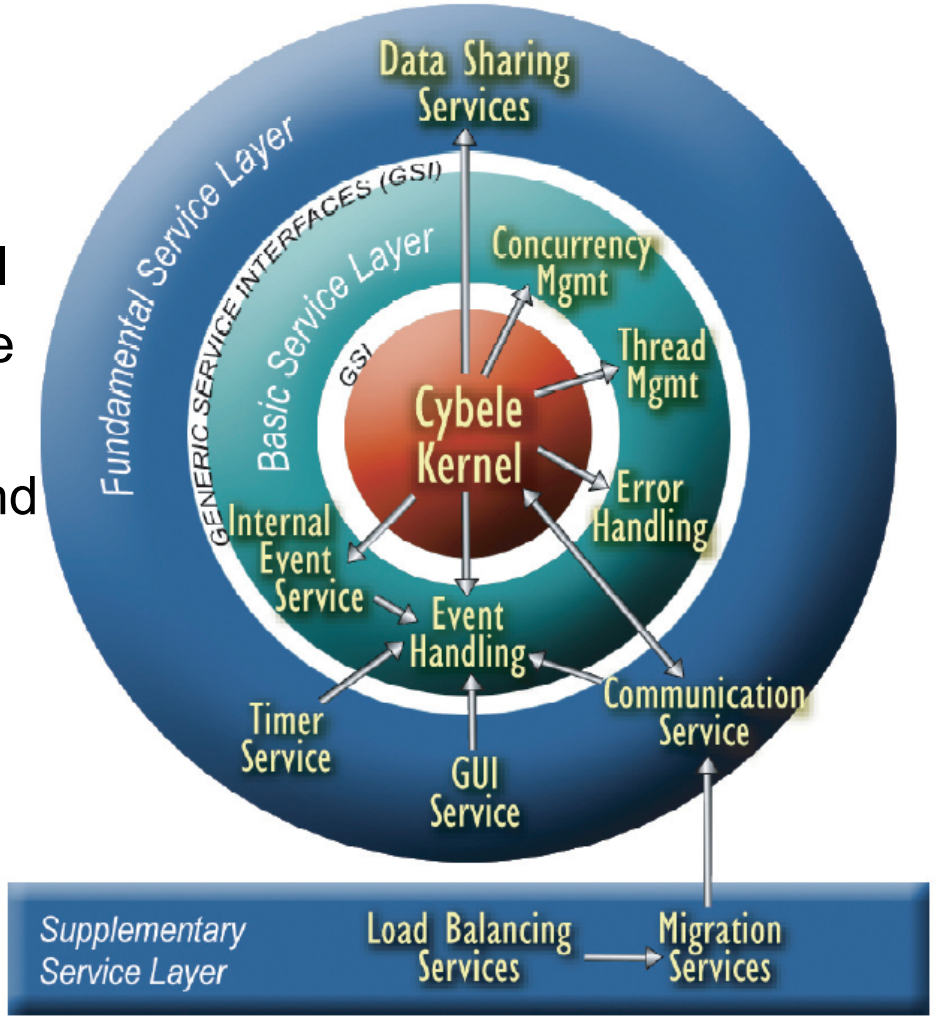
- www.madkit.org (open source, Java)
- Built upon the AGR (Agent/Group/Role) organizational model



- Allows high heterogeneity in agent architectures and communication languages.
- Point-to-point communication
- Can be transparently distributed.
- Runtime environment & API

