Introduction to Multi-Agent Programming

5. Agent Communication

Speech Acts, KIF, KQML,
FIPA, JADE, IPC

Alexander Kleiner, Bernhard Nebel

Contents

- Introduction
- Speech Acts
- Agent Communication Languages
  - KQML, KIF, FIPA, and Jade
- IPC (Inter Process Communication)
  - Case Study: Rescue Freiburg communication
- Summary

Introduction

- Communication in concurrent systems:
  - Synchronization of multiple processes
    - E.g., solving the "lost update scenario":
      - Two processes p_1 and p_2 access the shared variable *v*
      - During modifying of v by p_1, p_2 reads v and writes back the old value
      - Update from p_1 is lost
- Communication in OOP
  - Method invocation between different modules
    - E.g., object o2 invokes method m1 on object o1 by executing the code o1. m1(arg), where "arg" is the argument to communicate
    - Which objects makes the decision about the execution of m1?
- Communication in MAS?
  - Autonomous agents have control over both state and behavior
  - Methods are executed according to the agent's self-interest
  - However, agents can perform communicative actions, i.e. attempt to influence other agents
  - Agent communication implies interaction, i.e. agents perform communication acts

Speech Acts I

- Most treatment of communication in MAS is inspired from speech act theory
- The theory of speech acts is generally recognized to have begun with the work of the philosopher John Austin: "How to Do Things with Words" (Austin, 1962)
- *Speech act theory* studies the pragmatic use of language
  - an attempt to account for how language is used by people every day to achieve their goals and intentions
- *Speech act theory* treats communication as action
  - speech actions are performed by agents just like other actions, in the furtherance of their intentions

Speech Acts II

- Austin noticed that some utterances are rather like 'physical actions' that appear to *change the state of the world*
- For example:
  - declaring war
  - 'I now pronounce you man and wife'
- Austin identified a number of performative verbs, which correspond to various different types of speech acts
  - Examples of performative verbs are *request*, *inform*, and *promise*

Speech Acts III

- Searle (1969) extended Austin's work and identified the following five key classes of possible types of speech acts:
  - *Representatives:*
    commits the speaker to the truth of an expression, e.g., 'It is raining' (*informing*)
  - *Directives:*
    attempts to get the hearer to do something e.g., 'please make the tea' (*requesting*)
  - *Commissives:*
    which commits the speaker to do something, e.g., 'I promise to… ' (*promising*)
  - *Expressives:*
    whereby a speaker expresses a mental state, e.g., 'thank you!' (*thanking*)
  - *Declarations:*
    effect change of state, such as "declaring war" (*declaring*)
- Cohen and Perrault (1979) started to modeling speech acts in a planning system (STRIPS formalism)

Agent Communication Languages I
KQML and KIF

- *Agent communication languages* (ACLs) are standard formats for the exchange of messages
- KSE (Knowledge Sharing Effort) in early 1990s designed two ACLs with different purpose
  - The Knowledge Query and Manipulation Language (KQML), which is an 'outer' language for agent communication
  - The Knowledge Interchange Format (KIF), a language for expressing content, closely based on First Order Logic

Knowledge Interchange Format (KIF)

- KIF allows agents to express
  - properties of things in a domain, e.g., "Michael is a vegetarian"
  - relationships between things in a domain, e.g., "Michael and Janine are married"
  - general properties of a domain, e.g., "All students are registered for at least one course" (quantification ∀)
- Examples:
  - "The temperature of m1 is 83 Celsius":
    ```
    (= (temperature m1) (scalar 83 Celsius))
    ```
  - "An object is a bachelor if the object is a man and is not married":
    ```
    (defrelation bachelor (?x) :=
      (and (man ?x) (not (married ?x))))
    ```
  - "Any individual with the property of being a person also has the property of being a mammal":
    ```
    (defrelation person (?x) :=> (mammal ?x))
    ```

Knowledge Query and Manipulation Language (KQML) I

- KQML defines *communicative verbs*, or *performatives*, for example:
  - `ask-if` (‘is it true that. . . ’)
  - `perform` (‘please perform the following action. . . ’)
  - `tell` (‘it is true that. . . ’)
  - `reply` (‘the answer is . . . ’)
- Each message has a performative (the „class“ of a message) and a number of parameters

```
(ask-one
    :content (PRICE IBM ?PRICE)
    :receiver stockServer
    :language LPROLOG
    :ontology NYSE-TICKS
)
```

Asking about the price of IBM stock

Terminology

KQML II

Parameters of messages

Parameter	Meaning
`:content`	content of the message
`:language`	formal language the message is in
`:ontology`	terminology the message is based on
`:force`	will sender ever deny content of message?
`:reply-with`	reply expected? identifier of reply?
`:in-reply-to`	id of reply
`:sender`	sender ID
`:receiver`	receiver ID

