Java Agent DEvelopment Framework

What is JADE?

- A software framework
  - that simplifies the implementation of multi-agent systems
- The middle-ware for Multi-Agent System (MAS)
  - target users: agent programmers for MAS
  - agent services
    - life-cycle, white-page, yellow-page, message transport
  - tools to support debugging phase
    - remote monitoring agent, dummy agent, sniffer agent
  - designed to support scalability
    - (from debugging to deployment)
    - from small scale to large scale
- Implements Foundation for Intelligent Physical Agents (**FIPA**).
- Fully implemented in Java
  - distributed under LPGL

What is JADE for?

JADE is a platform for running agents

- To support:
  - An asynchronous agent programming model
  - Communication between agents either on the same or different platforms
  - Mobility, security, and other utilities
  - Develop custom applications to exploit Agent metaphor
  - Hook agent framework to JAVA GUI applications

Advantages of JADE

- Support for mobile devices
- Graphical runtime environment
- FIPA compliant
- Strong development team
- Open source
- Ontologies
- Predefined interaction protocols as specified by FIPA
- New interaction protocols can be easily introduced

Why is JADE Important?

- JADE Hides FIPA From The Programmer!

- No need to implement the Agent Platform and Agent Services

- No need to implement agent-management ontology and functionalities
  - An agent is registered with the Agent Platform
  - It is given a name and a unique address
  - The Agent class provides a simplified interface to access the services of the Directory Facilitator (registration, searching, …)

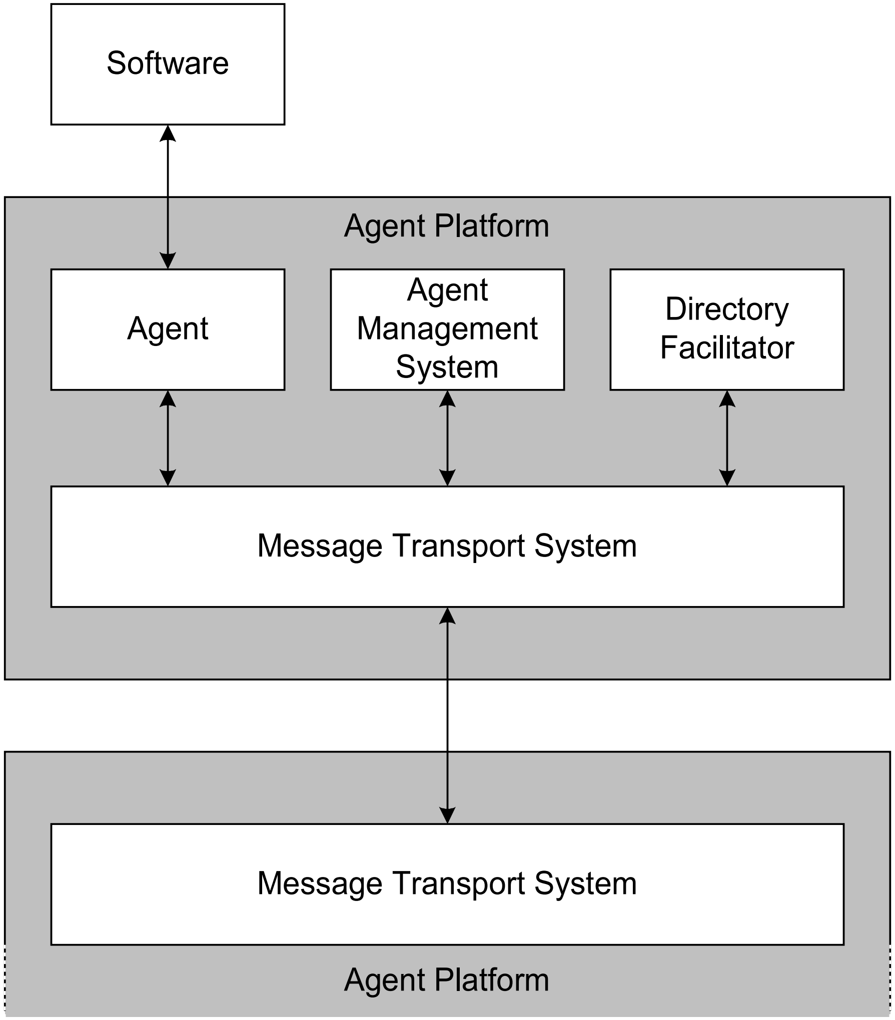
- No need to implement Message Transport and Parsing
  - Automatically (and possibly efficiently) done by the framework when sending/receiving messages

- Interaction Protocols must only be extended via handle methods

What is FIPA?

- Founded as a non profit Swiss organization in 1996
- Goal
  - Define a full set of standards for implementing systems within which agents could execute (agent platforms)
  - Specifying how agents should communicate and interact
- A number of standards were proposed, however, never gained commercial support
- Agent platforms adopting the "FIPA standard"
  - The JADE agent programming platform
  - The Spyse agent programming platform
  - JACK (Agent Oriented Software Group)
  - The April Agent Platform and Language
- FIPA is currently an IEEE standards committee
- FIPA2000 is the current approved FIPA specifications

Agent Management Model



- The Agent Platform supports the Message Transport System.
- The Message Transport Service is the default communication method between agents on different Agent Platforms.
- The Agent Platform supports routing tasks.

Agent Management

- Agent Management System (AMS) and Directory Facilitator (DF) have a common subset of actions:
  - Register
    - JADE automatically calls the register and deregister methods with the default AMS respectively
  - Deregister
    - But JADE provides control to do these tasks programmatically for special cases.
  - Search
    - Directory Facilitator (DF) is the agent who provides the default yellow page service, provides the capability to search agents with specific capability
  - modify
    - AMS supports the modification of the agents during run-time

FIPA Agent Execution Model

- Agent is autonomous
  - It completely controls its thread of execution
    - Private proxy of the life-cycle manager
  - It decides itself when to read received messages and which messages to read/serve
    - The transport mechanism fills a private queue but it does not call the agent code (no automatic callback)
- Agent needs concurrency
  - Can engage in multiple simultaneous conversations
  - Can execute several concurrent tasks

Agent Naming

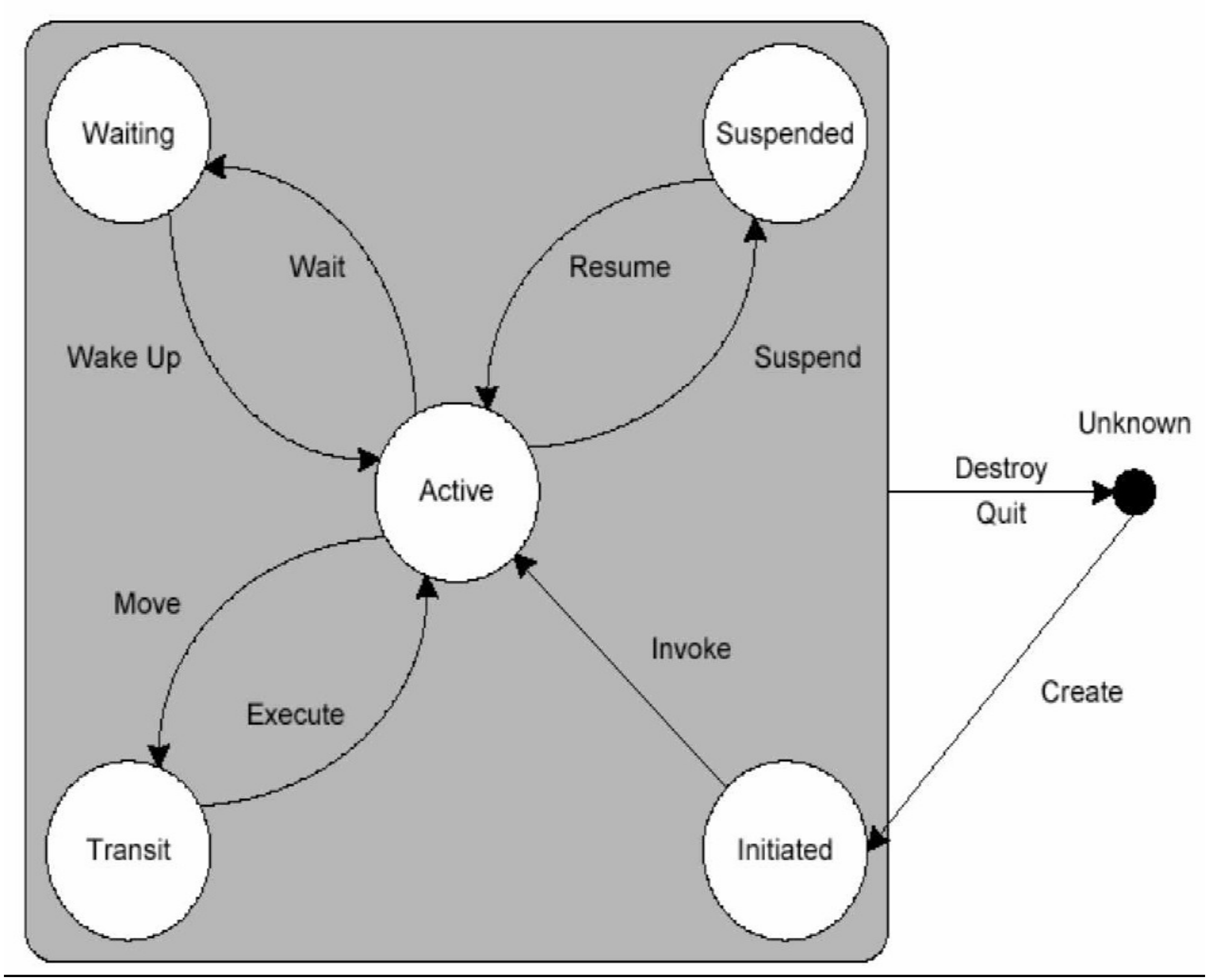
- Agent is identified through an extensible collection of parameter-value pairs, called an Agent Identifier (AID).

- AID comprises:
  - A name.
  - Other parameters, such as transport addresses, name resolution service addresses, and so on.

- The name of an agent is immutable; the other parameters in the AID of an agent can be changed.

- A given agent may support many methods of communication and can put multiple transport address values in the :addresses parameter of an AID.
- a globally unique name.
  - By default JADE composes this name as the concatenation of the local name – i.e. the agent name provided on the command line – plus the '@' symbol, plus the home agent platform identifier – i.e. <hostname> ':' <port number of the JADE RMI registry> '/' 'JADE')
  - i.e. GITIAgent@JadeServer:1099/JADE

Agent Lifecycle



JADE states

- **AP_INITIATED :** the Agent object is built, but hasn't registered itself yet with the AMS,
- **AP_ACTIVE :** the Agent object is registered with the AMS,
- **AP_SUSPENDED :** the Agent object is currently stopped.
- **AP_WAITING :** the Agent object is blocked,
- **AP_DELETED :** the Agent is definitely dead.
- **AP_TRANSIT:** a mobile agent enters this state while it is migrating
- **AP_COPY:** this state is internally used by JADE for agent being cloned.
- **AP_GONE:** this state is internally used by JADE when a mobile agent has migrated

Agent Communications - Messaging

- Message transport protocols
  - IIOP
  - HTTP
- Three encodings
  - String encoding
  - bit-efficient encoding
  - XML
- Each message protocol supports the three kinds encoding
- Only the message envelope depends on transport protocol.
  - Abstract Frame-based envelope syntax, mapped to protocol-dependent concrete syntaxes.

Message Structure

- A message is made up of two parts:
  - envelope expressing transport information
  - body comprising the ACL message of the agent communication.

- For message interpretation by an agent:
  - ACL semantics are defined only over the ACL message delivered in the message body of a FIPA message.
  - Information in the message envelope is supporting information only. How and if this information is used to by an agent is undefined by FIPA.

How much of FIPA is hidden by JADE?

- No need to implement the Agent Platform
  - AMS, DF, and ACC executed at start-up
- No need to implement agent-management ontology and functionalities
  - An agent is registered with the AP within its constructor
    - it is given a name and an address
  - The Agent class provides a simplified interface to access the services of the DF (registration, searching, …)
- No need to implement Message Transport and Parsing
  - automatically (and possibly efficiently) done by the framework when sending/receiving messages
- Interaction Protocols must only be extended via handle methods

Open Source

- JADE is available in Open Source under LGPL license from
  - The latest version of JADE is 3.5 released June 2007

LIST OF ACRONYMS AND ABBREVIATED TERMS

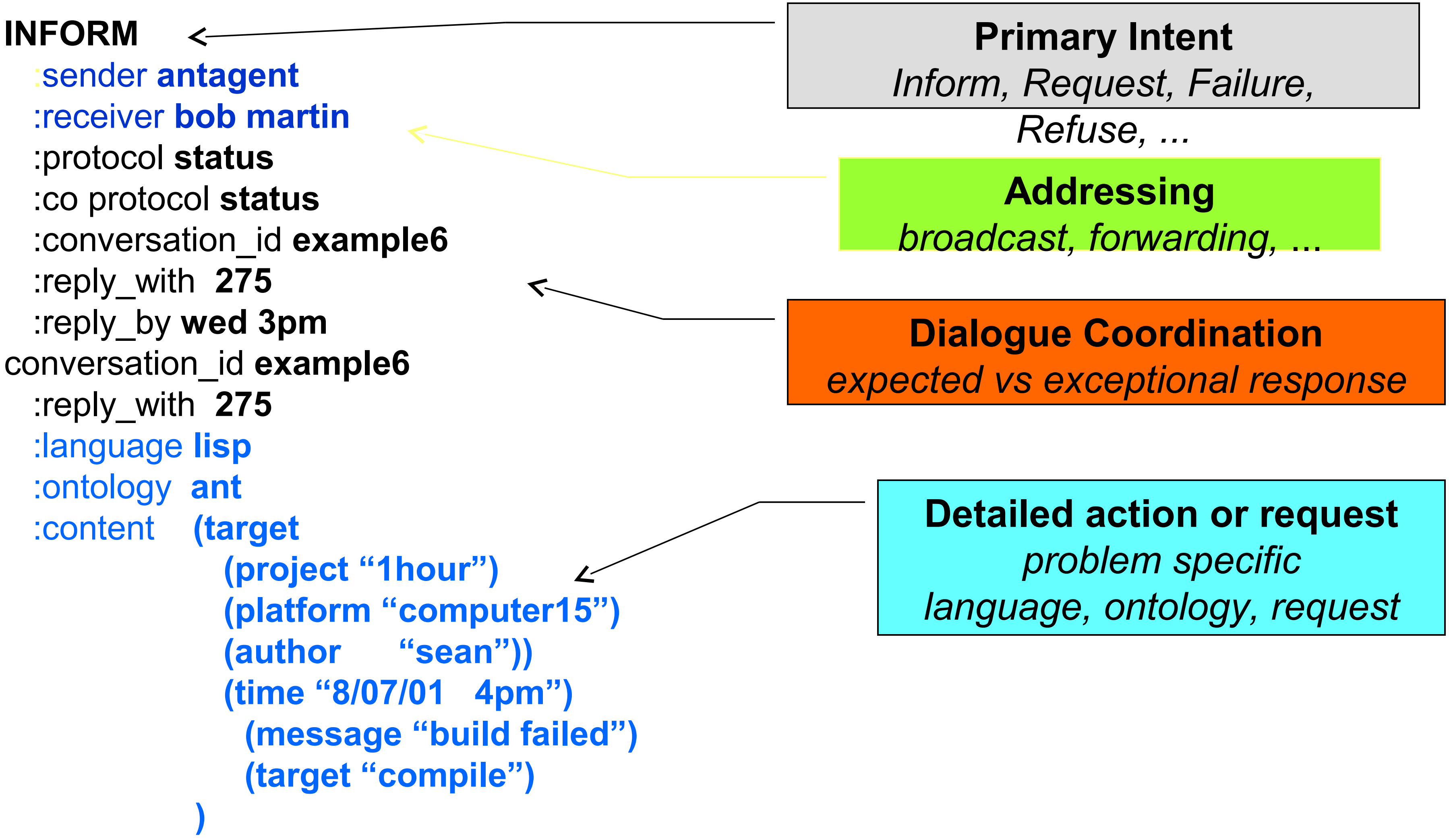
- ACL - Agent Communication Language
- AID - Agent Identifier
- AMS - Agent Management Service. According to the FIPA architecture, this is the agent that is responsible for managing the platform and providing the white-page service.
- AP - Agent Platform
- DF - Directory Facilitator. According to the FIPA architecture, this is the agent that provides the yellow-page service.
- FIPA - Foundation for Intelligent Physical Agents
- GUID - Globally Unique Identifier
- HTML - Hyper Text Markup Language
- HTTP - Hypertext Transmission Protocol
- IDL - Interface Definition Language
- IIOP - Internet Inter-ORB Protocol
- IOR - Interoperable Object Reference
- MTP - Message Transport Protocol. According to the FIPA architecture, this component is responsible for handling communication with external platforms and agents.
- ORB - Object Request Broker
- RMA - Remote Monitoring Agent. In the JADE platform, this type of agent provides a graphical console to monitor and control the platform and, in particular, the life-cycle of its agents.
- RMI - Remote Method Invocation