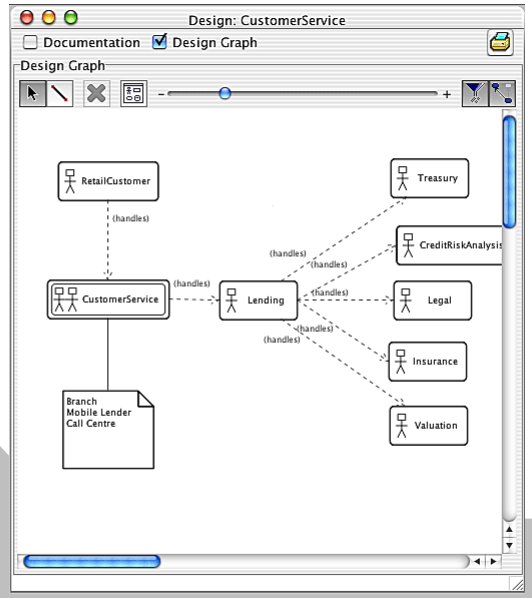
Cybele

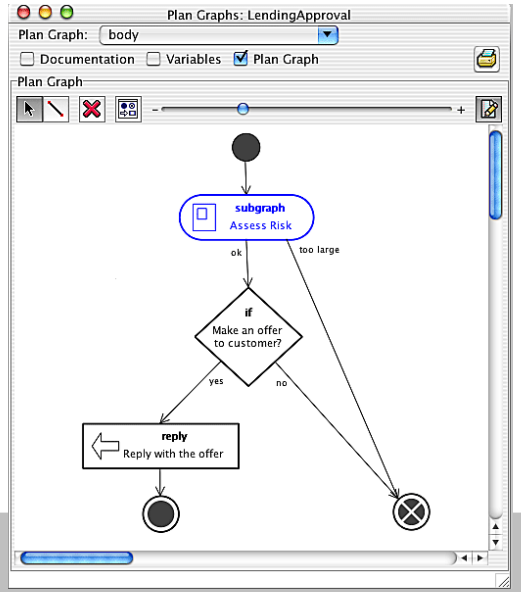
- www.opencybele.org (free & commercial)
- Runtime environment & API
- Service-layered architecture
- Agents are event-driven
- Events: messages, timer and agent internal events
- Messaging:
  - Publish-subscribe
  - Synch./Asynchronous
  - Broadcast, Multicast, Point-to-Point

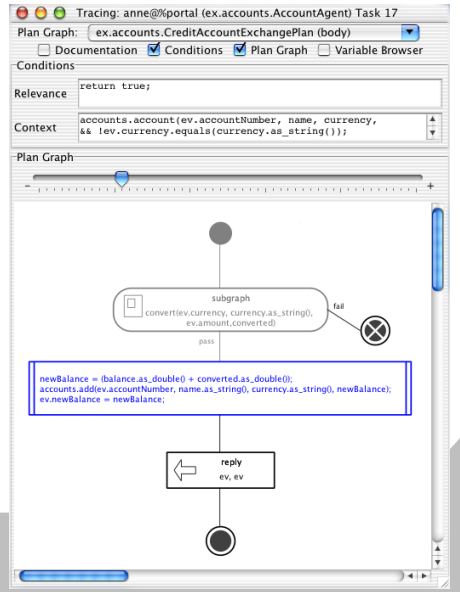


Jack

- www.agent-software.com (commercial)
- Component-based approach.
- Extends Java with agent-oriented concepts, such as:
  - Agents, Capabilities, Events, Plans, Knowledge Bases (Databases), Resource and Concurrency Management
- Supports the BDI (Belief Desire Intention) model
- Runtime environment & API