KQML III
Example dialogs

Dialogue (a)

```
(evaluate
  :sender A :receiver B
  :language KIF :ontology motors
  :reply-with q1 :content (val (torque m1)))
(reply
  :sender B :receiver A
  :language KIF :ontology motors
  :in-reply-to q1 :content (= (torque m1) (scalar 12 kgf)))
```

Talking about motors

Query reference q1

Asking about torque of motor 1

Answer: "It is 12kgf"

Dialogue (b)

```
(stream-about
  :sender A :receiver B
  :language KIF :ontology motors
  :reply-with q1 :content m1)
(tell
  :sender B :receiver A
  :in-reply-to q1 :content (= (torque m1) (scalar 12 kgf)))
(tell
  :sender B :receiver A
  :in-reply-to q1 :content (= (status m1) normal))
(eos
  :sender B :receiver A
  :in-reply-to q1)
```

Streaming of messages, e.g. request all available knowledge

Indication of "End of Stream"

KQML IV
Criticisms

- The basic KQML performative set was overly large and not standardized
  - different implementations of KQML where developed that could not, in fact, interoperate
- The language was missing the performative *commissives*
  - Commissives are crucial for agents coordinating their actions.
- These criticisms - amongst others - led to the development of a new language by the FIPA consortium

Agent Communication Languages II

Foundation for Intelligent Physical Agents (FIPA)

- FIPA is the organization for developing standards in multi-agent systems. It was officially accepted by the IEEE at its eleventh standards committee in 2005
- FIPA's goal in creating agent standards is to promote inter-operable agent applications and agent systems
- FIPA ACL's syntax and basic concepts are very similar to KQML, for example:

```
(inform
   :sender agent1
   :receiver agent2
   :content (price good2 150)
   :language sl
   :ontology hpl-auction
)
```

FIPA ACL

Set of Performatives in FIPA ACL

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

FIPA ACL Performatives

Requesting Information

subscribe	sender asks to be notified when statement changes
query-if	direct query for the truth of a statement
query-ref	direct query for the value of an expression

FIPA ACL Performatives

Passing Information

inform	together with **request** most important performative; basic mechanism for communicating information; sender wants recipient to believe info; sender believes info itself
inform-ref	informs other agent about value of expression (in its content parameter); typically content of **request** message (thus asking the receiver to give me value of expression)
confirm	confirm truth of content (recipient was unsure)
disconfirm	confirm falsity of content (recipient was unsure)

FIPA ACL Performatives

Negotiation

cfp	call for proposals; initiates negotiation between agents; content-parameter contains action (desired to be done by some other agent) (e.g.: „sell me car“) and condition (e.g.: „price < 1000$“)
propose	make proposal
accept-proposal	sender accepts proposal made by other agent
reject-proposal	sender does not accept proposal

FIPA ACL Performatives

Performing Actions

request	issue request for an action
request-when	issue request to do action if and when a statement is true
request-whenever	issue request to do action if and whenever a statement is true
agree	sender agrees to carry out requested action
cancel	follows request; indicates intention behind request is not valid any more
refuse	reject request

FIPA Interaction Protocols (IPs)

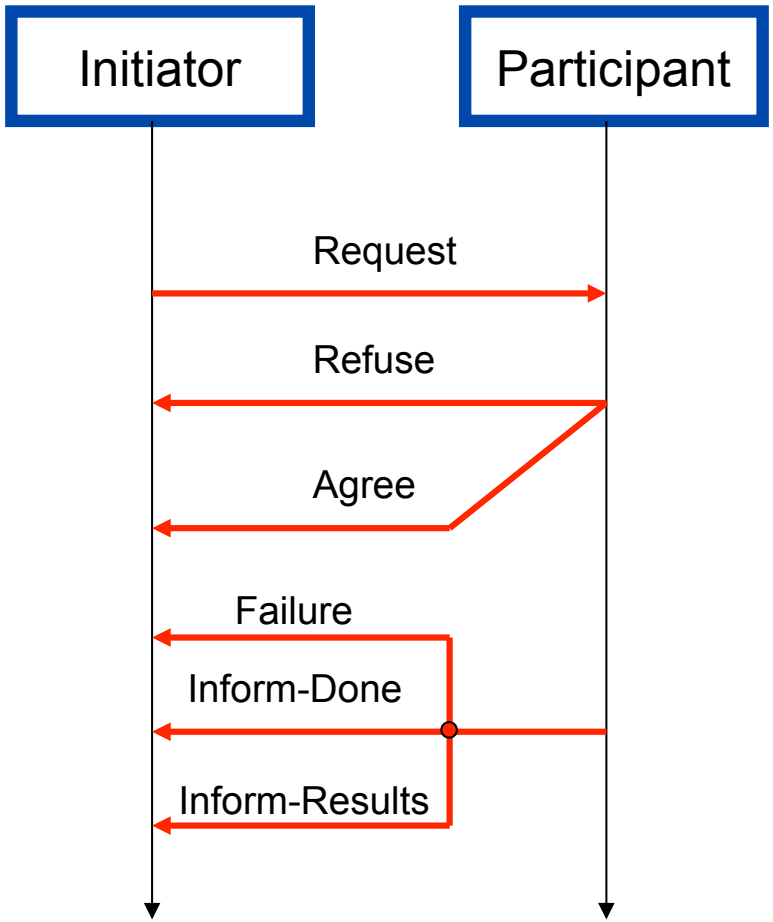
Interaction Protocols (IPs) are standardized exchanges of performatives according to well known situations