Terminology

Agent Communication Language (ACL)

- Semantically defined standard vocabulary and syntax
  - Agent exchange
  - String-based messages
- An ACL Message
  - Structured message, targeted for flexible communication
  - Performative (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

Agent Communication Example



```
INFORM
   :sender antagent
   :receiver bob martin
   :protocol status
   :co protocol status
   :conversation_id example6
   :reply_with  275
   :reply_by wed 3pm
conversation_id example6
   :reply_with  275
   :language lisp
   :ontology  ant
   :content    (target
                  (project “1hour”)
                  (platform “computer15”)
                  (author     “sean”))
                  (time “8/07/01   4pm”)
                    (message “build failed”)
                    (target “compile”)
               )
```

Primary Intent
Inform, Request, Failure, Refuse, ...

Addressing
broadcast, forwarding, ...

Dialogue Coordination
expected vs exceptional response

Detailed action or request
problem specific
language, ontology, request

Content Language (CL)

- Defines
  - proper syntax for content expressions
  - number of operators such as the logical connectors AND and OR
- Semantic Language (SL)
  - Describes actions, propositions, and identifying expressions
  - Well Formed Formulas, logical operators built in to SL. In addition there are some interesting existential quantifiers such as B, "believes", and U, "uncertain".

Agents and Agency

- What is an Agent?
  - There seem to be as many definitions as there are users of it
- Dictionary:

a·gent *n.*

1. One that acts or has the power or authority to act.
2. One empowered to act for or represent another: *an author's agent; an insurance agent.*
3. A means by which something is done or caused; instrument.
4. A force or substance that causes a change: *a chemical agent; an infectious agent.*
5. A representative or official of a government or administrative department of a government: *an FBI agent.*
6. A spy.
7. *Linguistics.* The noun or noun phrase that specifies the person through whom or the means by which an action is effected.

Motivation

- *Most important development in IT in past 10 years?*
- *How to build software in a dynamic, open, culturally diverse world?*
- *Intelligent personal assistants*

Are these agents?

- Web crawler
- Virus
- Email handler

The Agent Metaphor

- Human surrogate/servant
- Emphasis on communication
- Autonomy
- Persistence

Parent technologies

- artificial intelligence
- software engineering
- distributed systems
- organisational science
- databases
- economics
- game theory
- artificial life

What is an agent?

- Software agents are small, efficient pieces of code that combine the job execution properties of a scripting language with the communications capabilities of objects. Agents carry out specific, repetitive, and predictable tasks on behalf of a computer user, business process, or software application. And like software objects, agents have programmable properties and behaviours.

  from the WWW in 1996

What is an Agent?

"encapsulated computer system, situated in some environment, and capable of flexible autonomous action in that environment in order to meet its design objectives" (Wooldridge)

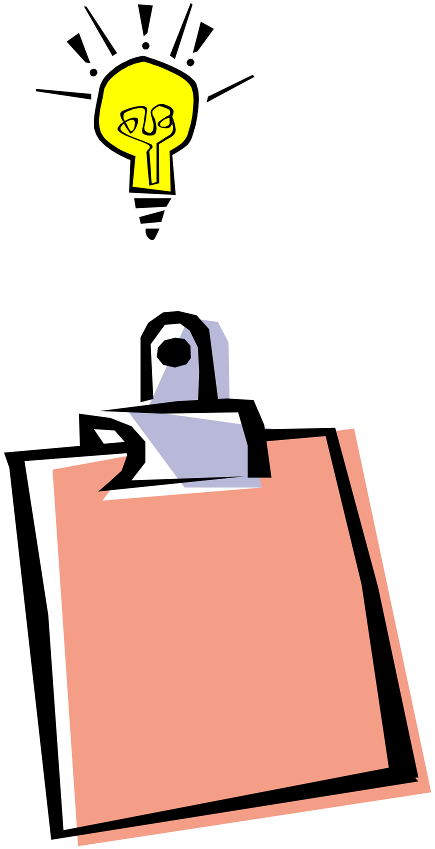
- ***control over internal state and over own behavior***
- ***experiences environment through sensors and acts through effectors***
- ***reactive: respond in timely fashion to environmental change***
- ***proactive: act in anticipation of future goals***

Why agents are useful

- Ability to perform in an open, dynamic, unpredictable environment
- Potential for exploiting the Internet
- Ease of interfacing with legacy software
- Aid to model a complex environment
- Potential for more intelligent software

Agent Advantages

- ‘Natural’ Abstraction level
- Easy to specify knowledge (for analysts)
- Useful breakdown of problems
- Modularity
- Coping with legacy systems



What agents are not

- Wizards
- Scripting languages
- Programming languages
- Neural Networks
- Pure rule-based systems
- Objects

What About Human Agents?

- Some characteristics of human agents
  - carries out some task for someone else (master/slave)
  - is given goals but has considerable leeway on how to achieve these goals
  - probably moves around
  - observes features of her or his environment
  - interacts with, and perhaps changes to some extent, his or her environment of objects and (possibly) other agents
  - has a memory to store instructions, goals and other information
  - (hopefully) exhibits some intelligence
- Keywords
  - goals, observations, autonomy, intelligence, mobility

How About Software Agents?

- Artificial Intelligence Agents
  - First to use agent term in the computer world
  - Usually individual 'creatures' living in an environment where they interact in some way
  - Agents are more or less autonomous
  - Most AI agents do not interact with other agents in the same environment
  - Exhibit some kind of intelligence
  - A good example: robot interacting with a real world environment

Example of An Ideal Agent

- Imagine an autonomous automatic pilot controlling an aircraft
  - Goal of safely landing at some airport.
- Requires:
  - Pro-activeness
    - Plan how to achieve this goal (may include pre-compiled plans)
    - Generate subsidiary goals (e.g., ascend to an altitude of 30,000 feet, then proceed due north at a speed of 400 knots . . .).
  - Autonomous
    - Execute its plans, but not blindly
    - Maintains local state
    - Responds appropriately if unforeseen circumstances (e.g., a change in weather conditions, a fault in the aircraft, a request from airtraffic control)
  - Reactiveness
    - Can not spend hours deliberating, must respond in a timely manner
  - Social ability
    - Auto-pilot cooperates with air-traffic controllers and perhaps other aircraft in order to achieve its goals

Trends in the Agent World

- Agent standards
  - FIPA, JAS, OMG
- Agent initiatives
  - Agentcities
  - TILAB and Motorola: JADE/LEAP for mobility
- Robust agent platforms
  - JADE + J2EE -> BlueJADE
- Fledgling work on Semantic Web
  - DAML + OIL

References

- Agent technology
  - FIPA:
  - JADE:
  - BlueJADE:
  - Agentcities:
  - CoABS GRID
- AI technology
  - Weka:
  - Lucene:
  - Jess:
- Agent groups
  - Utah:
  - Santa Cruz:
  - UMBC:
  - DARPA

Protocol: Allowed Message Patterns

Standards

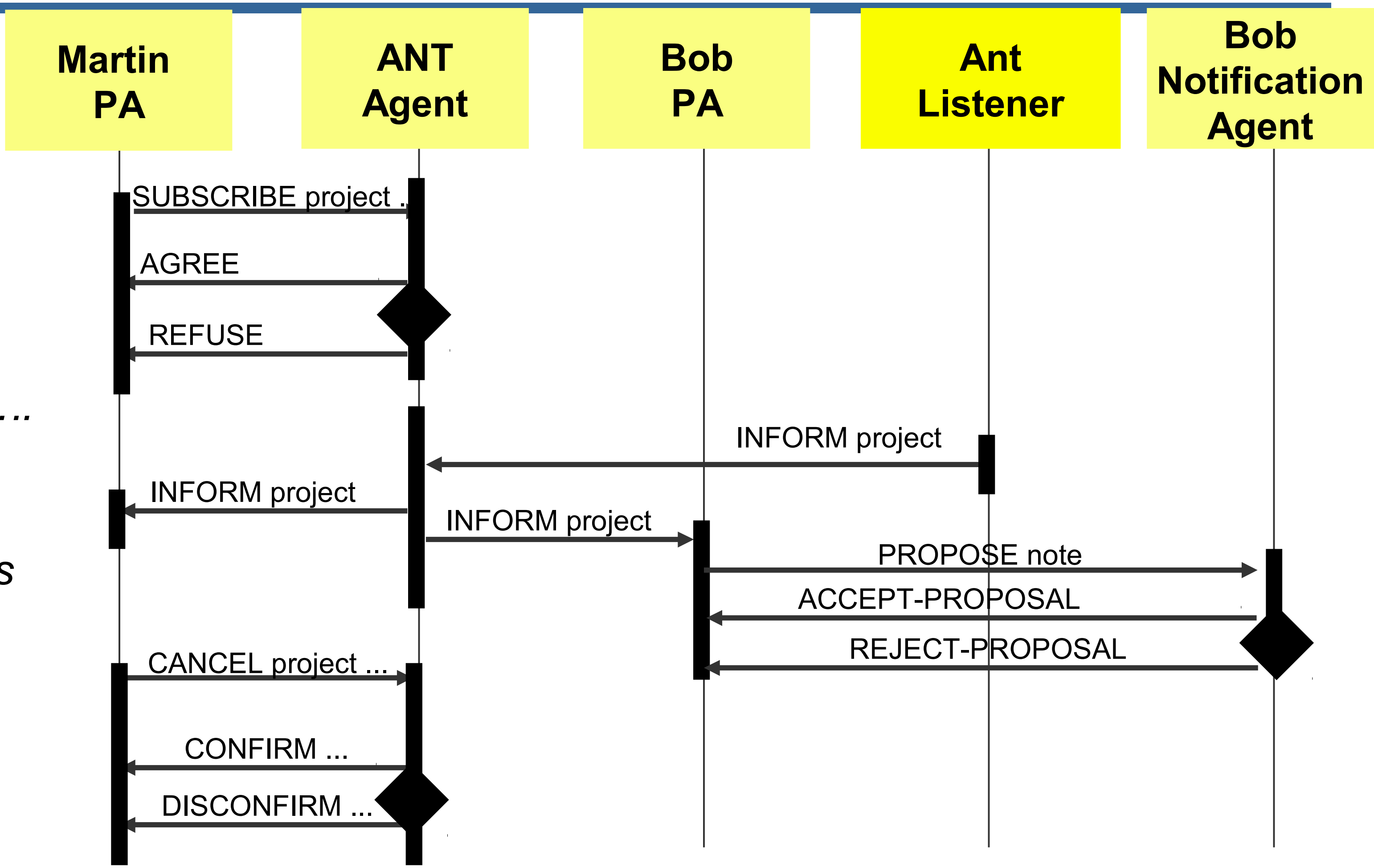
contract-net,
auctions,
planning,
negotiation,
management, ….

Tools

State Machines
Workflow
UML

Models

Meetings
Organizations
Context



JADE Intro

- Introduction to JADE concepts

Mapping Theory into Design

- *Agents are autonomous.*
- *Agents are social entities.*
- *Messages are speech acts, not invocations.*
- *An agent can say “no” and “I don’t care”.*

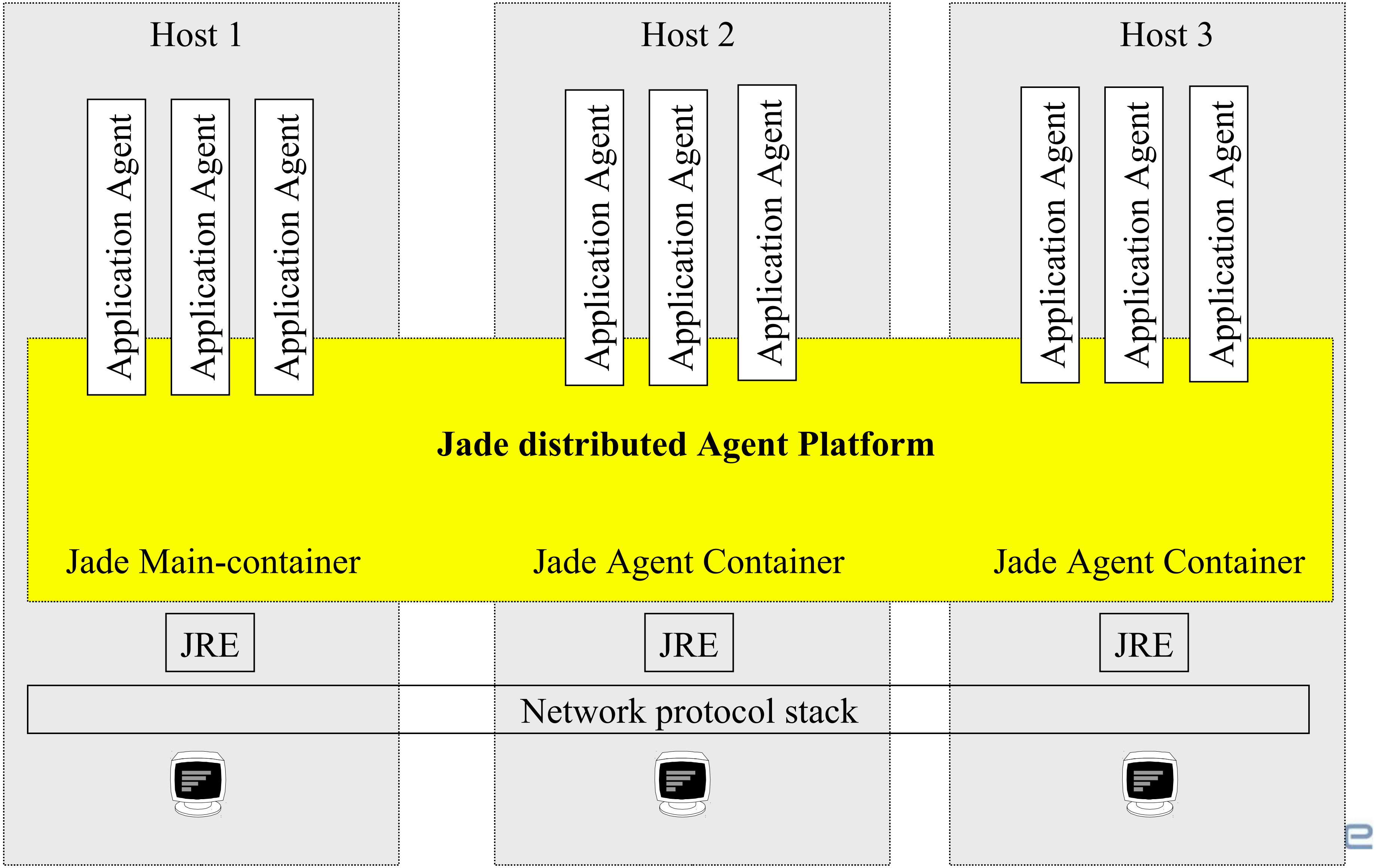
- Agents are active objects.
- Intra-agent concurrency is needed.
- Asynchronous messaging must be used.
- Peer-To-Peer messaging (built over distributed objects Client/Server interactions).