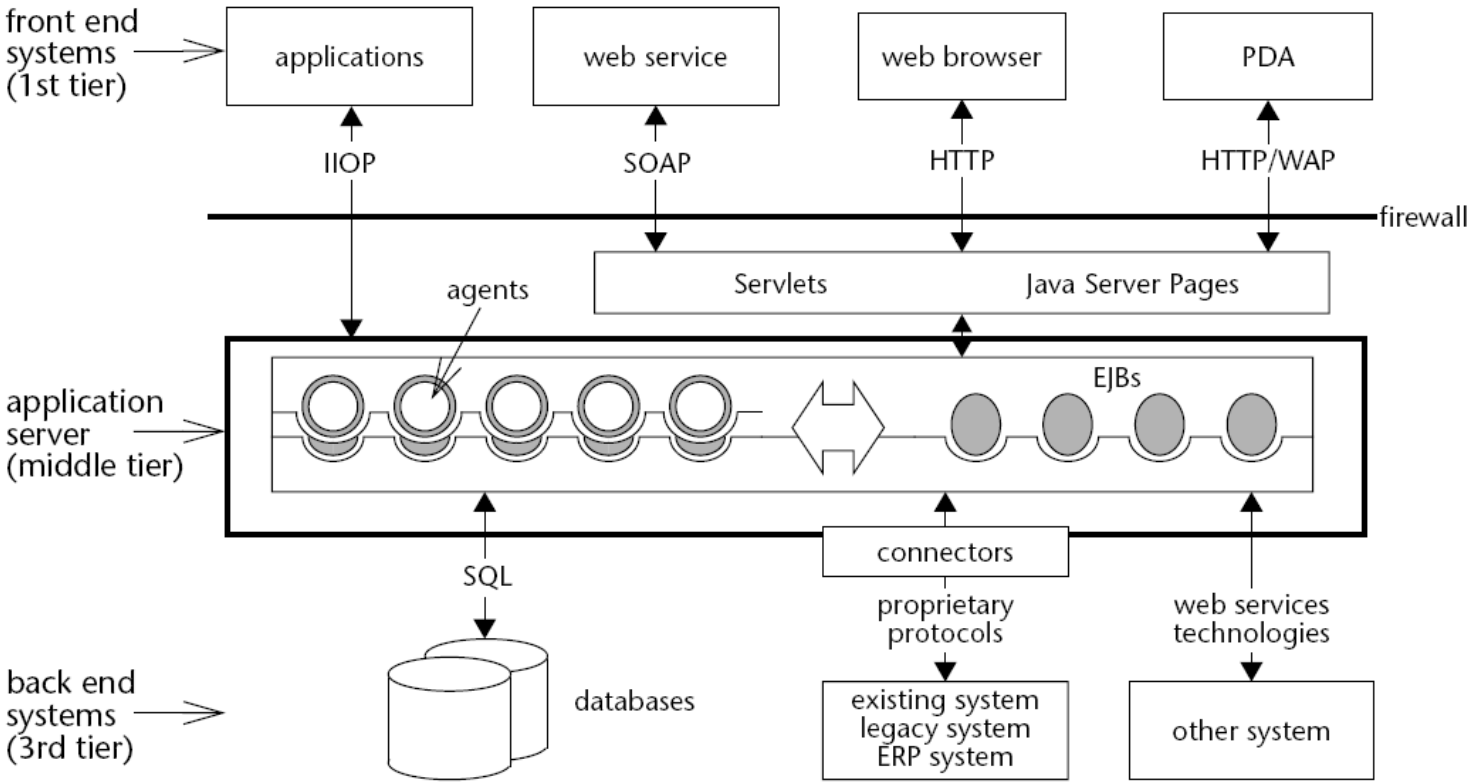


Agent Application Servers

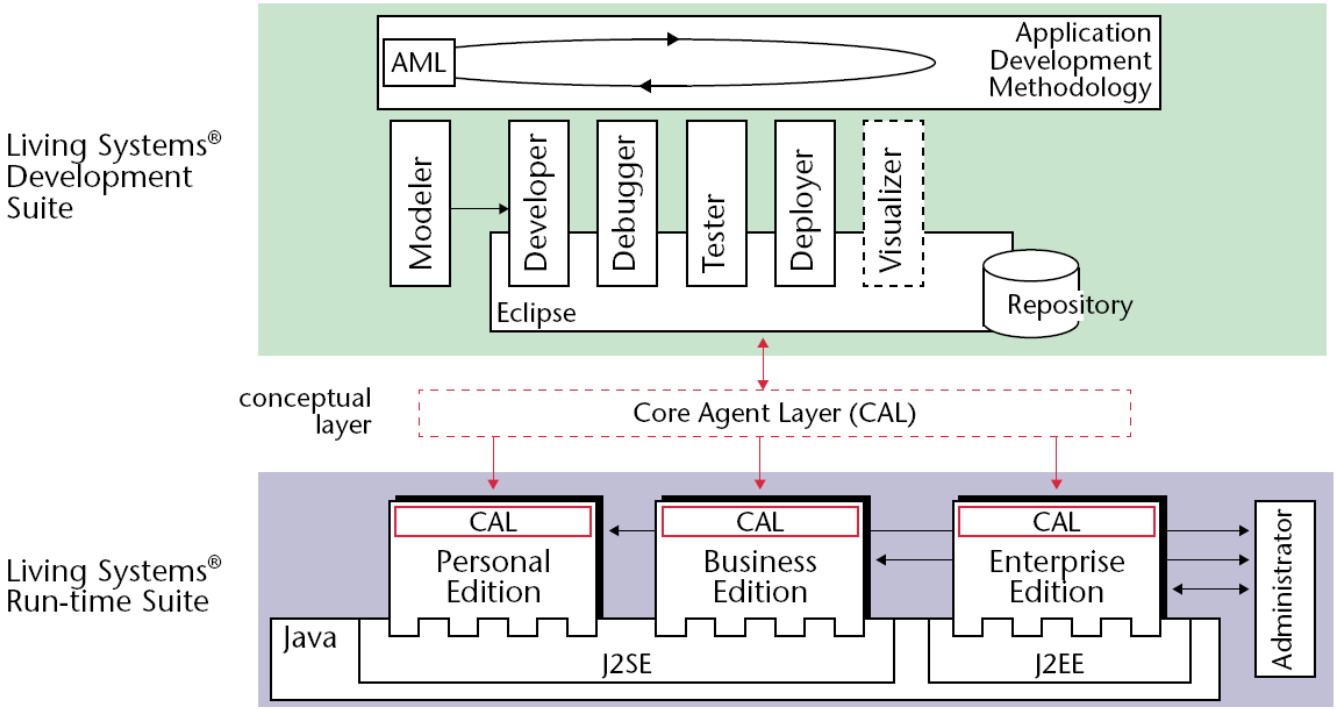
Real-world IT Expectations and Requirements for business systems:

- Integration in existing infrastructures
- Scalability to growing/changing needs

=> Integrate in application servers (e.g. JBoss, WebSphere)



Living Systems Technology Suite



- The **Application Development Methodology** (ADM) is a comprehensive software development process based on RUP (Rational Unified Process) and AML.
- The **Agent Modeling Language** (AML) is an agent-specific extension to UML 2.0