FIPA defined IPs are:

- FIPARequest
- FIPAQuery
- FIPARequestWhen
- FIPAContractNet
- FIPAIteratedContractNet
- FIPAAuctionEnglish
- FIPAAuctionDutch
- FIPABrokering
- FIPARecruiting
- FIPASubscribe
- FIPAPropose

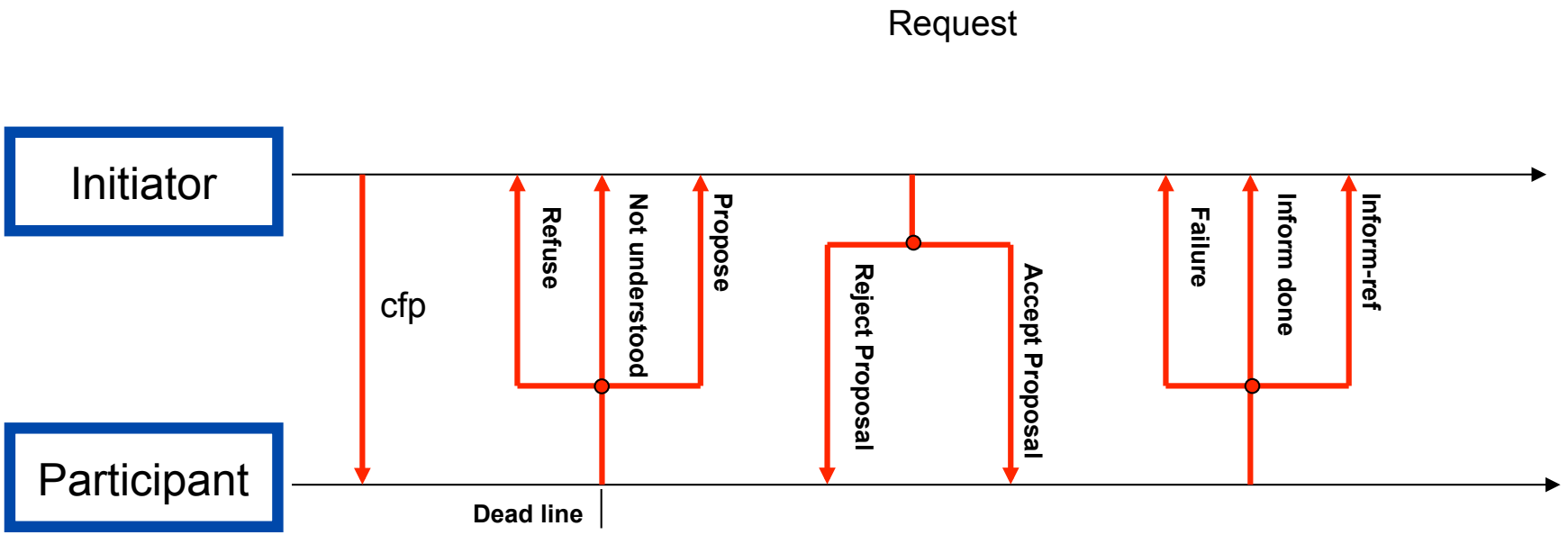
FIPA Interaction Protocols (IPs)

FIPA IP Example: Request



FIPA Interaction Protocols (IPs)

FIPA IP Example: Contract Net



Ontologies

- Ontologies ground the terminology used by the agents
  - For example, an agent wants to buy a screw. But what means then "size"? Is it in inch or centimeter?
- Very important in the Internet, sometimes encoded by XML
  - In contrast to HTML, whose meta-language mainly describes the page layout, XML allows to tag data with semantics → semantic web

(a) Plain HTML

```
<ul>
    <li><em>Music</em>,
        <b>Madonna<b>,
        USD12<br><p>
    <li><em>Get Ready</em>,
        <b>New Order</b>,
        USD14<br><p>
</ul>
```

(b) XML

```
<catalogue>
   <product type="CD">
      <title>Music</title>
      <artist>Madonna</artist>
      <price currency="USD">12</price>
   </product>
   <product type="CD">
      <title>Get Ready</title>
      <artist>New Order</artist>
      <price currency="USD">14</price>
   </product>
</catalogue>
```

Plain HTML vs. XML

Java Agent Development Framework (JADE)

- Open Source project originated by Telecom (TILAB), currently governed by an international board, e.g. Motorola, France Telecom, Whitestein, …
- JADE allows the rapid creation of distributed, multi-agent systems in Java
- High interoperability through FIPA compliance
- JADE includes:
  - A library for developing agents (which implements message transport and parsing)
  - A runtime environment allowing multiple, parallel and concurrent agent activities
  - Graphical tools that support monitoring, logging, and debugging
  - Yellow Pages, a directory where agents can register their capabilities and search for other agents and services