JADE API

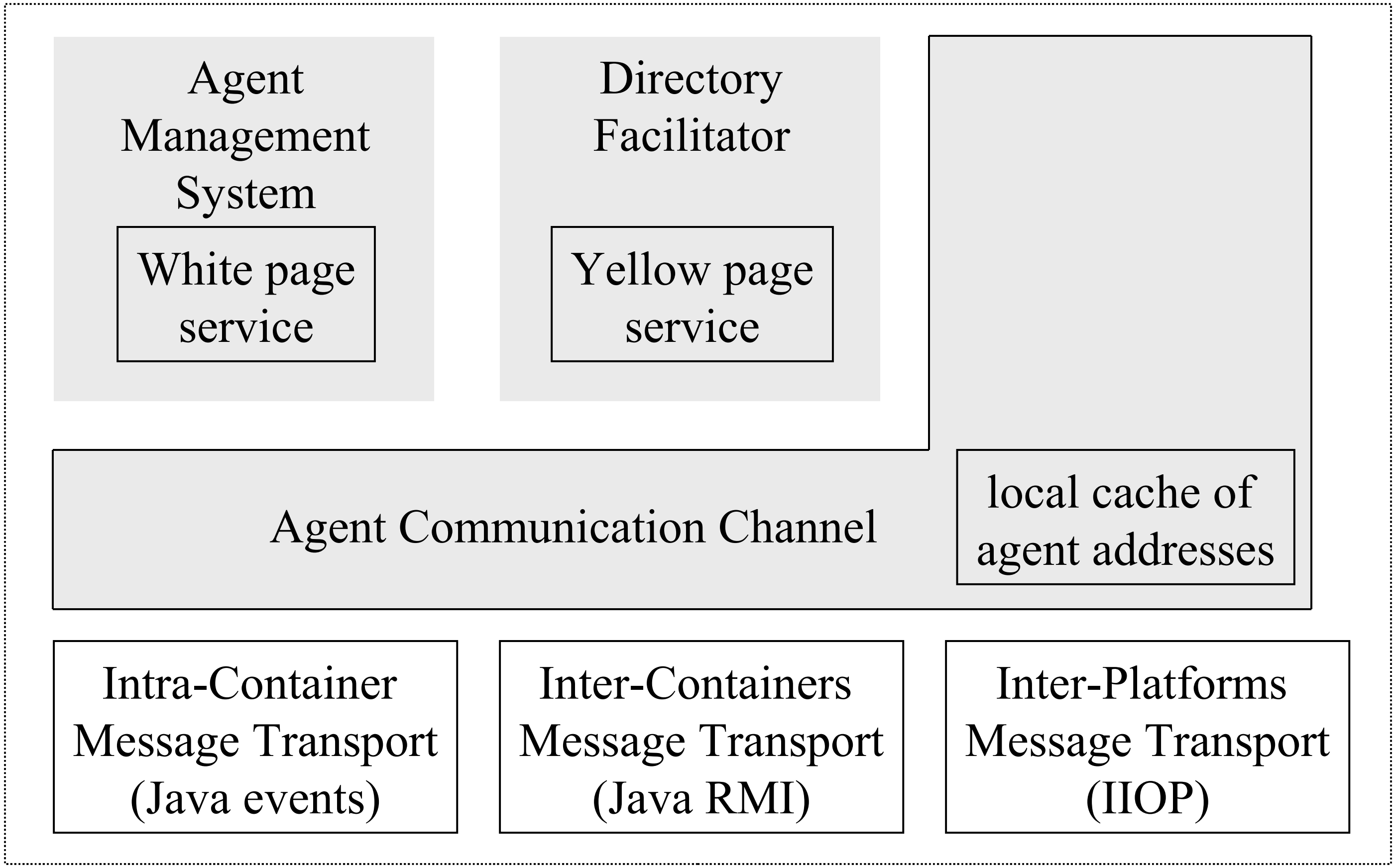
- The JADE framework includes a library of interaction protocols and generic agent behaviours, that must be customized for the specific application needs in order to create the agent capabilities

- Quite Complex 100+ classes
- Don't need to know most of it
- Examples give you most of what you need
- Key Classes:
  - jade.core.Agent: base class for agents
  - jade.core.behaviours.* : Behaviours
  - jade.lang.acl.ACLMessage: Message class
  - jade.core.AID: Agent identifer

JADE Agent Platform



JADE Agent Container

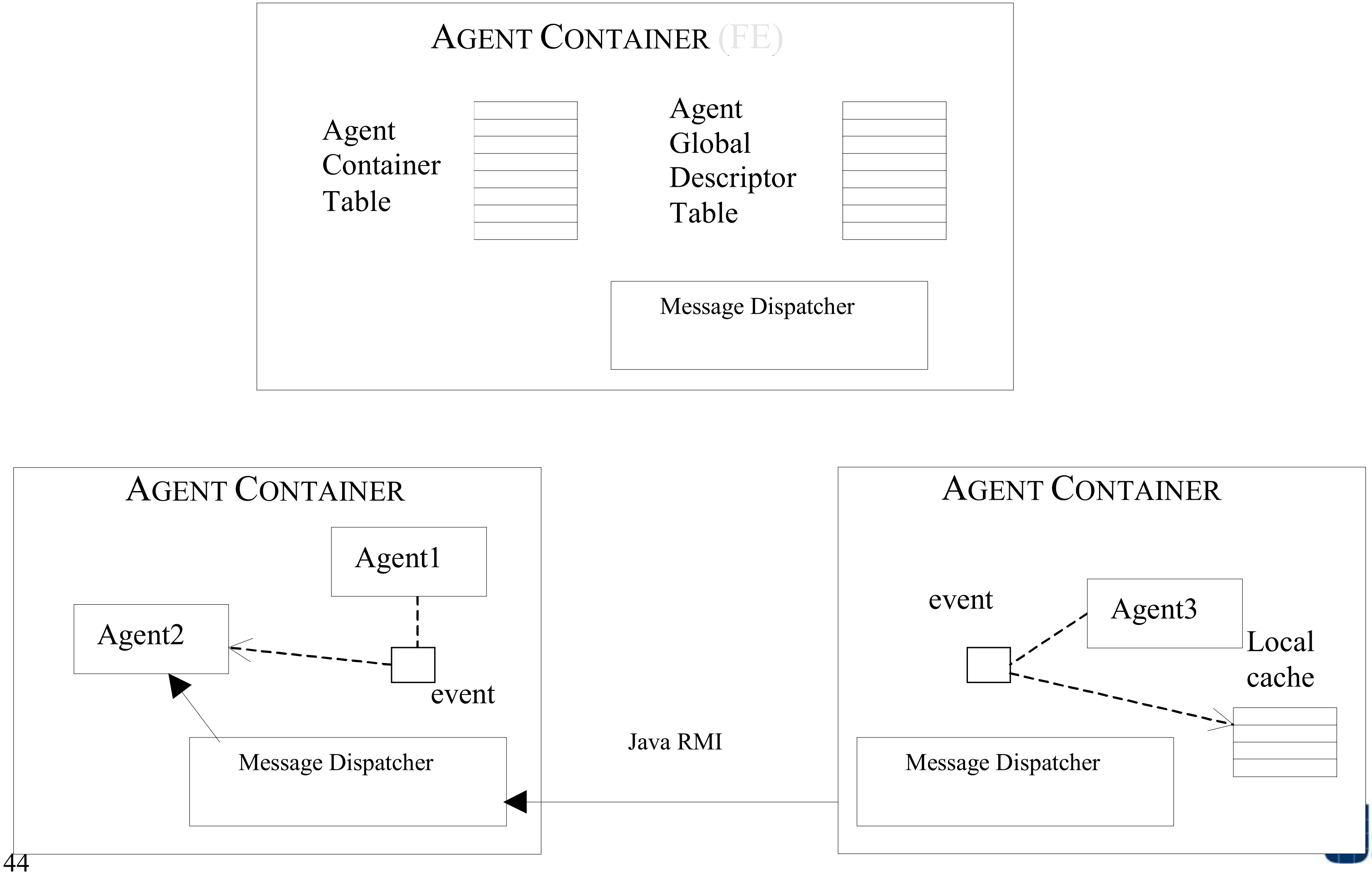

Agent Management System
White page service
Directory Facilitator
Yellow page service
Agent Communication Channel
local cache of agent addresses
Intra-Container Message Transport (Java events)
Inter-Containers Message Transport (Java RMI)
Inter-Platforms Message Transport (IIOP)

Note: The internal architecture of a JADE main container is similar, but it does not contain the AMS, the DF, and the IIOP modules.

JADE Communication

- Every agent has a private queue of ACL messages created and filled by the JADE communication sub-system
- Designed as a chameleon to achieve the lowest cost for message passing
  - the mechanism is selected according to the situation
  - the overheads depend on the receiver’s location and the cache status

JADE Communication



Communication Model

- Library of interaction protocols
- Framework directly supports parsing
  - Envelope parser (in JADE 2.0)
  - Agent Communications Language (ACL) parser
  - Content Language (CL) parser
    - including support for uuencoded Java serialization
  - Ontology checker
- Framework can be extended by the user
  - Support to define/save/load new ontologies
  - Interface for CL Parser/Encoder
  - Automatically used by the framework

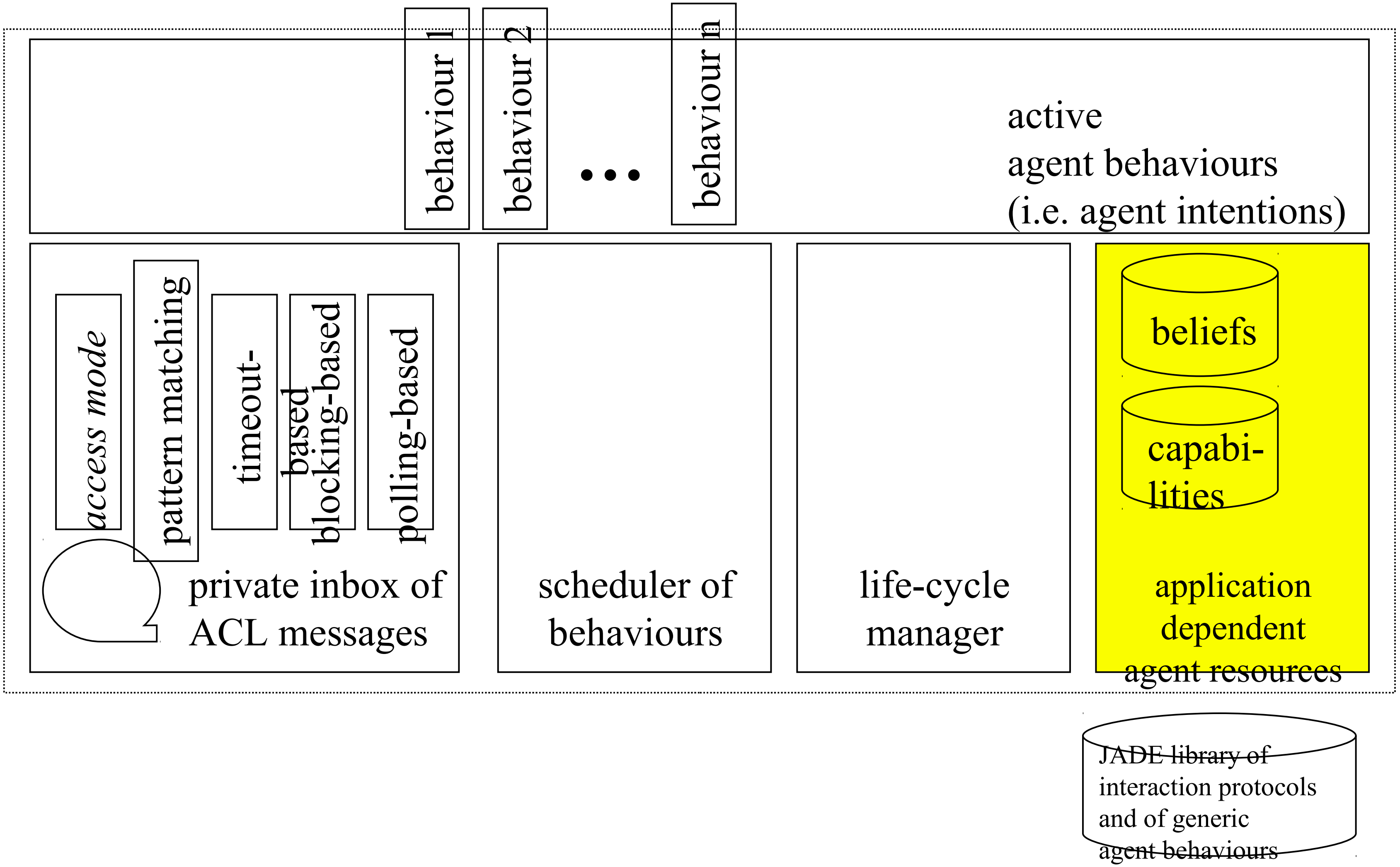
Agent Execution Model

- Agent is autonomous
  - it completely controls its thread of execution
    - private proxy of the life-cycle manager
  - it decides itself when to read received messages and which messages to read/serve
    - the transport mechanism fills a private queue but it does not call the agent code (no automatic callback)
- Agent needs concurrency
  - can engage multiple simultaneous conversations
  - can execute several concurrent tasks
    - Java multi-thread or/and
    - JADE behaviours with cooperative scheduling
      - one thread-per-agent rather than one thread-per-task/conversation