Conclusions

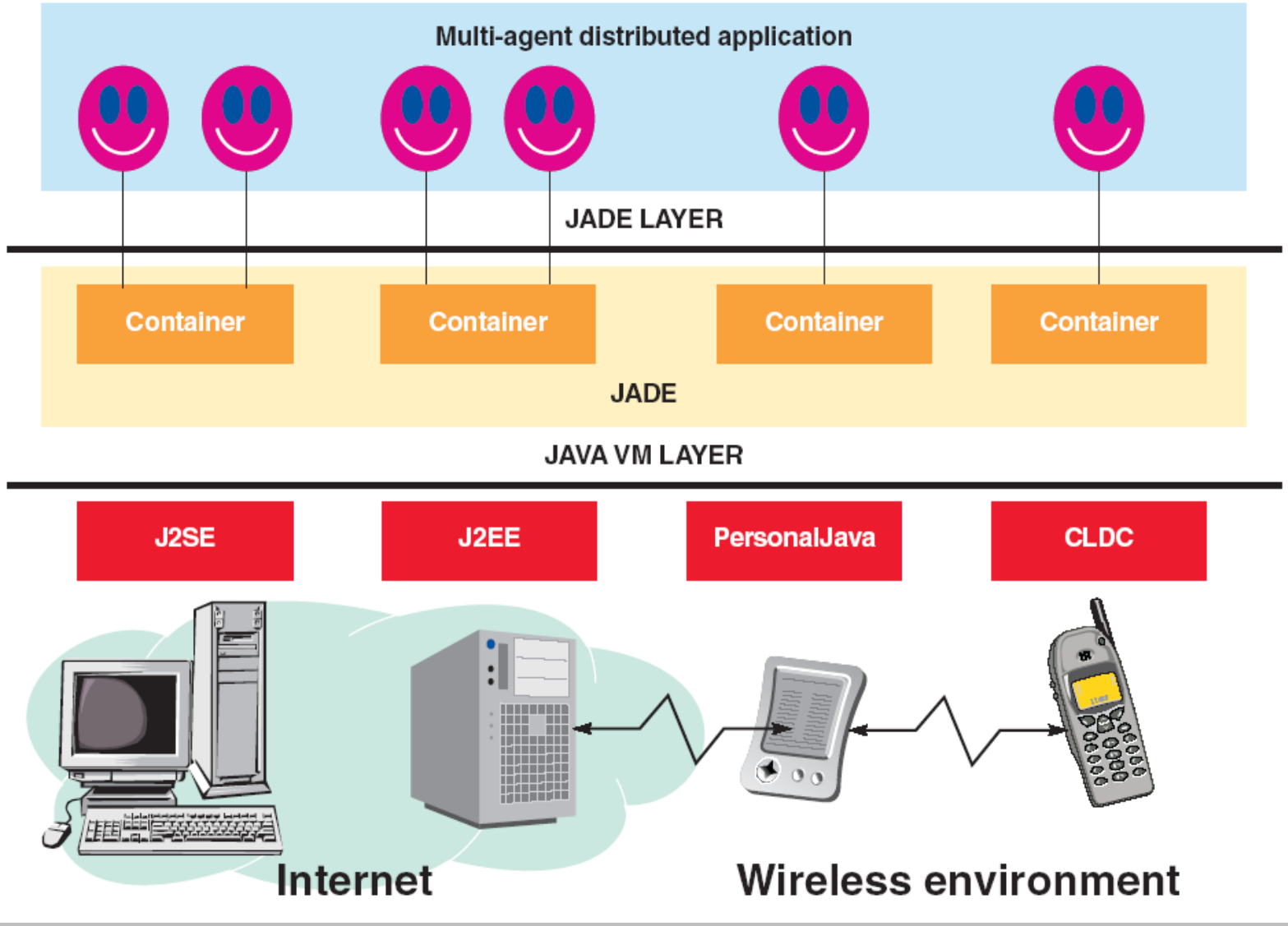
- Need for robust multiagent platforms that should match multiagent models
- Existing platforms:
  - Many experimental projects
  - Open-source and commercial platforms are already showing good results in industry,
  - However, they are not yet adopted by big programmer communities
- Need:
  - integrate with existing accepted middleware
  - Usage of agent-oriented methodologies
  - Convergence with web technology
    (Web services as agents & Semantic web)