JADE II
Connectivity

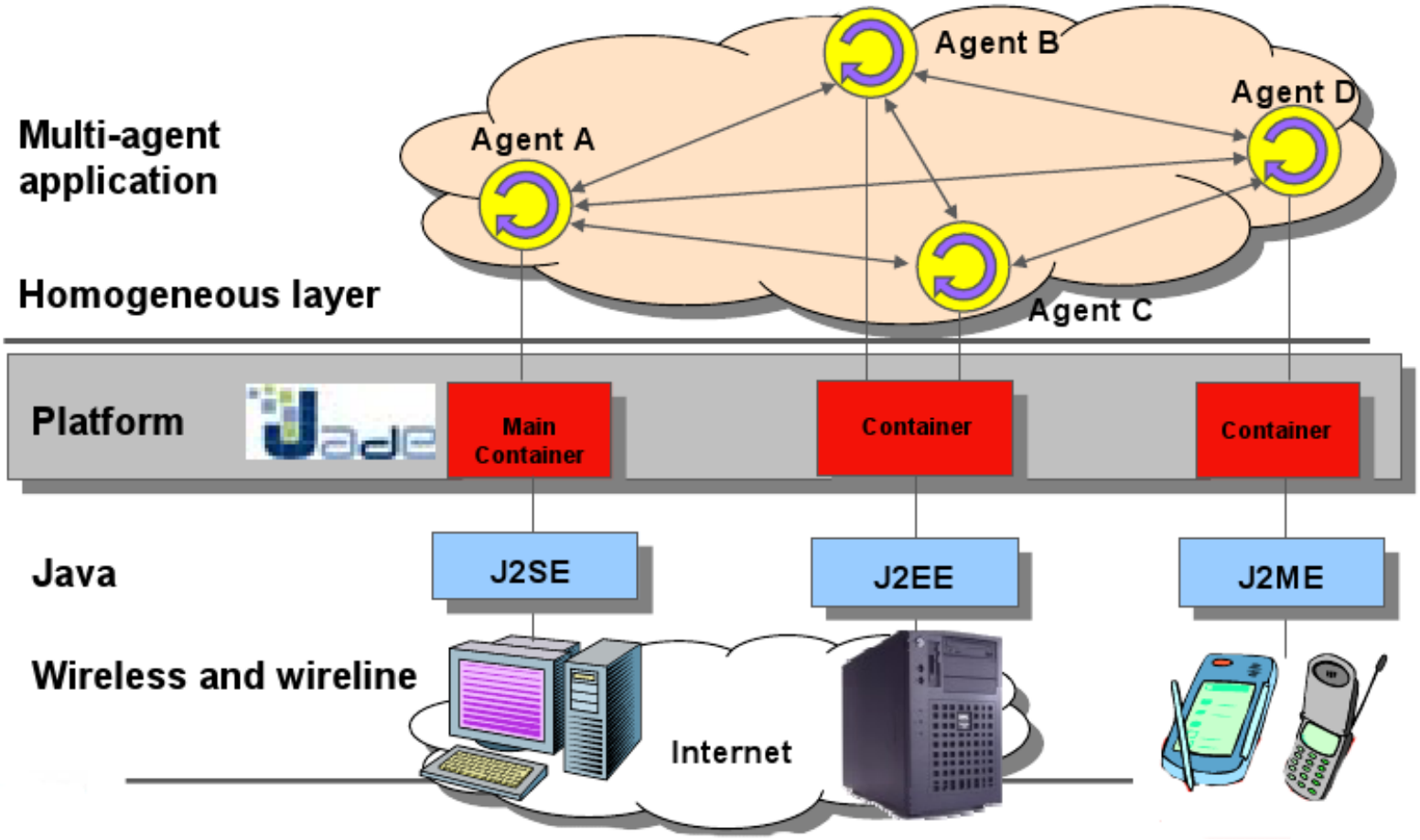


Image taken from the Jade Tutorial

JADE III
Code Example

```
public class AgentThatSearchesAndUseAService
    extends jade.core.Agent
{
    public void setup()
    {
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setType("SearchedService");
        DFAgentDescription[] agents = DFService.search(this,dfd);
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver(agents[0].getAID();
        msg.setContent("execute service");
        send(msg);
        System.out.println(blockingReceive());
    }
}
```

Note DF means “Directory Faciliator”, an agent for accessing the yellow pages

JADE IV
Behaviors

- JADE Behaviors
  - A behavior is basically an event handler, a method which describes how an agent reacts to an event: the reception of a message or a timer interrupt
  - The Event Handler code is placed in a method called action. Every behavior is scheduled following a round robin algorithm.
- Methods of the agents involving behaviors:
  - *addBehaviour* & *removeBehaviour*
- Examples of Behaviors already included in JADE:
  - SimpleBehavior - CyclicBehavior
  - TickerBehavior - WakerBehavior
  - ReceiverBehavior - SequentialBehavior
  - ParallelBehavior - FSMBehavior