Agent Tasking Model

- JADE uses the Behaviour abstraction to model the agent tasks
- An agent can instantiate at run time new behaviours according to its needs and capabilities
- Keeps the number of threads required to run the agent platform small
- Does not prevent the programmer from using one thread-per-task implementation (no specific support is given for this case: Java multi-thread is considered sufficient)

Generic JADE agent

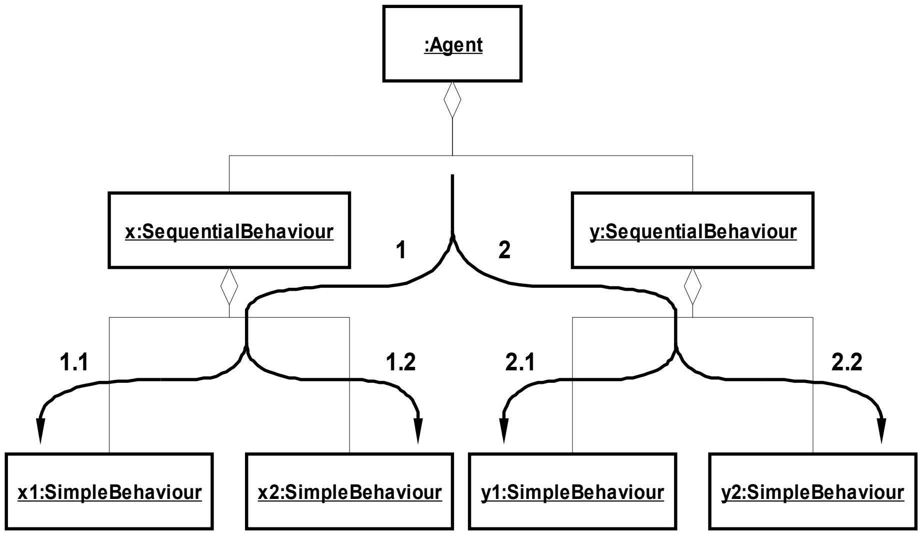


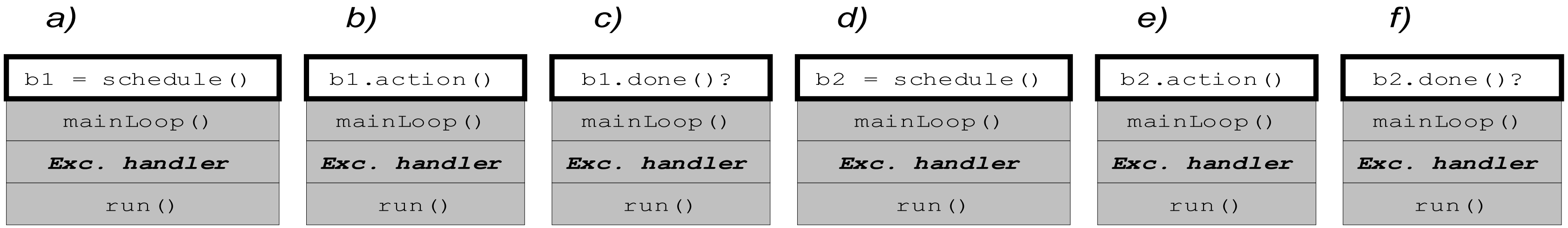
Concurrency in JADE

- Different containers on the same platform
  - 1 JVM per container
- Different agents on the same container
  - run in a pre-emptive multi-threaded environment scheduled by the JAVA Virtual Machine
- Different behaviours on the same agent
  - scheduled cooperatively
    - every behaviour must release the control to allow the other behaviours to be executed
    - no stack to be saved, more effort to the programmer
    - JADE scheduler carries out a round-robin non-preemptive policy among all behaviours in the ready queue
  - Behaviours can be composed into a tree
    - every Behaviour is a Finite State Machine
      - one state per execution time slot

Behaviour-Based Concurrency Model

- Multithreaded inter-agent scheduling.
- Behaviour abstraction
  - *Composite* for structure
  - *Chain of Responsibility* for scheduling.
  - No context saving.





a)	b)	c)	d)	e)	f)
b1 = schedule()	b1.action()	b1.done()?	b2 = schedule()	b2.action()	b2.done()?
mainLoop()	mainLoop()	mainLoop()	mainLoop()	mainLoop()	mainLoop()
Exc. handler	***Exc. handler***	***Exc. handler***	***Exc. handler***	***Exc. handler***	***Exc. handler***
run()	run()	run()	run()	run()	run()

Scalability in JADE

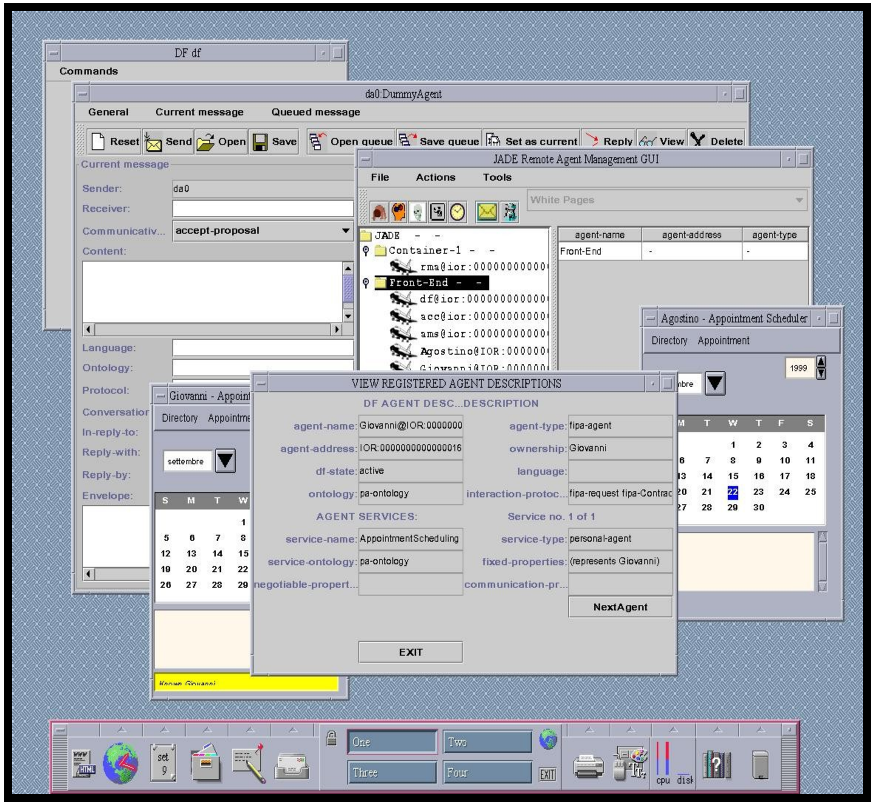
- Configuration of a platform
  - From one MAS on a single host
    - single-host platform
  - To one agent on a single host
    - agent platform on a cluster of hosts
  - Configuration can be changed at run-time
    - Hot restarting is possible thanks to the local caches
      - Agent is referred by name => no need to get new reference
- The agent container front end is the bottle-neck but it is involved only when strictly necessary

Communication Overheads

- Same container
  - No remote invocations
  - ACL message object is cloned
- Same Agent Platform, different container, cache hit
  - One RMI call, the ACL message object is serialized and unserialized by RMI run time
- Same Agent Platform, different container, cache miss
  - Two RMI calls (update the cache, send the message), the ACL message object is serialized and unserialized by RMI run time
- Different platforms
  - CORBA remote invocation through IIOP
  - Double marshalling from Java object to Java String to IIOP byte stream (sender side)
  - Double unmarshalling from IIOP byte stream to Java String to Java object (receiver platform side)

JADE Support Tools

- Administration tools.
  - RMA Management Agent.
    - White pages GUI.
    - Agent life cycle handling.
  - Directory Facilitator GUI.
    - Yellow pages handling.
- Development tools.
  - DummyAgent.
    - Endpoint Debugger.
  - Message Sniffer.
    - Man-in-the-middle.



Current message

Sender: da0

Receiver:

Communicativ...: accept-proposal

Content:

Language:

Ontology:

Protocol:

Conversation:

In-reply-to:

Reply-with:

Reply-by:

Envelope:

JADE - -
Container-1 - -
rma@ior:00000000000
Front-End - -
df@ior:000000000000
acc@ior:00000000000
ams@ior:00000000000
Agostino@IOR:000000
Giovanni@IOR:000000

agent-name	agent-address	agent-type
Front-End	-	-

VIEW REGISTERED AGENT DESCRIPTIONS

DF AGENT DESC...DESCRIPTION

agent-name: Giovanni@IOR:0000000
agent-type: fipa-agent
agent-address: IOR:000000000000016
ownership: Giovanni
df-state: active
language:
ontology: pa-ontology
interaction-protoc...: fipa-request fipa-Contrac

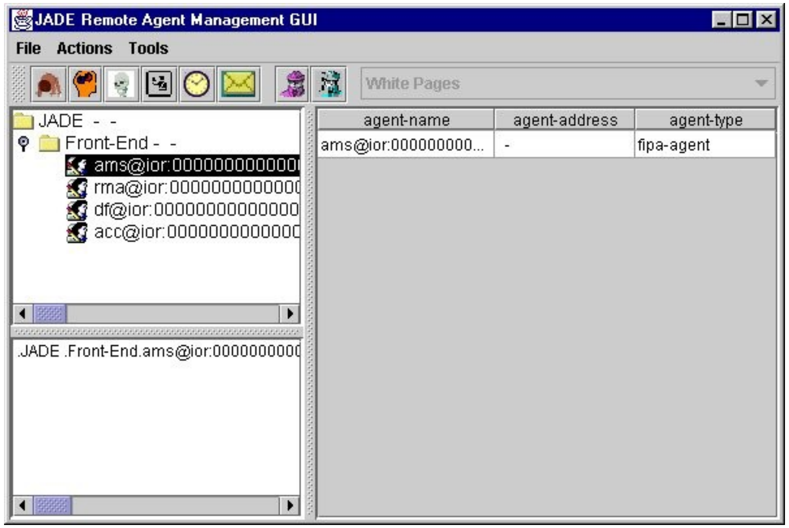
AGENT SERVICES:
Service no. 1 of 1

service-name: AppointmentScheduling
service-type: personal-agent
service-ontology: pa-ontology
fixed-properties: (represents Giovanni)
negotiable-propert...:
communication-pr...:

NextAgent

EXIT

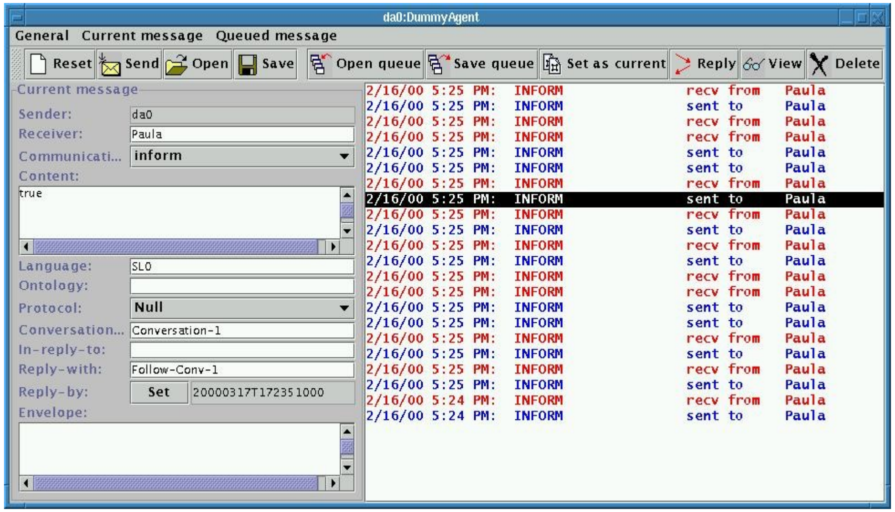
JADE Support Tools



- JADE - -
  - Front-End - -
    - ams@ior:000000000000
    - rma@ior:000000000000
    - df@ior:0000000000000
    - acc@ior:000000000000

.JADE .Front-End.ams@ior:0000000000

agent-name	agent-address	agent-type
ams@ior:000000000...	-	fipa-agent



Sender: da0

Receiver: Paula

Communicati... inform

Content:
true

Language: SL0

Ontology:

Protocol: Null

Conversation... Conversation-1

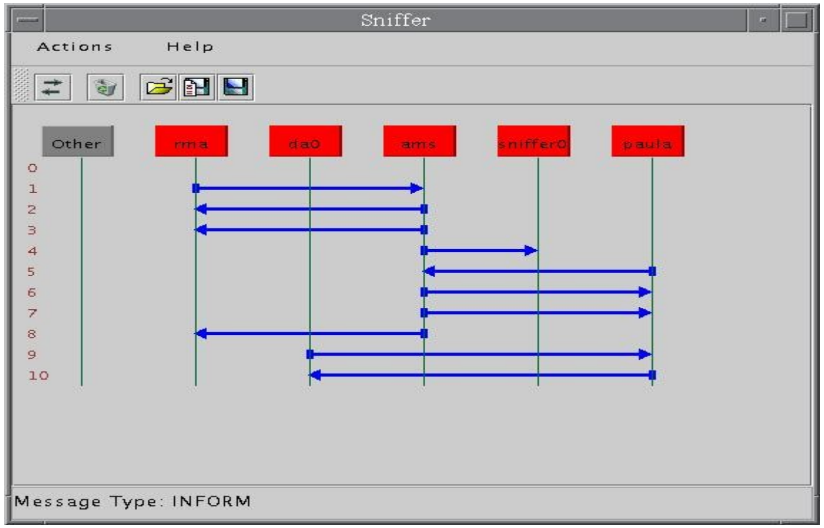
In-reply-to:

Reply-with: Follow-Conv-1

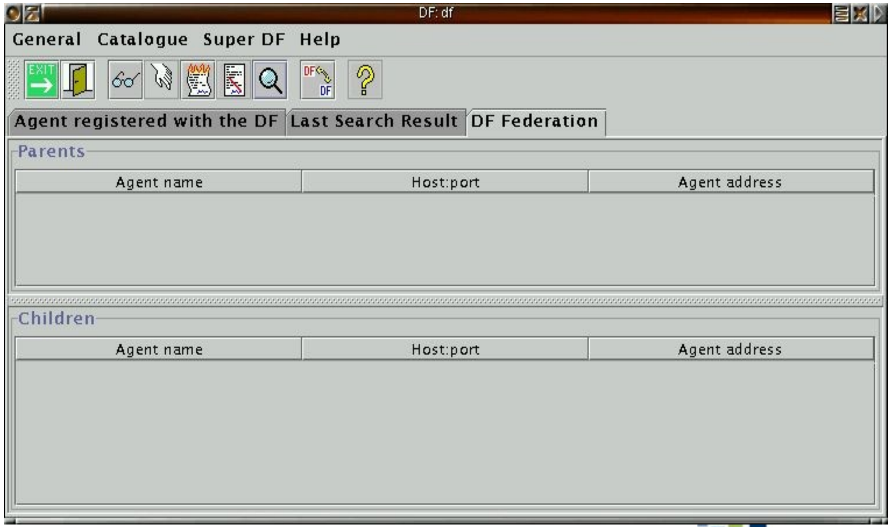
Reply-by: Set 20000317T172351000

Envelope:

```
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:25 PM:   INFORM          recv from   Paula
2/16/00 5:25 PM:   INFORM          sent to     Paula
2/16/00 5:24 PM:   INFORM          recv from   Paula
2/16/00 5:24 PM:   INFORM          sent to     Paula
```