A Tutorial on the Java Agent Development Environment JADE

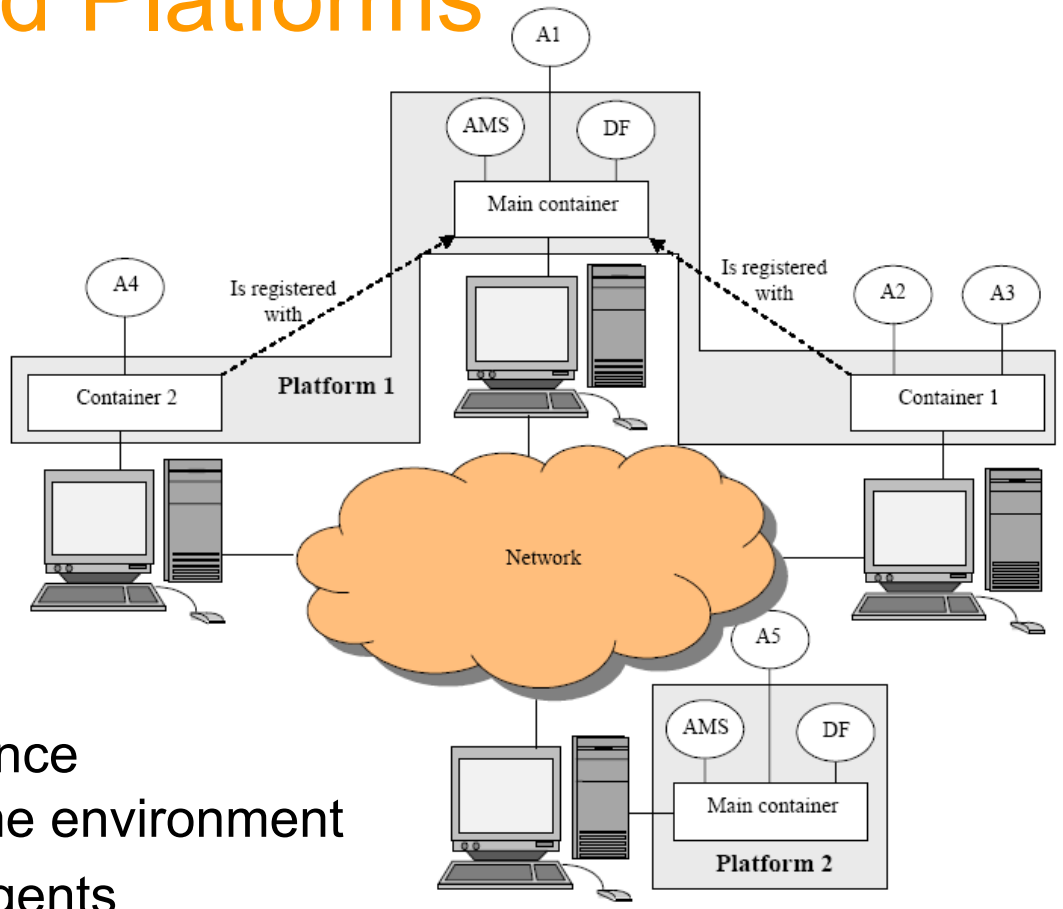
JADE

- jade.tilab.com
- The most used agent platform
- Middleware for multiagent systems
- FIPA compliant (standard for agent platforms)
- Include:
  - Runtime environment
  - Java Library
  - Graphical tools for administration and monitoring

JADE Overview



Containers and Platforms



- **Container**:
  - each running instance of the JADE runtime environment
  - contains several agents
- **Platform**:
  set of active containers

The Main Container

- The Main Container must always be active
- All containers register to it
- It holds two special agents:
  - **Agent Management System** (AMS)
    - Provides the naming service
    - Authority in the platform for creating and killing agents
  - **Directory Facilitator** (DF)
    - Yellow Page service
    - Agents can register services or search for them

Agent Creation

- Define a class that extends `jade.core.Agent` and overrides `setup()`
- Create an instance of this agent class
- Each agent object has a unique identifier AID as an instance of `jade.core.AID`.
  This AID can be retrieved with `getAID()`.

```
import jade.core.Agent;

public class HelloAgent extends Agent {

  protected void setup() {
    System.out.println("Hello! My name is "+ getAID().getName());
  }

}
```

Agent Termination

- When calling its `doDelete()` method, an agent terminates
- On termination, agent's `takeDown()` is called to clean up

```
public class HelloAgent extends Agent {
  protected void setup() {
    System.out.println("Hello! My name is "+ getAID().getName());
    Object[] args = getArguments();
    if (args != null) {
      for (int i = 0; i < args.length; i++) {
        System.out.println("Args[" + i + "] = " + args[i]);
      }
    }
    doDelete();
  }
  protected void takeDown() {
    System.out.println("Agent End!");
  }
}
```