JADE V
Debugging: “Dummy Agent”

- Functionalities:
  - compose and send a custom messages
  - load/save the queue of messages from/to a file

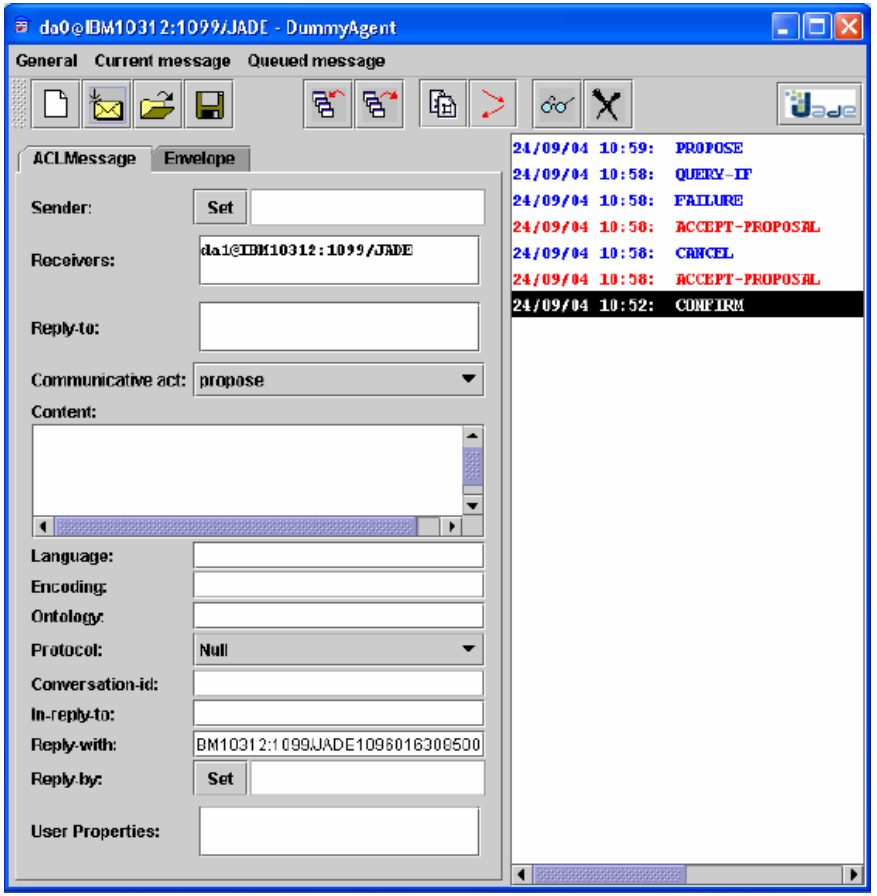


Image taken from the Jade Tutorial

JADE VI
Debugging: “Sniffer Agent”

- Functionalities:
  - display the flow of interactions between selected agents
  - display the content of each exchanged message
  - save/load the data flow