Message Type: INFORM



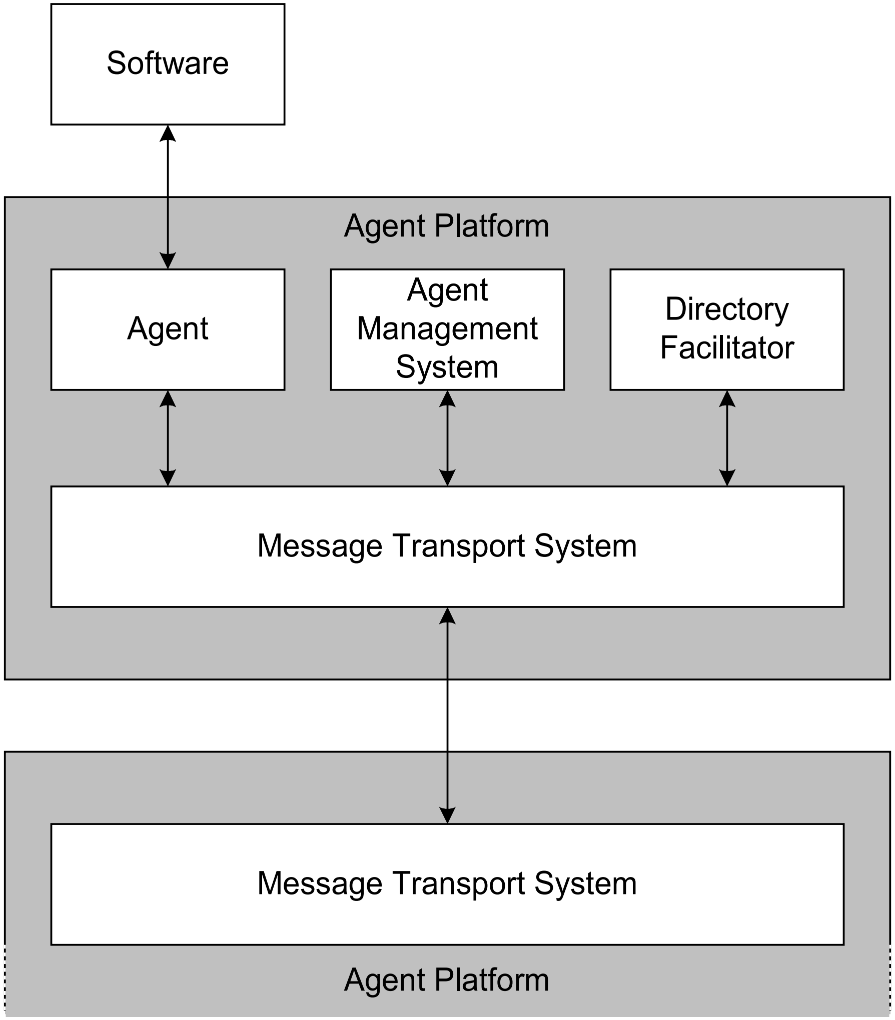
Parents

Agent name	Host:port	Agent address

Children

Agent name	Host:port	Agent address

Agent Management Model

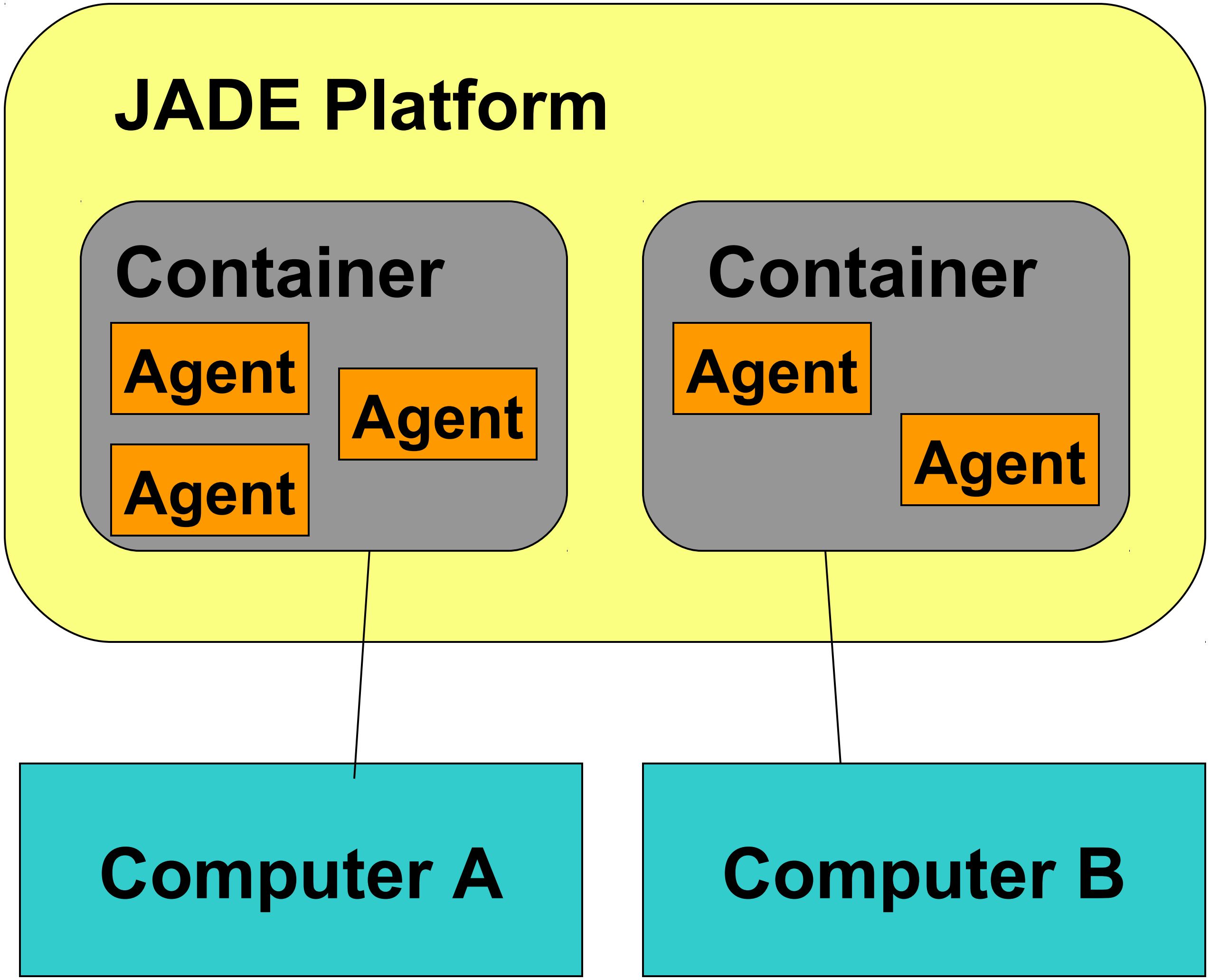


- The main difference is the ACC is not an agent.
- The ACC supports the Message Transport System.
- The Message Transport Service is the default communication method between agents on different APs.
- The ACC also supports AP routing tasks.

Jade Toolkit

- Platform Runtime
  - Agent lifecycle
  - Message transport
  - Service agents
- Agent Support
  - Internal scheduling
  - Decoding and understanding communication
- Debugging
  - Tracing Messages
  - Examining Agent State

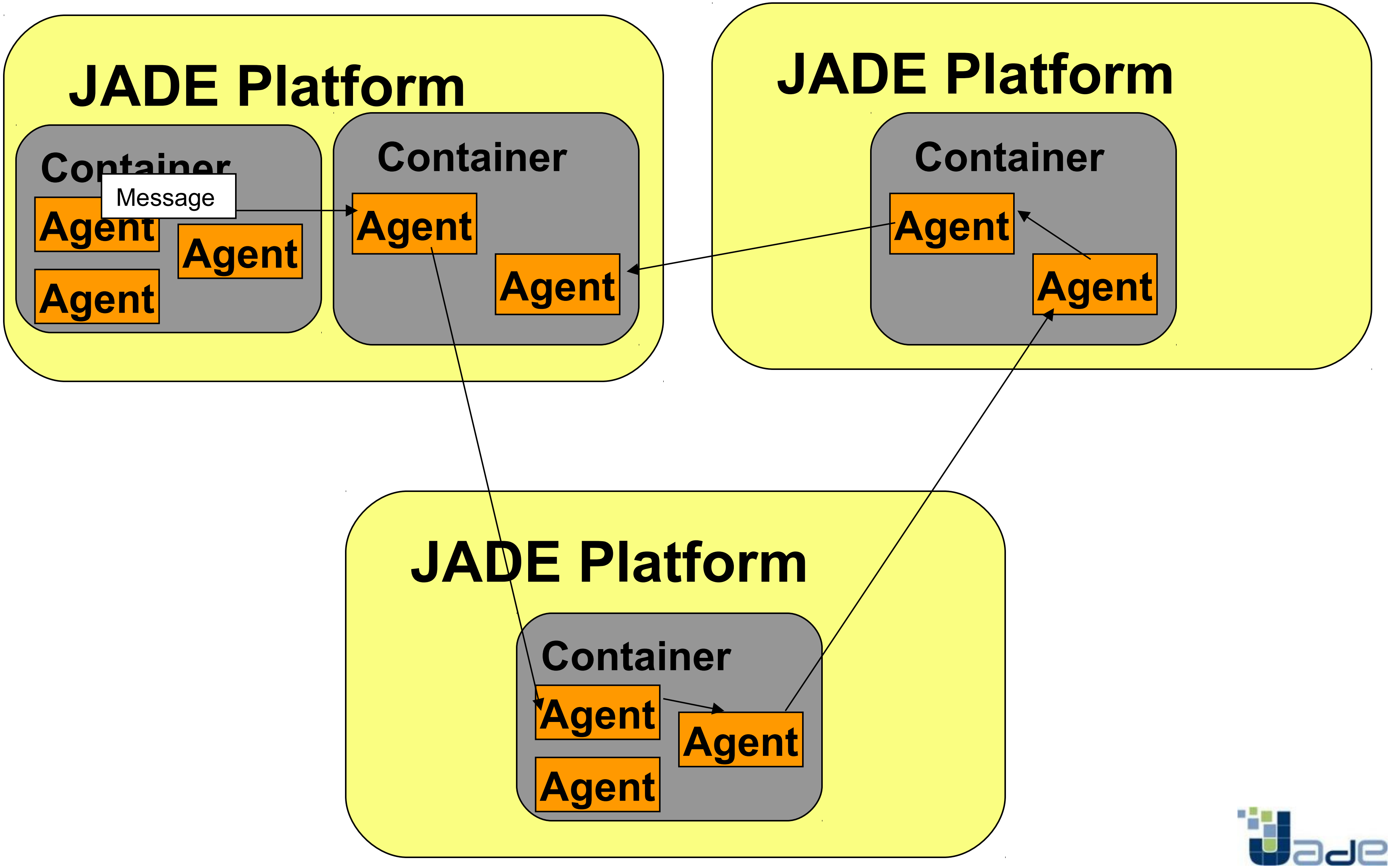
JADE Platform



JADE Architecture

- Distributed Architecture
  - Multiple Hosts
  - Multiple Processes
  - Multiple “Agent Containers”
- Agent Containers
  - One java process per container
  - Transparent to agents
  - Main container hosts platform services
  - Linked together using Java RMI

Distributed Architecture



Platform Services

- Implemented as agents
- Agent Management Service (AMS)
  - "White Pages"
  - Maintains set of agents on a platform
- Directory Facilitator (DF)
  - "Yellow Pages"
  - Provides a service directory
  - Maps service descriptions to Agent Identifiers
  - Agents can add/modify/delete entries for themselves

JADE Programming Model

- JADE based applications are made by one or more ***Agents***
  - A JADE agent is mapped onto an user defined Java class, that must subclass `Agent` class in `jade.core` package.
  - Agent activities are mapped onto user defined subclasses of `Behaviour` class in `jade.core.behaviours` package.

Internal Agent Architecture

- Agent is a single-threaded Java program
- Has tasked-based programming model
  - Set of Behaviours
  - Scheduled by the Agent
  - Tasks operate in parallel
  - Allows explicit coordination between tasks
  - Single-threaded
- Send/Receive messages through API
  - Decoding/Encoding message content

Agent Class

- The agent name is an Agent Identifier, represented by AID class
- The name attribute is a globally unique identifier.
- The full name of an agent is no longer composed of the concatenation of its local name and the platform IIOP address
  - FIPA does not allow to distinguish between the local name and its home AP address.
- The method, getAID(), returns the agent AID.