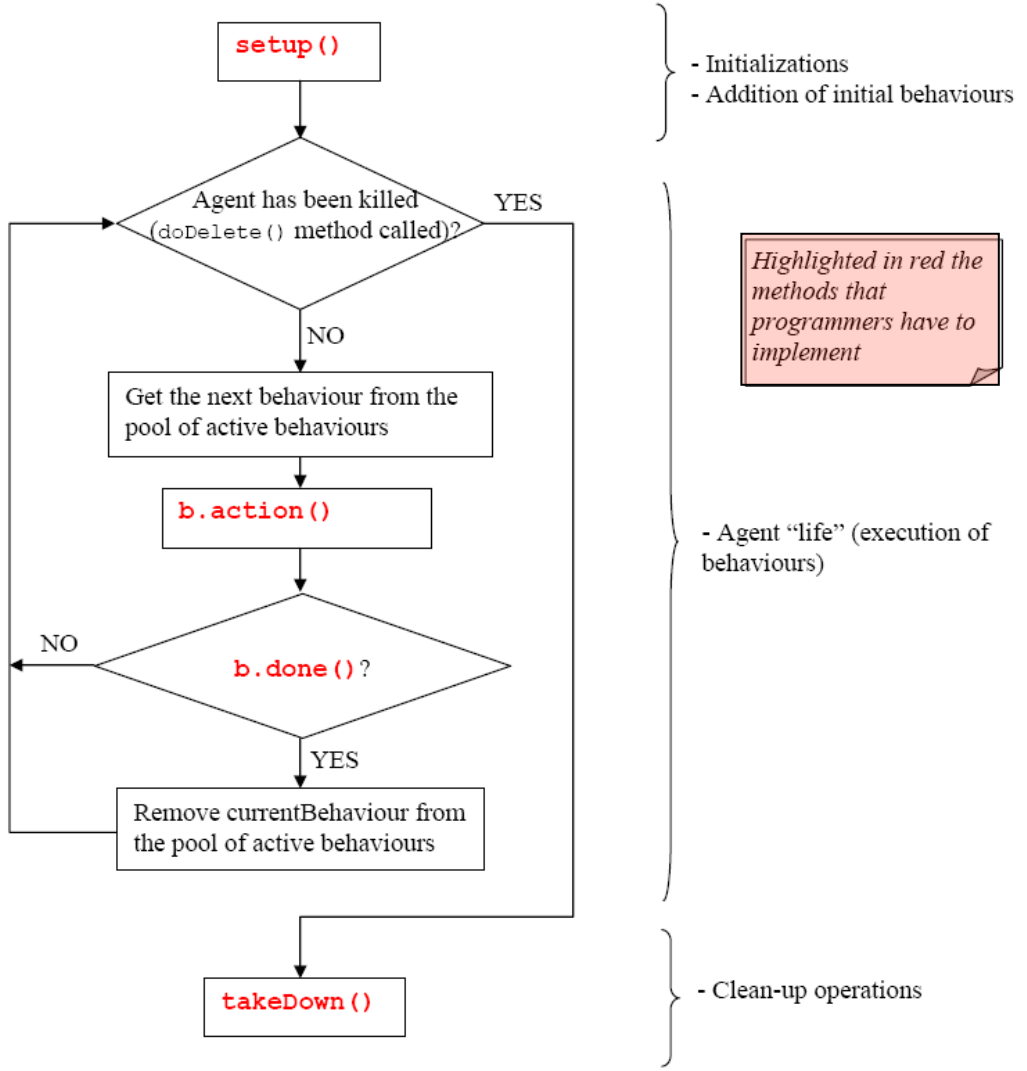
Agent Behaviours

- Agent tasks are defined with behaviour objects
- A behaviour extends `jade.core.behaviours.Behaviour` and re-implements:
  - `action()`: task in itself
  - `done()`: specifies if a behaviour has completed or must be removed from the behaviour pool
- To make an agent execute behaviours, create instances of the corresponding behaviour classes and call `addBehaviour()` on the agent to add them in its pool of actions.
- Behaviours can be added at any time
- If no behaviour, an agent sleeps

Behaviour Scheduling and Execution

- Several behaviours are executed concurrently in an agent
- Cooperative concurrency (and not preemptive!):
  - When a behaviour is executed, its `action()` method is called and runs until it returns. ONLY then a behaviour switch occurs.