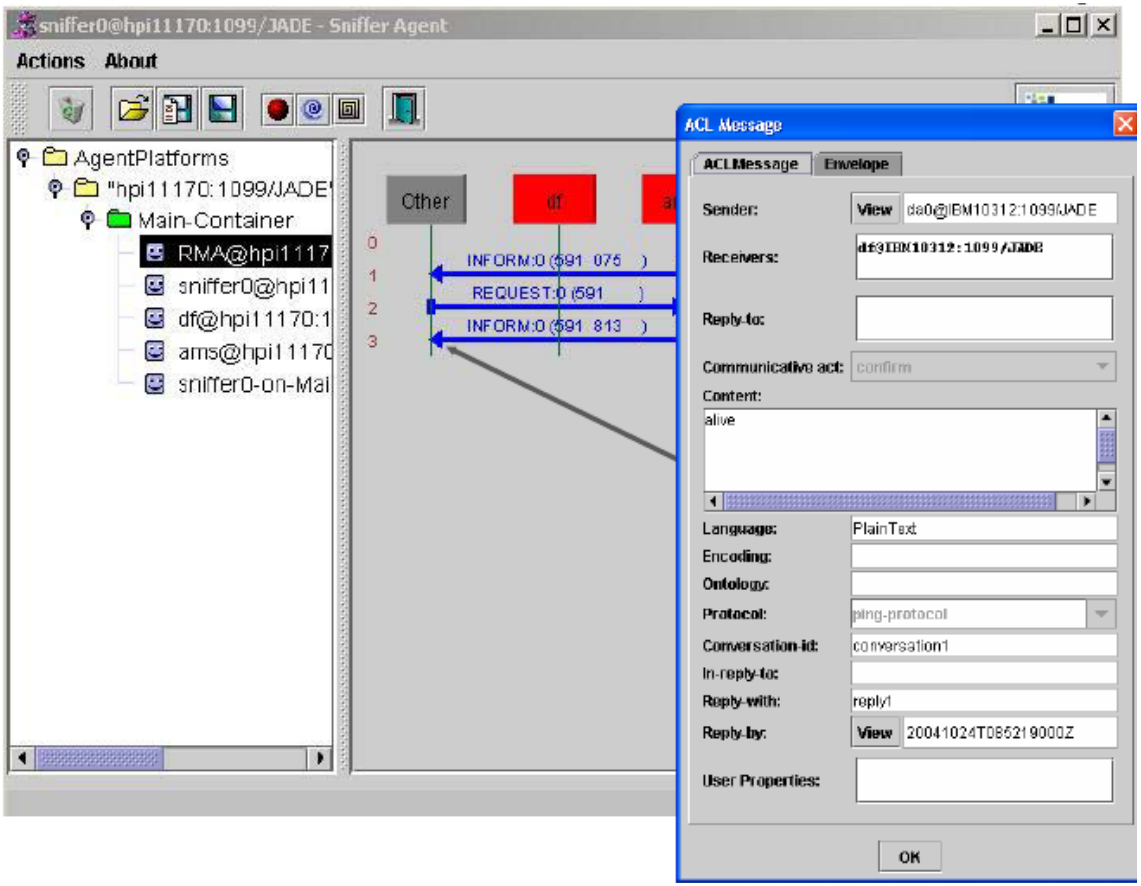


Image taken from the Jade Tutorial

JADE VII
Debugging: “Log Manager Agent”

- Functionalities:
  - browse all Logger objects on its container (both JADE-specific and application-specific)
  - modify the logging level
  - add new logging handlers (e.g. files)