JADE Agent

- Your agent should inherit from Agent

- See JADE API docs for inherited methods

- Everything begins with setup:

  protected void setup () { … }

  - This is where you initialize and addBehaviours to handle the processing

- Other useful methods:
  - protected void takeDown() { … } // cleanup
  - public void addBehaviour(Behaviour b)
  - public final ACLMessage blockingReceive(MessageTemplate pattern, long millis)
  - public final ACLMessage receive(MessageTemplate pattern)
  - public final void send(ACLMessage msg)

Behaviours

- Behaviour not Behavior (British spelling)
- Each agent has a set of active behaviours
- Each behaviour should achieve a single task or sub-task
  - “Send this message”
  - “Buy X”
- A behaviour
  - Runs until it is finished
  - Should behave fairly—yield control, not block
  - No infinite loops
- All active behaviours get executed with equal frequency
- Can create composite behaviours

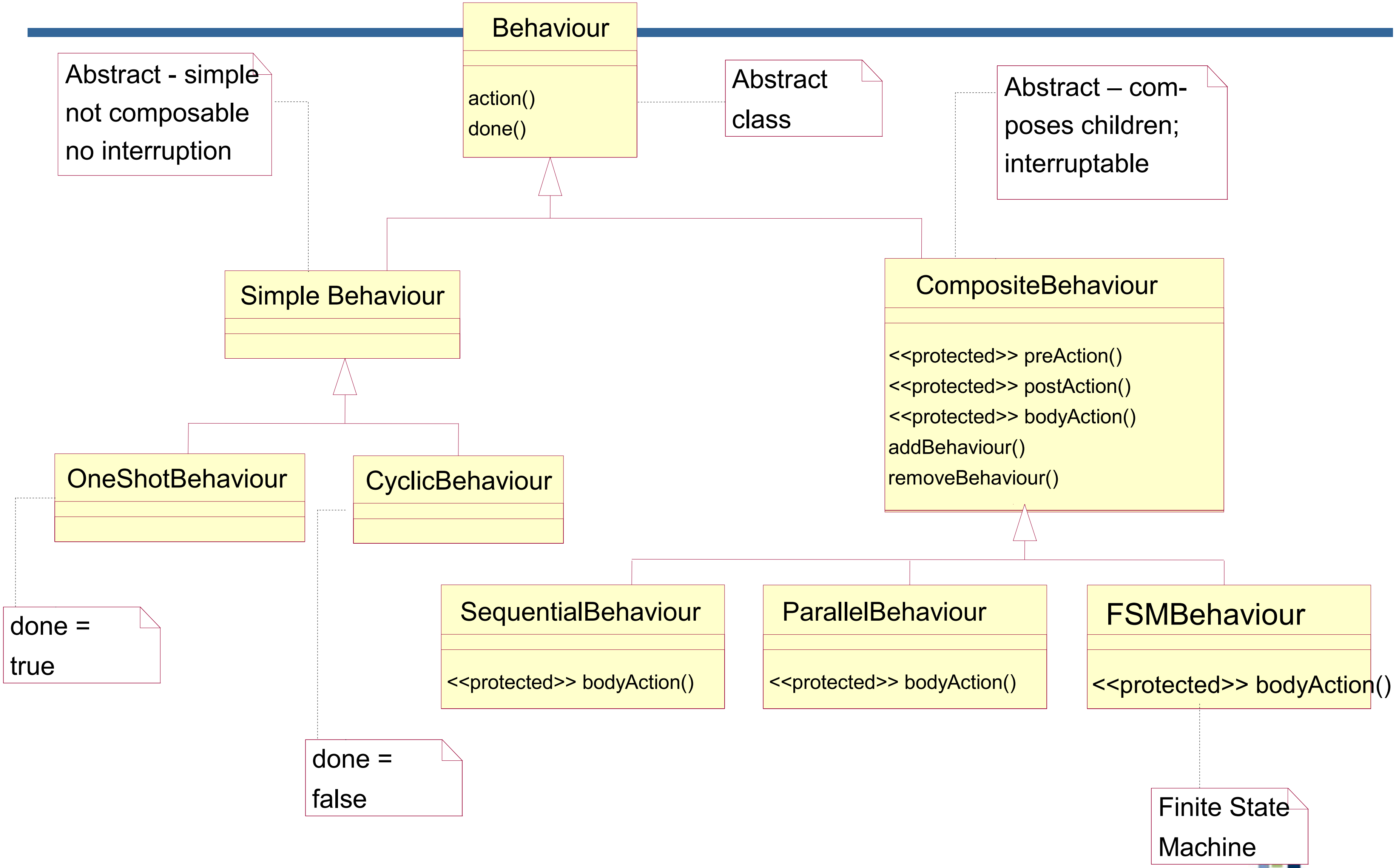
Basic Behaviour Model

- Round-robin scheduling of all behaviours in the run queue
  - When message arrives, all blocked behaviours are added to run queue
  - Next behaviour selected from all “active behaviours”
  - All active behaviours get executed equally often
- Primary methods:
  - public abstract void action()
    - Each time the behaviour has a turn, action is called
  - public abstract boolean done()
    - When action is finished this is called to determine if we put back into run queue or leave in list of finished behaviours
    - Returns false while working or true when finished
  - public void block()
    - Tells to move to blocked queue when the action exits
  - public void block(long millis)
    - Adds ability to wait until woken up or timeout

Predefined Behaviours

- You define action yourself
- SimpleBehaviour
  - You define done() yourself
- CyclicBehaviour
  - You may NOT define done()
  - ‘done’ is permanently set to return false
- OneShotBehaviour
  - You may NOT define done()
  - ‘done’ is permanently set to return true

Behaviours



Behaviour

action()

done()

Abstract class

Abstract - simple not composable no interruption

Abstract – composes children; interruptable

Simple Behaviour

CompositeBehaviour

<<protected>> preAction()

<<protected>> postAction()

<<protected>> bodyAction()

addBehaviour()

removeBehaviour()

OneShotBehaviour

CyclicBehaviour

done = true

done = false

SequentialBehaviour

<<protected>> bodyAction()

ParallelBehaviour

<<protected>> bodyAction()

FSMBehaviour

<<protected>> bodyAction()

Finite State Machine

Assigning Behaviour

Agent class, which must be extended by agent programmers exposes two methods:

addBehaviour(Behaviour) and removeBehaviour(Behaviour), which allow to manage the ready tasks queue of a specific agent.

Behaviours and sub-behaviours can be added whenever is needed, and not only within Agent.setup() method. Adding abehaviour should be seen as a way to spawn a new (cooperative) execution thread within the agent

Example of a Behaviour

```
public class my3StepBehaviour extends OneShotBehaviour{
  private int state = 1;
  private boolean finished = false;

  public void action() {
    switch (state) {
      case 1: { op1(); state++; return; }
      case 2: { op2(); state++; return; }
      case 3: { op3(); state=1; finished = true; return; }
     }
  }

  public boolean done() {
    return finished;
  }
}
```

Composite Behaviours

- Complex Behaviour has child behaviours which may be simple or complex
- When Complex behaviour is active
  - Behaviour decides which children are active.
  - Agent Scheduler runs active children
  - Simple behaviours actually implement tasks
- Complex behaviours decide which tasks are necessary

Adding Behaviour to an Agent

```
addBehaviour(new TickerBehaviour(this, 1000) {
      protected void onTick() {
        System.out.println("Agent "+myAgent.getLocalName()+":
   tick="+getTickCount());
      }
   });
```

Adding WakerBehaviour

```
addBehaviour(new WakerBehaviour(this, 10000) {
      protected void handleElapsedTimeout() {
        System.out.println("Agent "+myAgent.getLocalName()+":
   It's wakeup-time. Bye...");
        myAgent.doDelete();
      }
    });
```

Adding CycleBehaviour

```
addBehaviour( new CyclicBehaviour( this ) {
         public void action() {
             ACLMessage msg = receive(
  MessageTemplate.MatchPerformative(
              ACLMessage.INFORM ) );

                    if (msg != null) {
                       if
  (HostAgent.GOODBYE.equals( msg.getContent() )) {
                         .......
```

Complex Behaviours: Sequential

- Child behaviours are activated in order added.
- First behaviour is run until completed, then next and so on.
- Complex behaviour finishes when last child does.

Complex Behaviours: Parallel

- All child behaviours are activated simultaneously
- Behaviour can exit:
  - When first child completes (conditionally dependent)
  - When all children complete

class ParallelBehaviour

This class is a CompositeBehaviour that executes its sub-behaviours concurrently and terminates when a particular condition on its sub-behaviours is met. Proper constants to be indicated in the constructor of this class are provided to create a ParallelBehaviour that ends when all its sub-behaviours are done,

Complex Behaviours: Finite State Machine

- Deterministic FSM.
- Sub-behaviours are states
- Current state is active behaviour
- Transitions occur depending on outcome of last behaviour

Complex Behaviours: Protocols

- Off-the shelf implementation of interaction protocols
- Does sending/receiving messages for you
- Checks protocol compliance
- Each place where agent can act in protocol is a sub-behaviour (or a callback)
- Just need to plug in logic to implement agent decisions
- Built internally using FSM behaviours

Example Agent

- Example1
  - SimpleReceiverAgent
    - Same as sender but uses SimpleReceiverBehaviour
  - SimpleReceiverBehaviour
    - CyclicBehaviour
    - Checks to see if message has arrived and prints if it does
    - Does a block(500) to stop busy waiting
    - Could use blockingReceive or block()

Execute the behaviour in a dedicated Thread

```
public class ThreadedAgent extends Agent {
private ThreadedBehaviourFactory tbf = new
   ThreadedBehaviourFactory();
protected void setup() {
// Create a normal JADE behaviour
Behaviour b = new OneShotBehaviour(this) {
public void action() {
// Perform some blocking operation that can take a long time
}
};
myAgent.addBehaviour(tbf.wrap(b));
}
}
```

Agent Communications

Agent Communication

- All communication is message-based unlike Remote Procedure Calls (RPC)
- No explicit link between initiation and response
- Platform Routes messages between agents
- Agent constructs and decodes messages internally

Constructing a Message

- Create Message Object (ACLMessage)
- Set performative
- Set sender and recipients
- Set message attributes
- Fill content (from java objects)

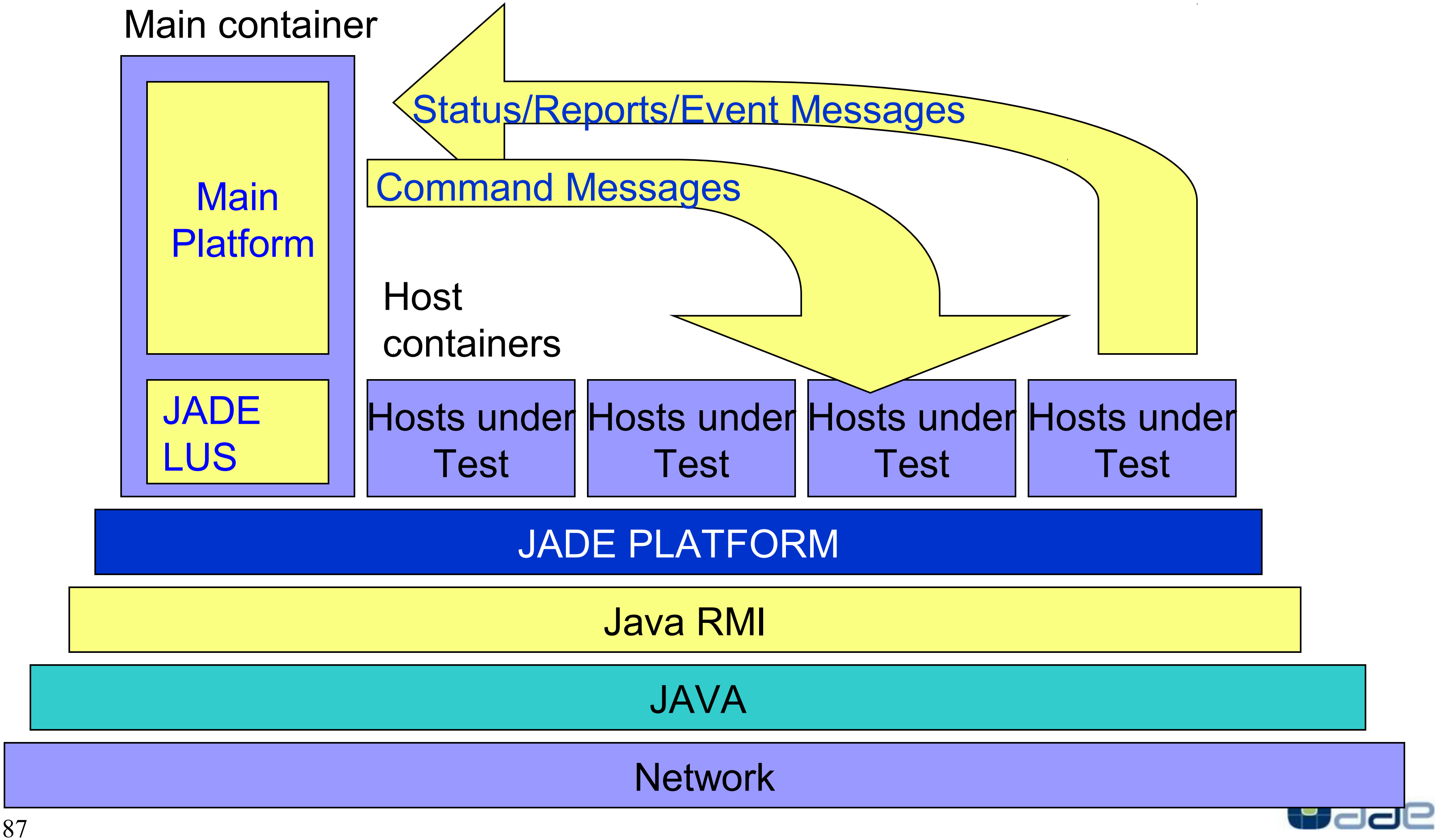
Message Structure

- Envelope
  - Constructed/Interpreted by platform
  - Routing: Indicates sender and recipients of message and where the message has been
- ACL Message
  - Performative (REQUEST/INFORM)
  - Set of standard attributes
    - Language of message body
    - Conversation ID
    - Ontology used in message
  - Body
    - Interpreted by agents
    - Conforms to some content language and Ontology

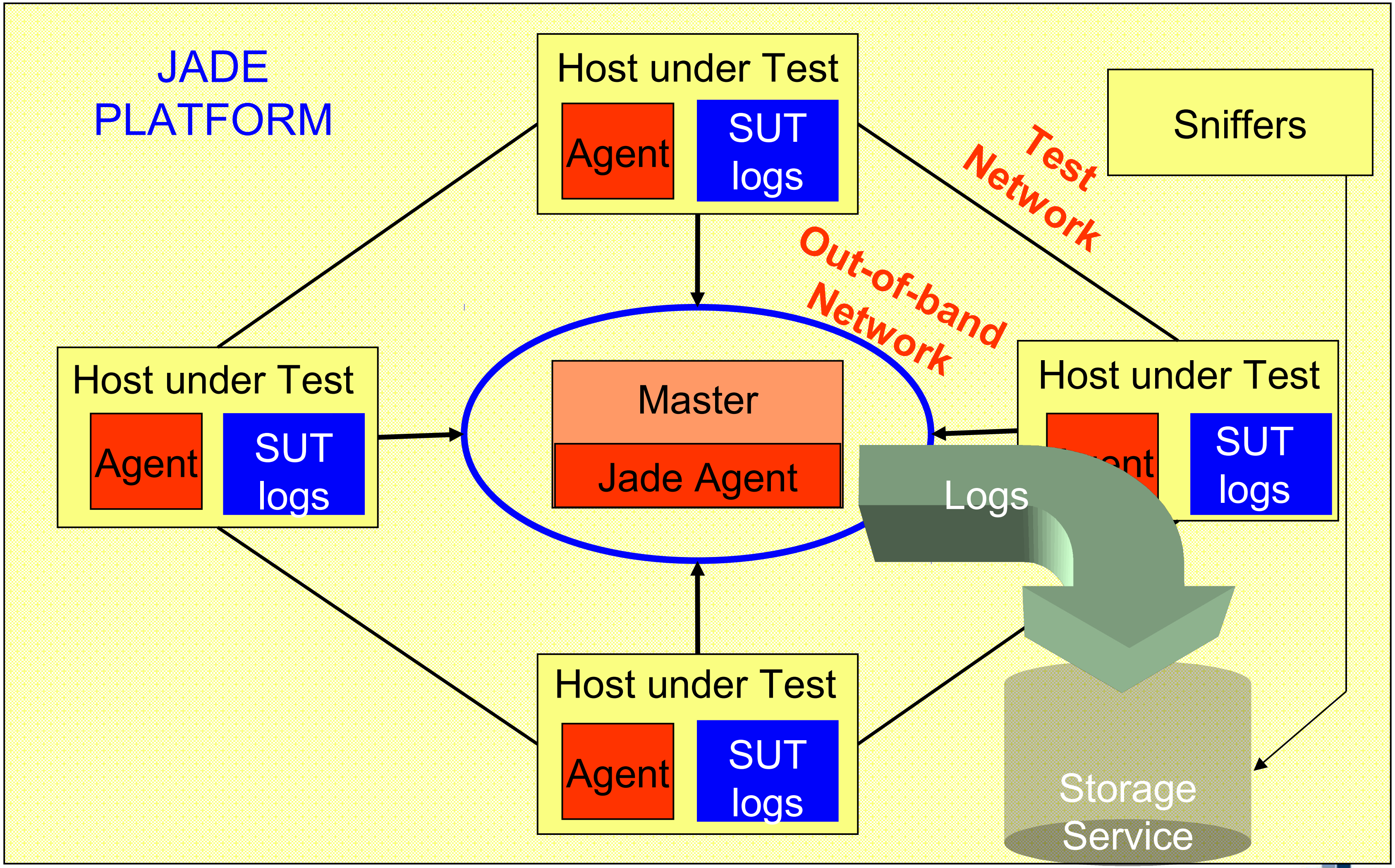
Message Content

- Content of message complies with Content Language (CL) and Ontology
- Semantic Language (SL)
  - SL0: Objects, Actions, Results
  - SL1: Basic Logical Operators, Predicates AND/OR/NOT
  - SL2: Higher-level modalities: Belief, Intention, Goal
- Objects in language are described by Ontology