The Agent Execution Model

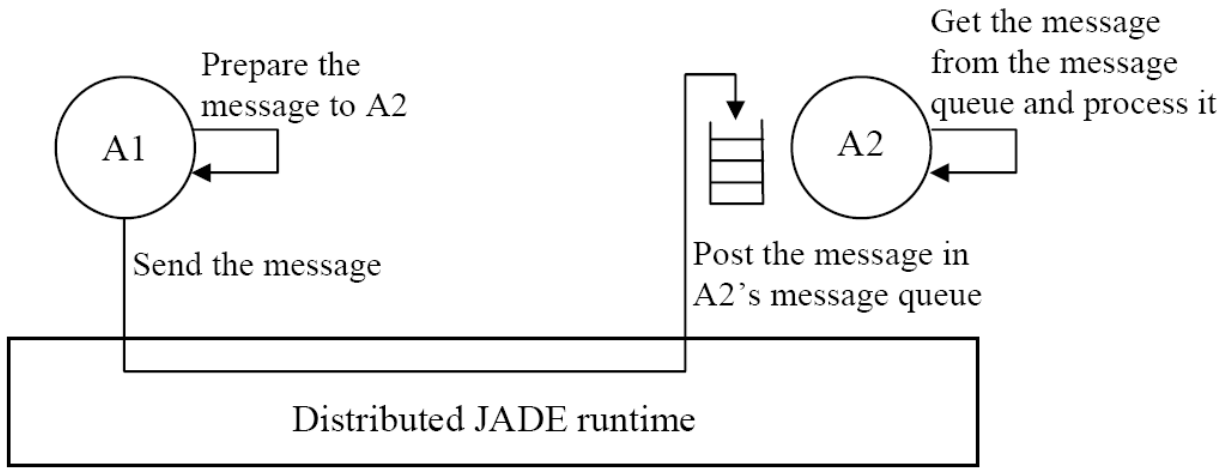


Behaviours Types

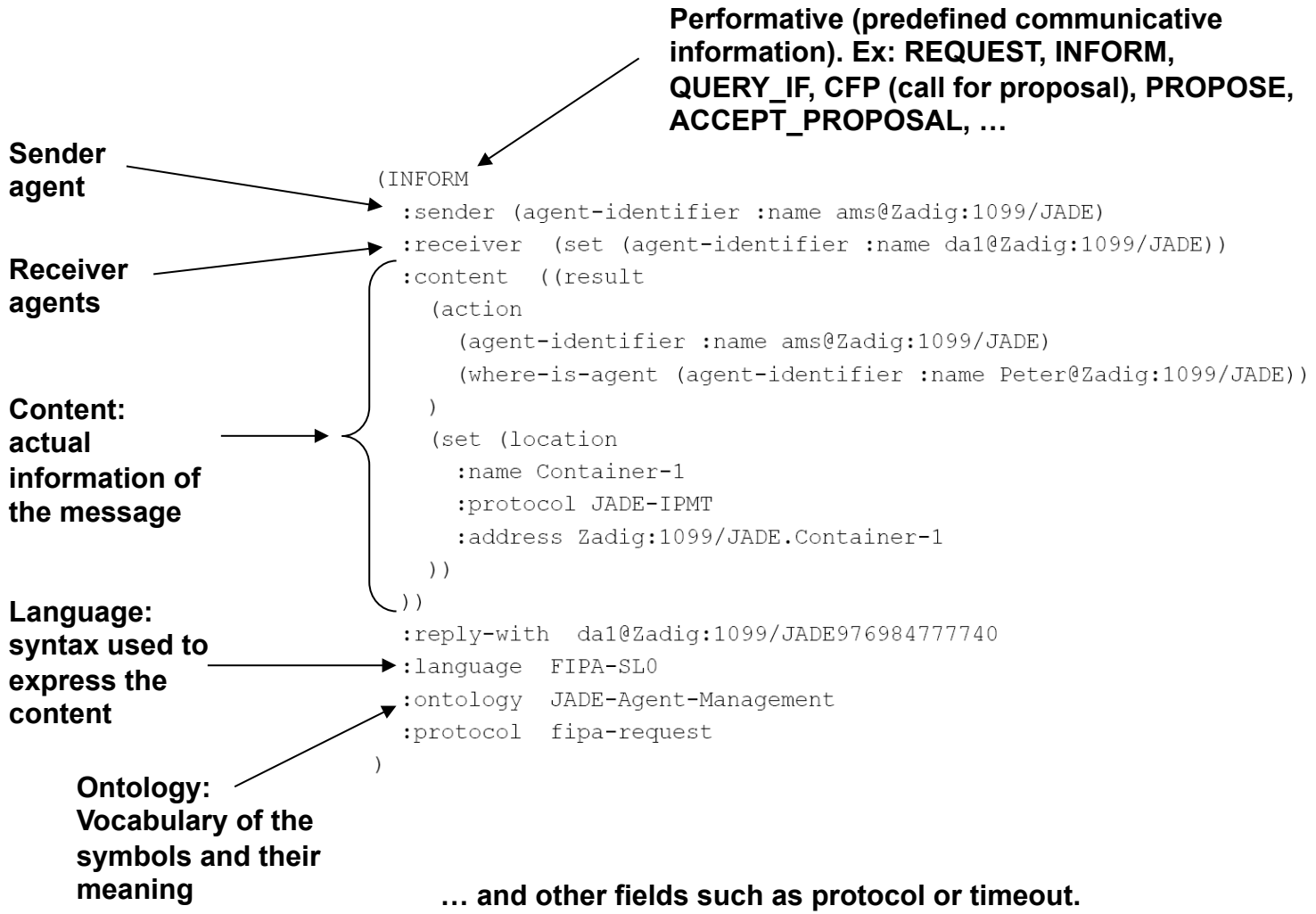
- Main Behaviour types:
  - One-shot behaviour:
    - Method `action()` is executed only once
    - Method `done()` returns simply `true`: completes immediately
  - Cyclic behaviour:
    - Method `action()` executes the same operations each time it is called
    - Never completes
  - Complex behaviour:
    - Method `action()` executes different operations depending on the agent status
    - Completes on a specific condition

Communication Model

- Asynchronous message-passing
- When a message arrives in the message queue, the agent is notified.
- BUT it is up to the receiver agent to decide when to pick up the message: autonomy is respected.



The Agent Communication Language (ACL)



Sending and Receiving Messages

- Sending a message:

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

- Receiving a message:

```
ACLMessage msg = receive();
if (msg != null) {
  // Do something
}
```

- Both are methods of the `Agent` class

Receiving with Templates …

- When you call `receive()`, this retrieves the first message in the mailbox
- To avoid a message to be stolen by another behaviour, you can use a template that matches specific characteristics (e.g. match a specific ontology).

```
MessageTemplate tpl = MessageTemplate.MatchOntology("Weather-forecast");

ACLMessage msg = receive(tpl);
if (msg != null) {
  // Do something
}
```

… and in blocking Mode

- Agent blocking reception is also possible (with or without timeouts): `blockingReceive()`
- This is dangerous, because it can block the whole agent (including all other behaviours)

Blocking a Behaviour waiting for a Message

- Message arrival is not fixed (asynchronous comm.)
- Instead of reading the mailbox in an infinite while loop:
  - A behaviour can call `block()`
  - It is put in a blocked state and removed from the agent pool (This does not block the agent).
  - When a message arrives, all behaviours are inserted back in the agent pool and can therefore read the message