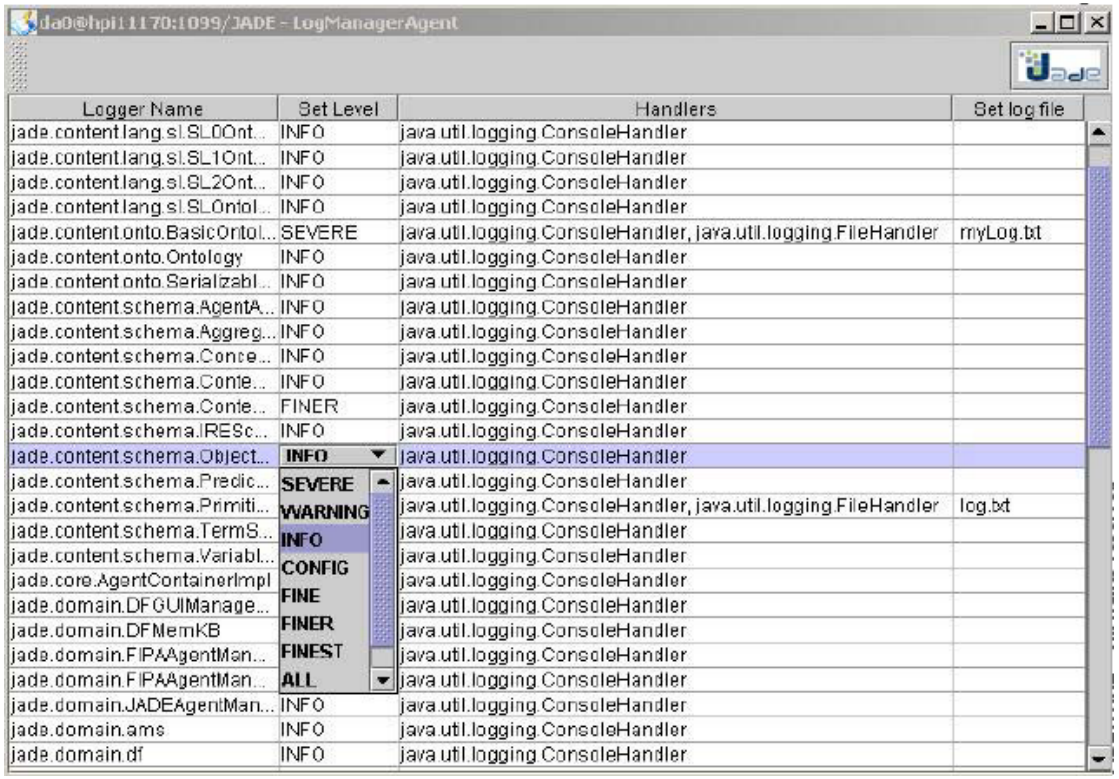


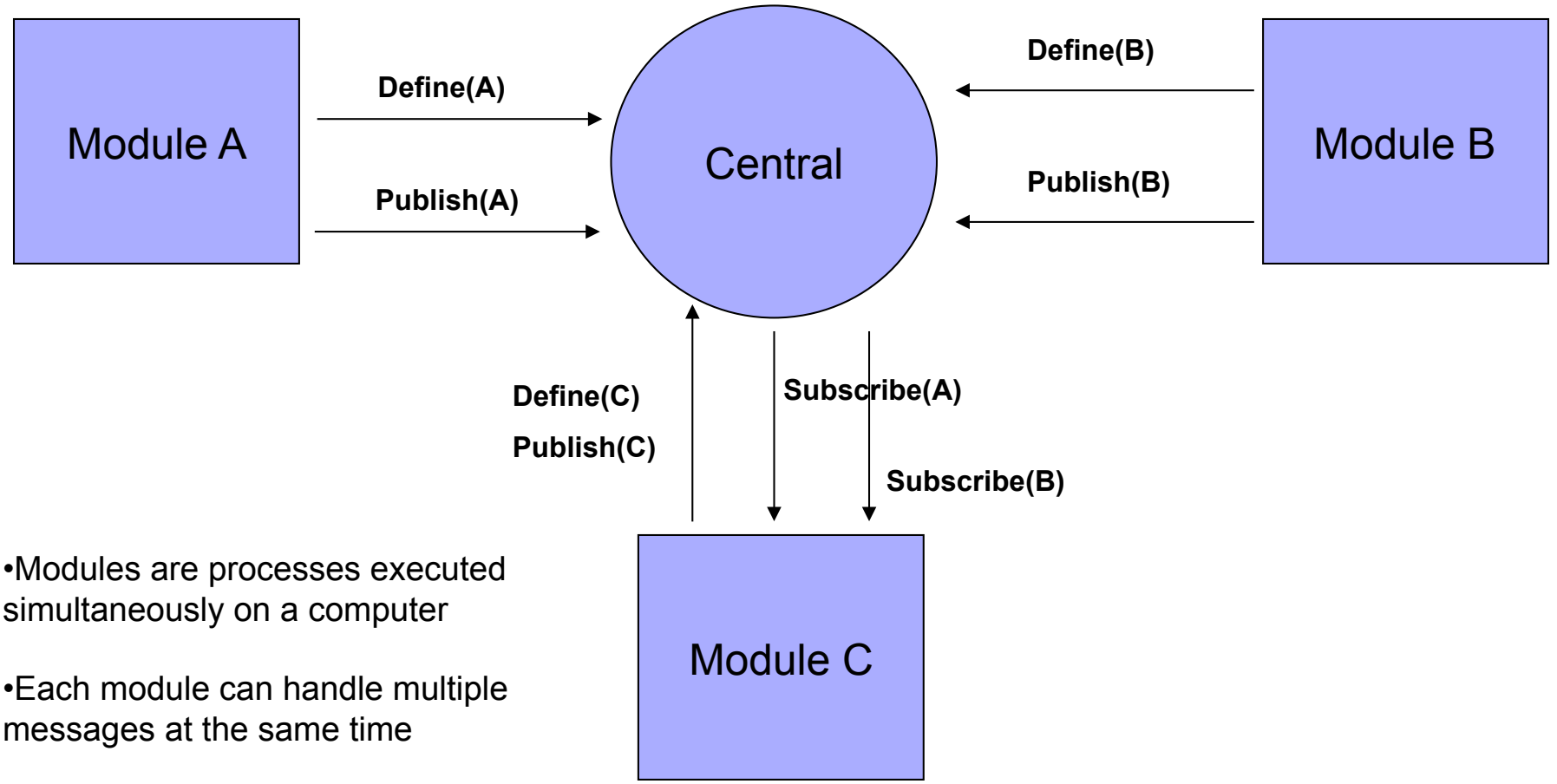
Image taken from the Jade Tutorial

Inter Process Communication (IPC)

- NOT an ACL but an efficient tool within fully cooperative & distributed environments
- Very similar to ROS, the framework from WillowGarage (PR2)
- Platform-independent library for distributed network-based message passing, runs with C,C++, Lisp, and JAVA
- Provides facilities for client/server and publish / subscribe communication
  - Communication takes place either point-to-point or via a "central" thread, whereas the latter allows data logging and visualization
- Marshalling and passing of complex data structures
- Has been used by our group during RoboCup, the Sick Race, and the TechX challenge