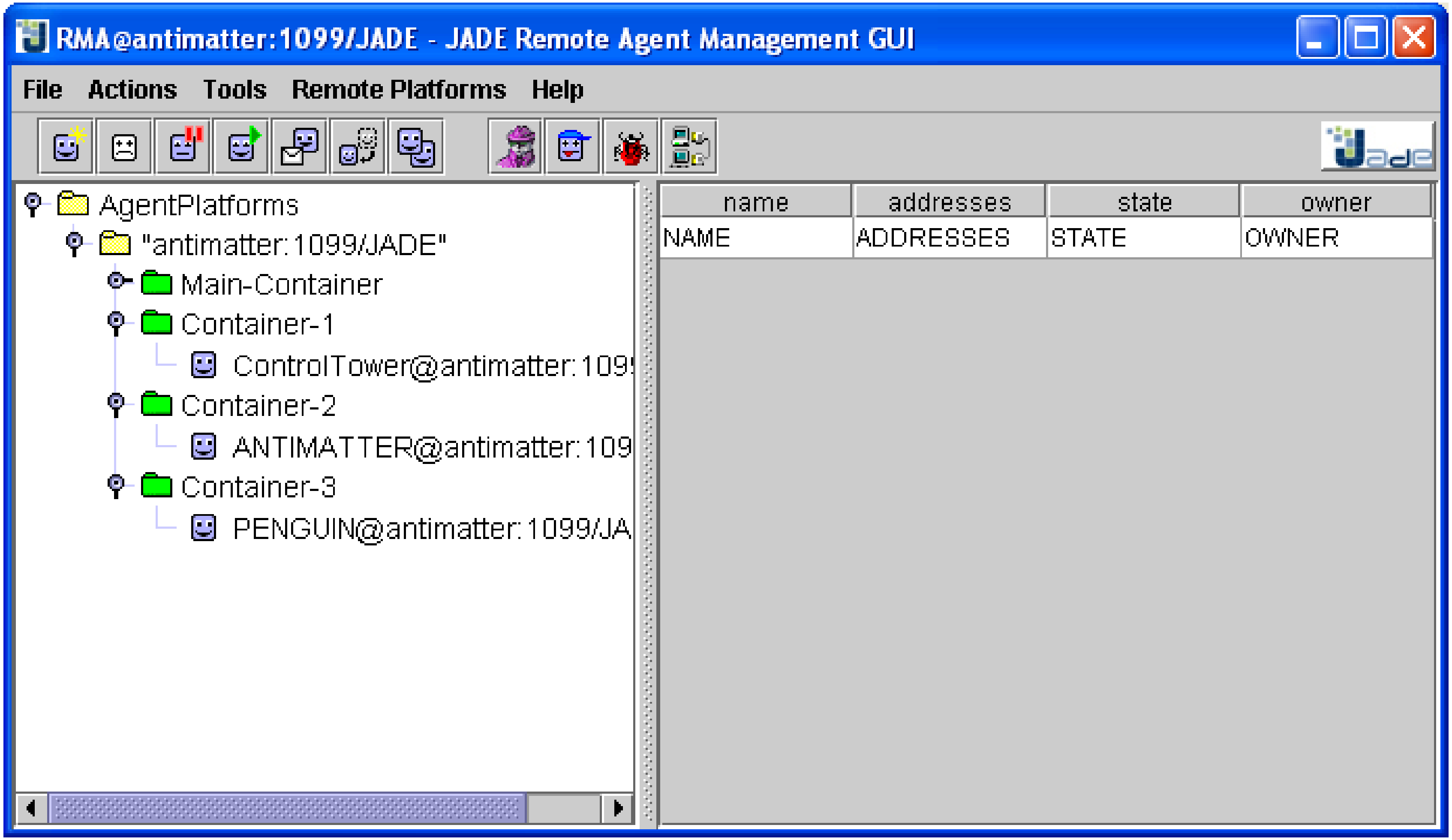
Jade Messaging Architecture



Instrumentation : Underlying Technology



JADE Platform Control/Status display

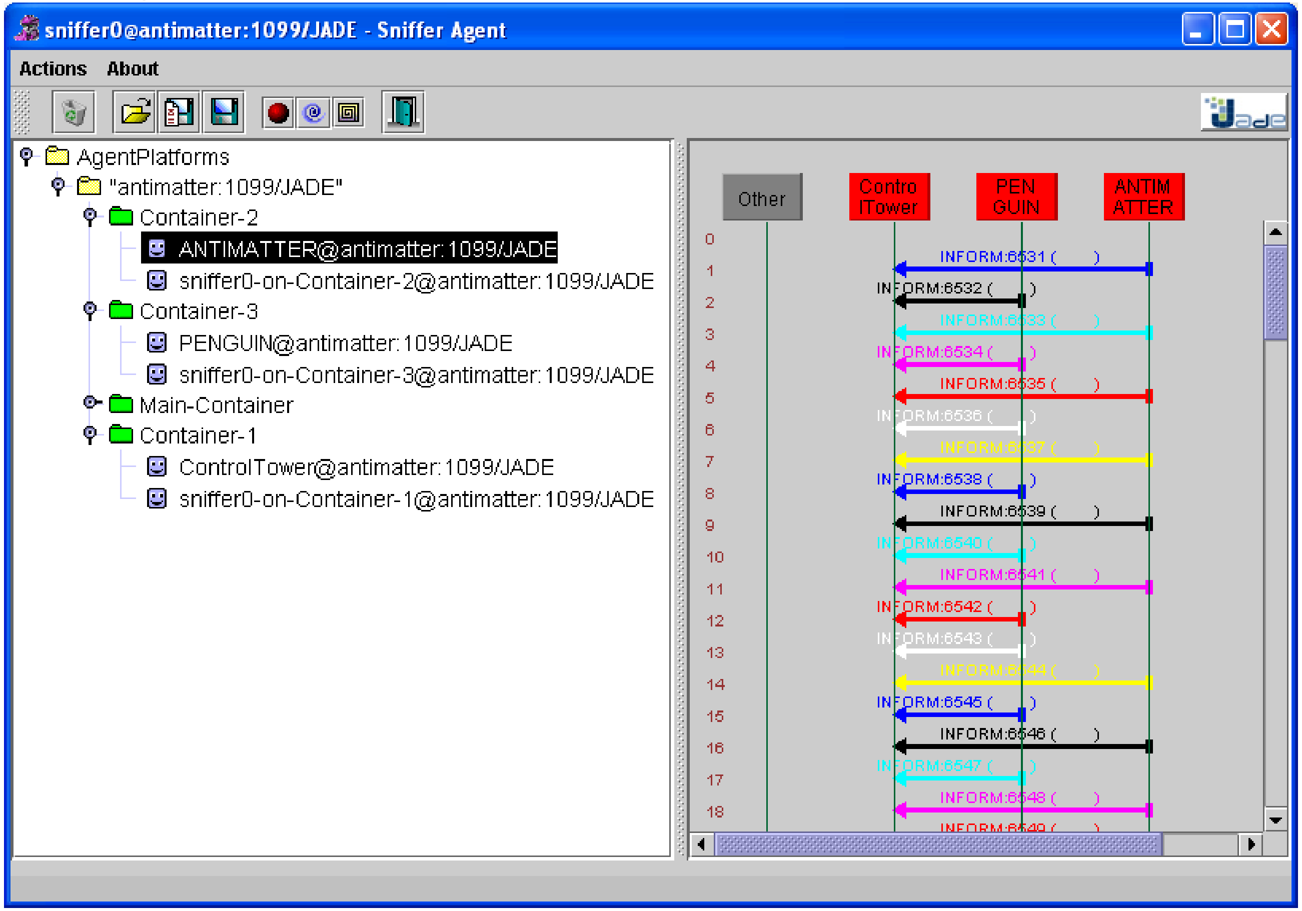


- AgentPlatforms
  - "antimatter:1099/JADE"
    - Main-Container
    - Container-1
      - ControlTower@antimatter:109
    - Container-2
      - ANTIMATTER@antimatter:109
    - Container-3
      - PENGUIN@antimatter:1099/JA

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

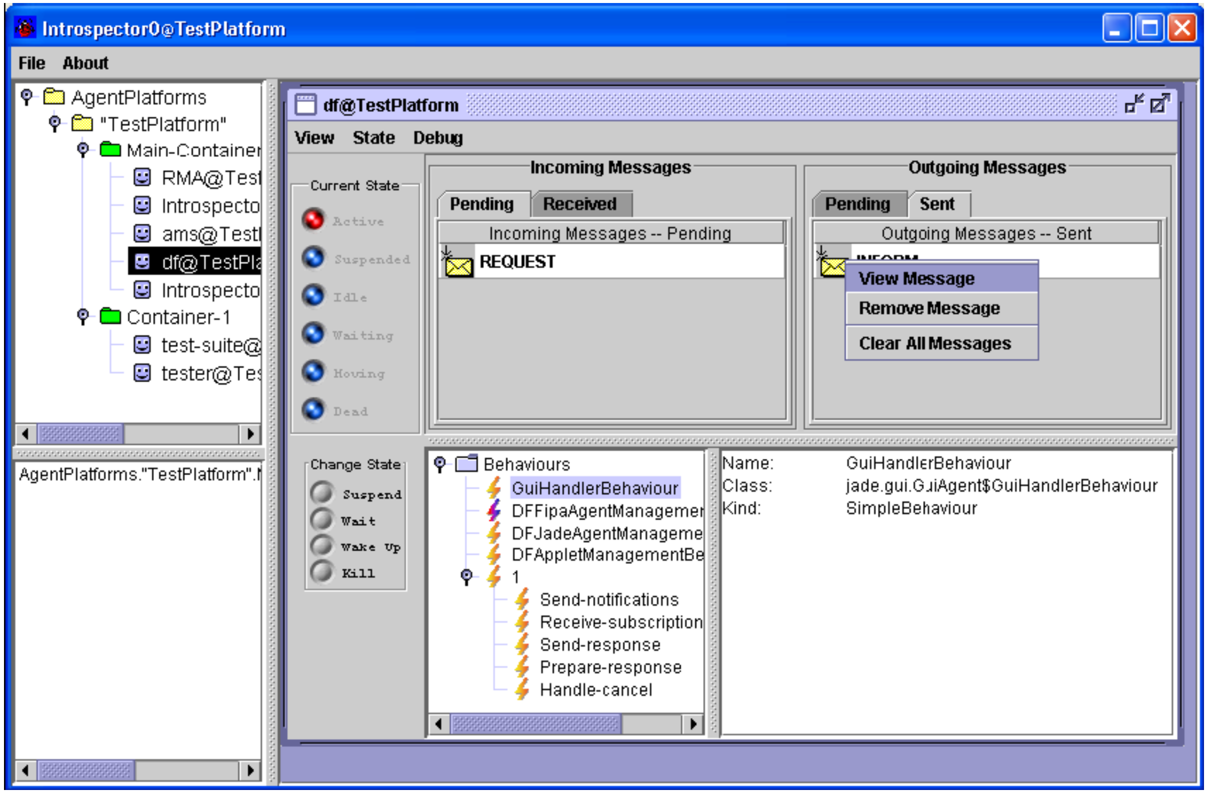
JADE Sniffer Agent

Sniffer Agent is basically a FIPA-compliant Agent with sniffing features.



Introspector Agent

This tool allows to monitor and control the life-cycle of a running agent and its exchanged messages



Name: GuiHandlerBehaviour
Class: jade.gui.GuiAgent$GuiHandlerBehaviour
Kind: SimpleBehaviour

Ontology

- Ontology provides description of concepts in the world and relationships between them

Ontology: Terms & Relationships

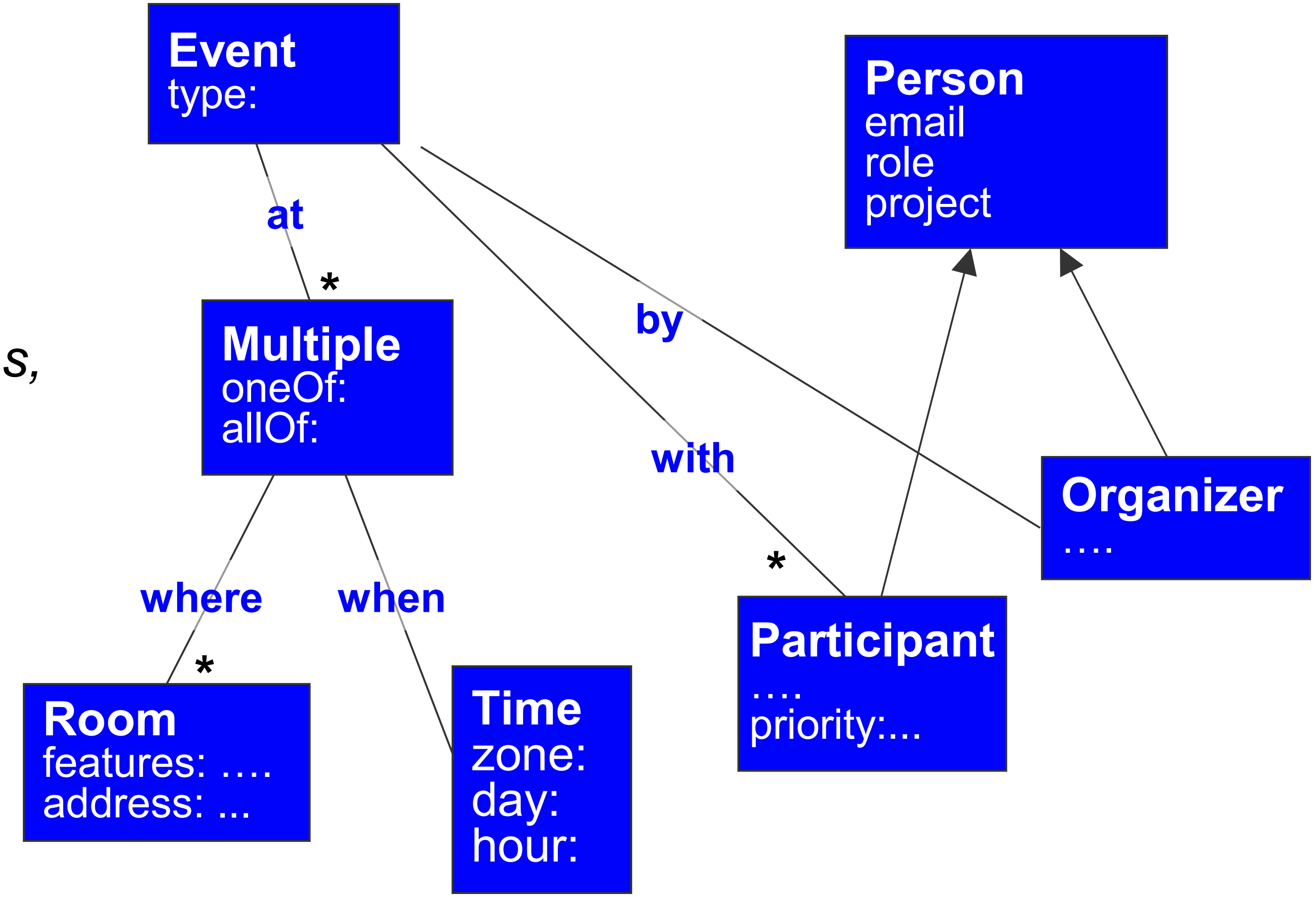
Standards

Shopping, Events, Travel, People, Organizations, Roles,

Context, …

Tools & Models

UML, XML, AUML, DAML, OIL, RDF, Protégé, ...