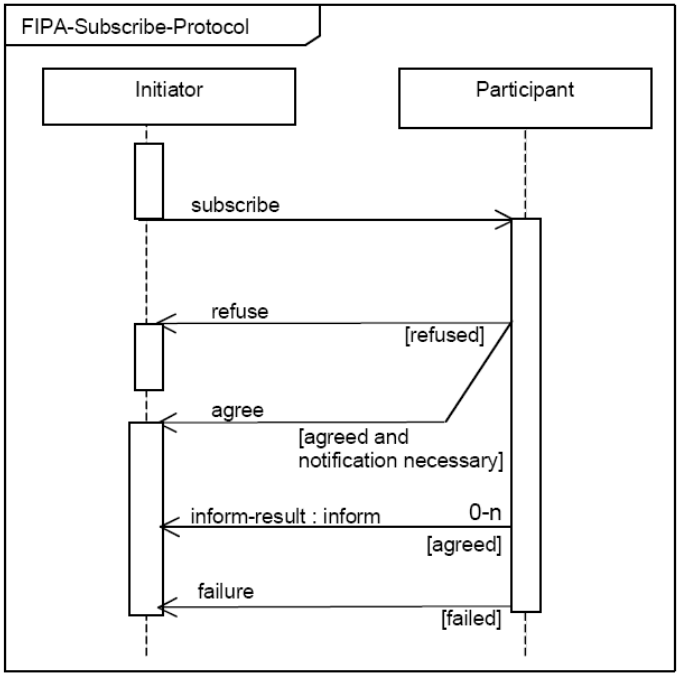
```
MessageTemplate tpl = MessageTemplate.MatchOntology("Weather-forecast");

Public void myAction() {
  ACLMessage msg = myAgent.receive(tpl);
  if (msg != null) {
    // Do something
  }
  else {
    block();
  }
}
```

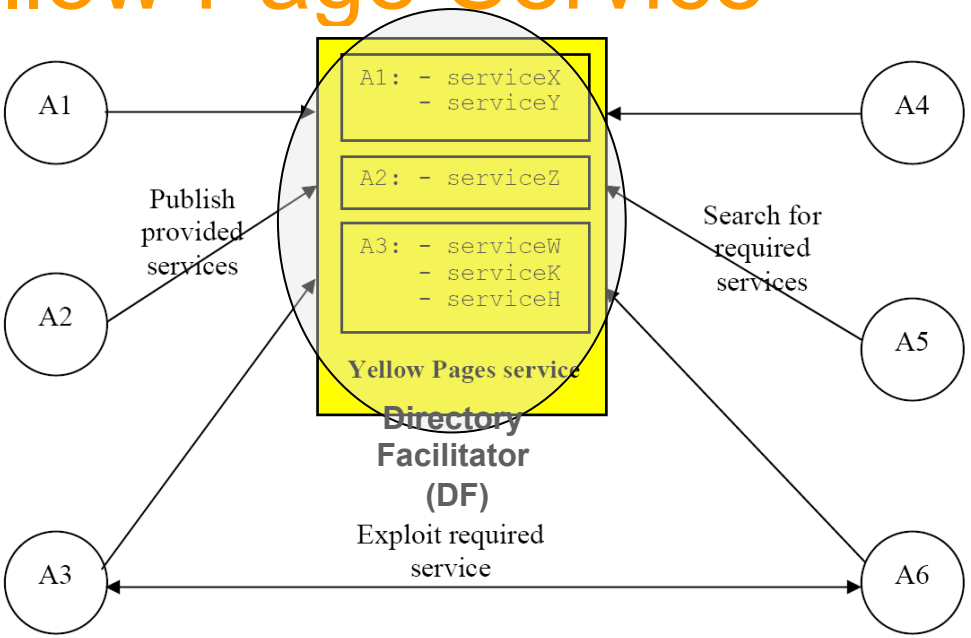
Common pattern of Reading messages:
1. Read messages within a behaviour
2. and block the behaviour if a message is not available

Interaction Protocols

- Predefined sequences of messages define conversations, which are called **Interaction Protocols**
- The package `jade.proto` models standard interaction protocols (e.g. the FIPA-Subscribe-Protocol) and classes to define new protocols.
- These classes provide:
  - the flow of messages
  - timeouts (if any)
  - callback methods that should be redefined for actions to be taken when messages arrive or when timeouts occur.



The Yellow Page Service



- The Directory Facilitator (DF) is an agent implementing the yellow pages service
- Static utility methods of `jade.domain.DFService`: `register()`, `modify()`, `deregister()` and `search()`
- A subscription mechanism exists
- Services must be described with the class `ServiceDescription`:
  service type, name, language, ontology, interaction protocol, …

Mobility Support

- Agents can migrate between containers
- Intra-platform: mobility only within the same platform
- "Hard Mobility":
  - Status: an agent stops its execution, moves to a remote container and restarts its execution at the point it was stopped
  - Code: if the code is not on the new host, it is retrieved
- To move, an agent must be `Serializable`