Ontology Description

Common vocabulary of agreed upon definitions and relationships between those definitions, to describe a particular subject domain.

E.g.:

Agent management

ontology

UMTS wireless technology ontology

Cinema ontology

Weather ontology

IEEE Standard Upper Ontology

W3C.org

- An ontology defines the terms used to describe and represent an area of knowledge. Ontologies are used by people, databases, and applications that need to share domain information (a domain is just a specific subject area or area of knowledge, like medicine, tool manufacturing, real estate, automobile repair, financial management, etc.). Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them (note that here and throughout this document, definition is not used in the technical sense understood by logicians). They encode knowledge in a domain and also knowledge that spans domains. In this way, they make that knowledge reusable.

W3C.org Concepts

- Classes (general things) in the many domains of interest
- The relationships that can exist among things
- The properties (or attributes) those things may have

W3C.org

Ontologies are usually expressed in a logic-based language, so that detailed, accurate, consistent, sound, and meaningful distinctions can be made among the classes, properties, and relations. Some ontology tools can perform automated reasoning using the ontologies, and thus provide advanced services to intelligent applications such as: conceptual/semantic search and retrieval, software agents, decision support, speech and natural language understanding, knowledge management, intelligent databases, and electronic commerce.

The End

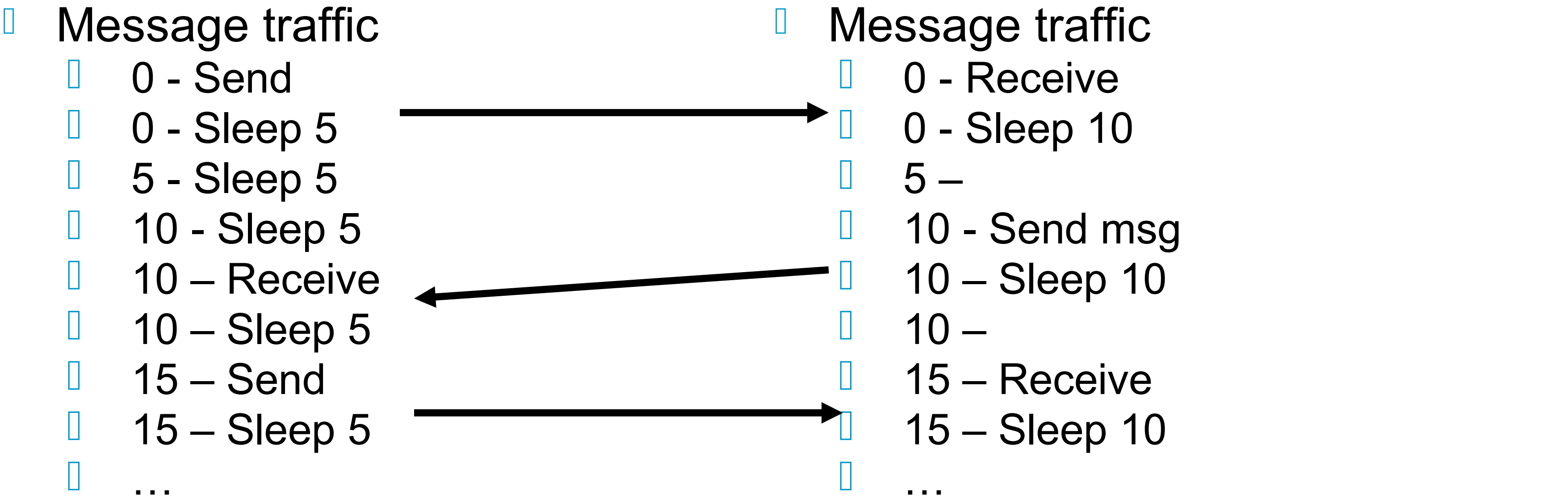
More on Behaviours

- Suppose that we want two agents that talk back and forth to each other
  - Have one be the initiator and start the conversation
    - Could imagine other ways of doing this – first one to start becomes the initiator
  - To simplify things, we have one agent and use it twice
    - So, we do a condition to see if it is the initiator or not
    - The code is nearly identical and thus creating another whole agent program seems like a waste
- Example2 provides the code
  - See next slide for an illustration of how it works

Exchange Messages

- Me (initiator)
  - Send message
  - Wake up:
    - If message, remember that you have seen it
    - If no message and seen a message, send a reply
  - Go back to sleep for 5 seconds

- You
  - Not initiator, so don't send
  - Wake up:
    - If message, remember that you have seen it
    - If no message and seen a message, send a reply
  - Go back to sleep for 10 seconds



- Message traffic
  - 0 - Send
  - 0 - Sleep 5
  - 5 - Sleep 5
  - 10 - Sleep 5
  - 10 – Receive
  - 10 – Sleep 5
  - 15 – Send
  - 15 – Sleep 5
  - …

- Message traffic
  - 0 - Receive
  - 0 - Sleep 10
  - 5 –
  - 10 - Send msg
  - 10 – Sleep 10
  - 10 –
  - 15 – Receive
  - 15 – Sleep 10
  - …

A Different Approach

- Question – what if you wanted to use two behaviours?
  - One responsible for sending answers
  - One responsible for receiving the message
- Is this the same exact model as example2?

Wakes on message arrival or time out finished.

- Receiver

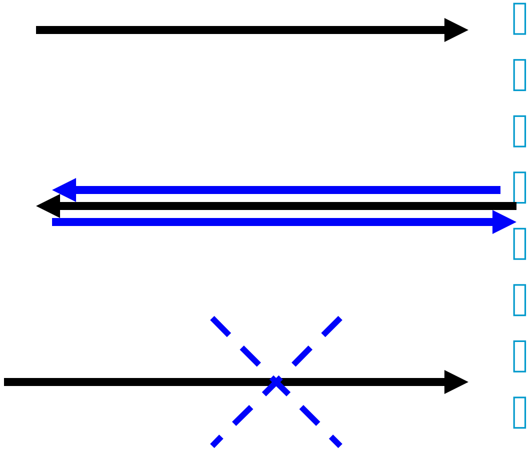
```
ACLMessage msg = receive();
if (msg != null) {
    // process message
    receivedMessage = true;
}
block();
```

Only wakes up when a message arrives

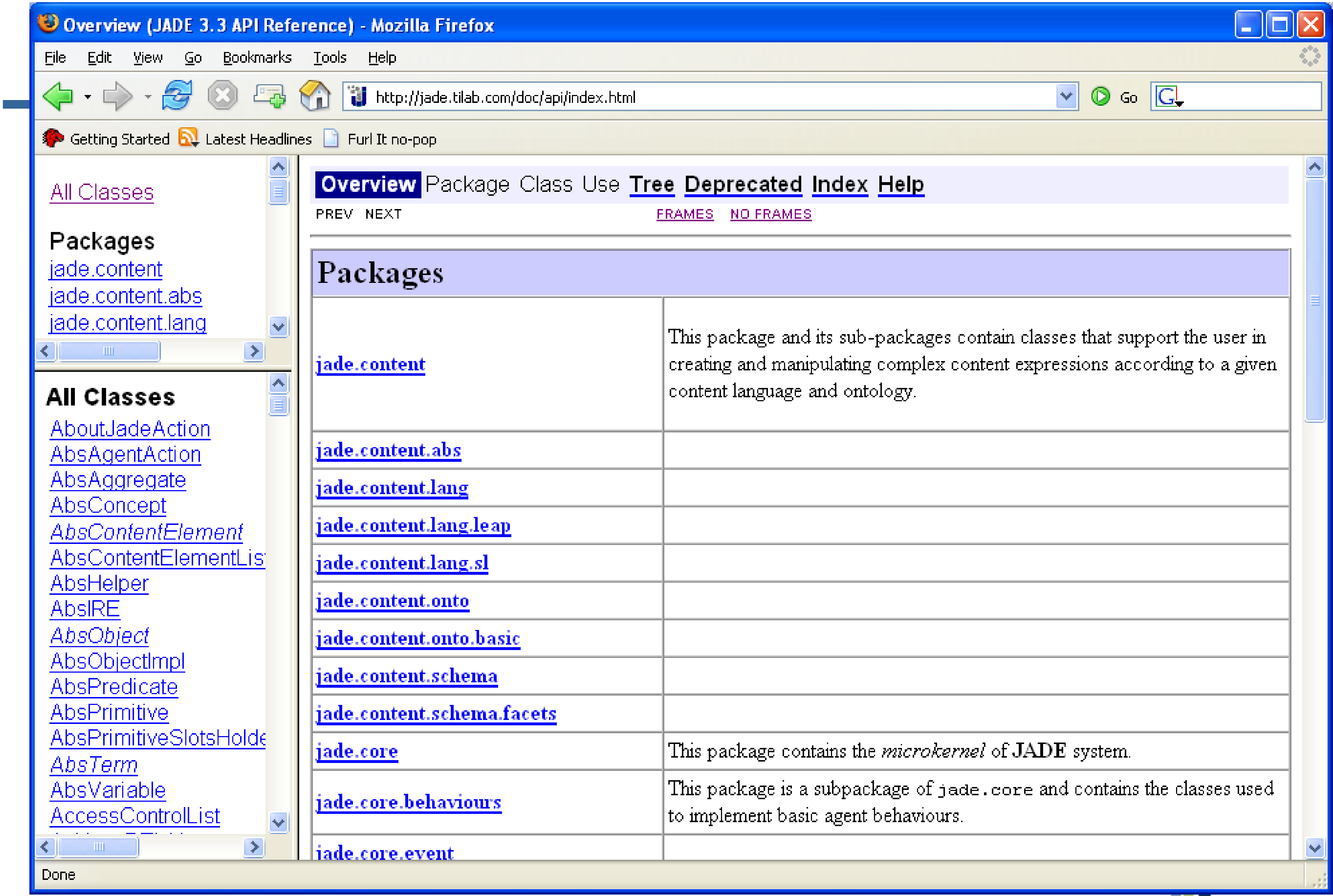
```
// Did I wake up too early?
<<do some computation on time>>
if (!tooEarly) {
    if (receivedMessage) {
        sendMessage();
    }
    block(fulltime);
} else {
    <<compute how much time left to wait>>
    block(timeLeft);
}
```

Exchange Messages - 2

- Me (Initiator)
  - Send message
  - Wake up:
    - If message, remember that you have seen it
    - If no message and seen a message, send a reply
  - Go back to sleep for 5 seconds
- Message traffic
  - 0 - Send
  - 0 - Sleep 5
  - 5 - Sleep 5
  - 10 - Sleep 5
  - 10 – Receive
  - 10 – Sleep 5
  - 15 – Send
  - 15 – Sleep 5
  - …



- You
  - Not initiator, so don’t send
  - Wake up:
    - If message, remember that you have seen it
    - If no message and seen a message, send a reply
  - Go back to sleep for 10 seconds
- Message traffic
  - 0 - Receive
  - 0 - Sleep 10
  - 5 –
  - 10 - Send msg
  - 10 – Sleep 10
  - 10 –
  - 15 – Receive
  - 15 – Sleep 10
  - …


Overview (JADE 3.3 API Reference) - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://jade.tilab.com/doc/api/index.html
Go
Getting Started
Latest Headlines
Furl It no-pop
Done

All Classes

Packages
jade.content
jade.content.abs
jade.content.lang

All Classes
AboutJadeAction
AbsAgentAction
AbsAggregate
AbsConcept
AbsContentElement
AbsContentElementLis
AbsHelper
AbsIRE
AbsObject
AbsObjectImpl
AbsPredicate
AbsPrimitive
AbsPrimitiveSlotsHolde
AbsTerm
AbsVariable
AccessControlList

Overview Package Class Use **Tree** **Deprecated** **Index** **Help**

PREV NEXT FRAMES NO FRAMES

Packages

jade.content	This package and its sub-packages contain classes that support the user in creating and manipulating complex content expressions according to a given content language and ontology.
jade.content.abs	
jade.content.lang	
jade.content.lang.leap	
jade.content.lang.sl	
jade.content.onto	
jade.content.onto.basic	
jade.content.schema	
jade.content.schema.facets	
jade.core	This package contains the *microkernel* of **JADE** system.
jade.core.behaviours	This package is a subpackage of `jade.core` and contains the classes used to implement basic agent behaviours.
jade.core.event	

BankServerAgent

```
ACLMessage msg = receive();
 if (msg == null) { block(); return; }
try { Object content = msg.getContentObject(); switch (msg.getPerformative())
 { case (ACLMessage.REQUEST):
   if (action instanceof CreateAccount) addBehaviour(new
   HandleCreateAccount(myAgent, msg)); else if (content instanceof
   MakeOperation) addBehaviour(new HandleOperation(myAgent, msg));
 ... }
```

JADE

- Java Agent Development Environment
- Standards Based - FIPA (Foundation for Intelligent Physical Agents)
  - http://www.fipa.org/
  - Non-profit association
  - Purpose to promote the success of emerging agent-based applications, services, and equipments
  - FIPA has over 65 member companies

FIPA Agent Platform

- Must Provide
  - Life Cycle Management
  - White Page Service
  - Yellow Page Service
  - Message Transport Service
- Optionally Provides
  - Ontology Service
  - Human Agent Interaction
  - Agent Software Integration