IPC Communication Models I

Publish/Subscribe

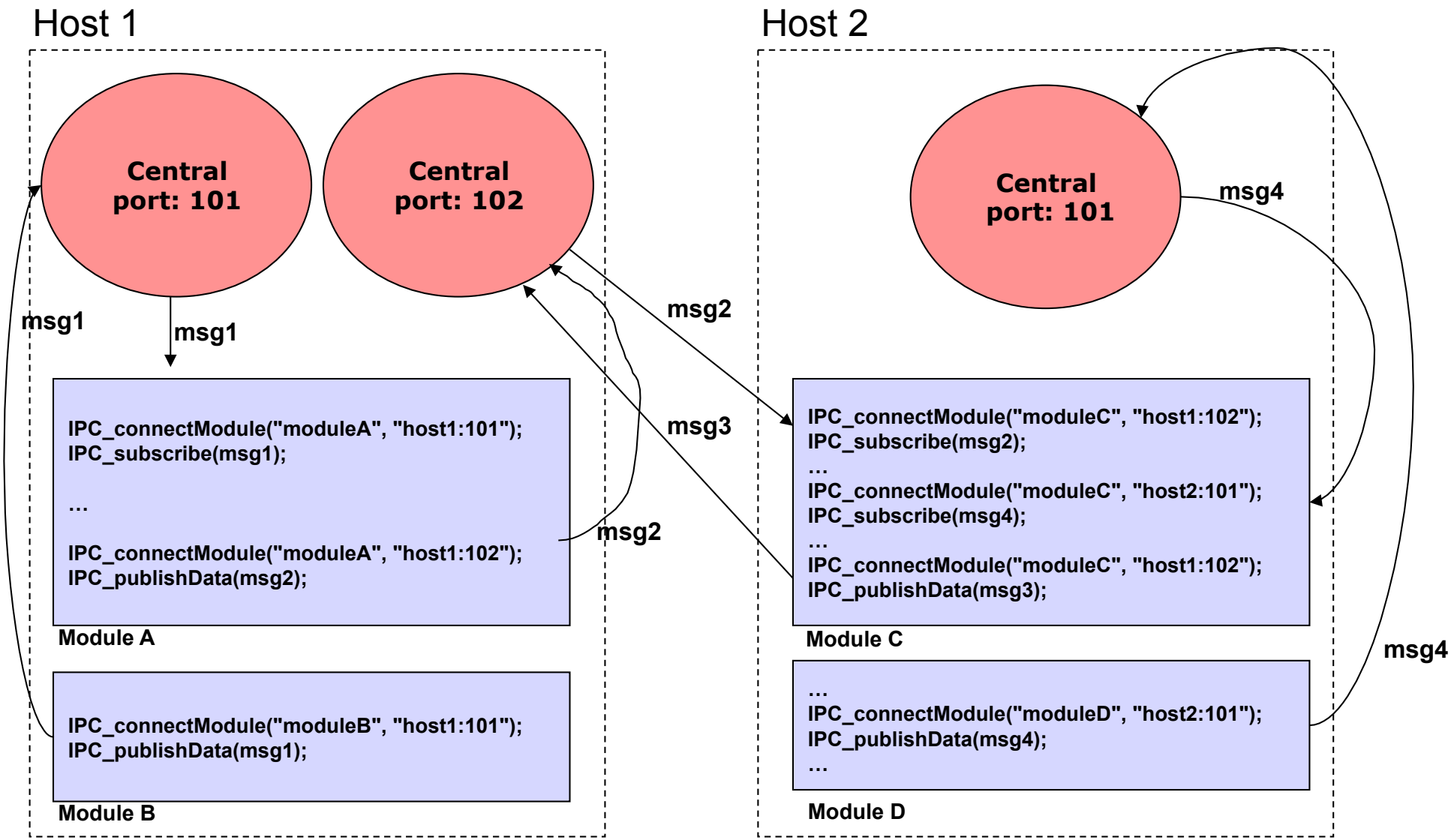


- Modules are processes executed simultaneously on a computer
- Each module can handle multiple messages at the same time

Publish/Subscribe
Module Architecture

```
module MODULE_C
static: quit, dataA, dataB
quit ← false
dataA ← NULL
dataB ← NULL

CONNECT-TO-CENTRAL()
SUBSCRIBE-HANDLER(msgHandlerA, dataA)
SUBSCRIBE-HANDLER(msgHandlerB, dataB)
DEFINE_MESSAGE(msgC)
while (not quit) do
        listen_for_messages()
        dataC ← PROCESS-DATA(dataA, dataB)
        PUBLISH-DATA(dataC)
End

Function msgHandlerA(dataA)
        UPDATE-DATA(dataA)
End

Function msgHandlerB(dataB)
        UPDATE-DATA(dataB)
End
```

Distributed execution

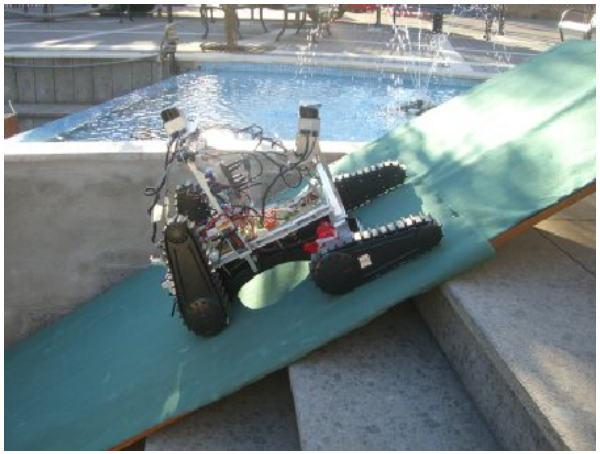


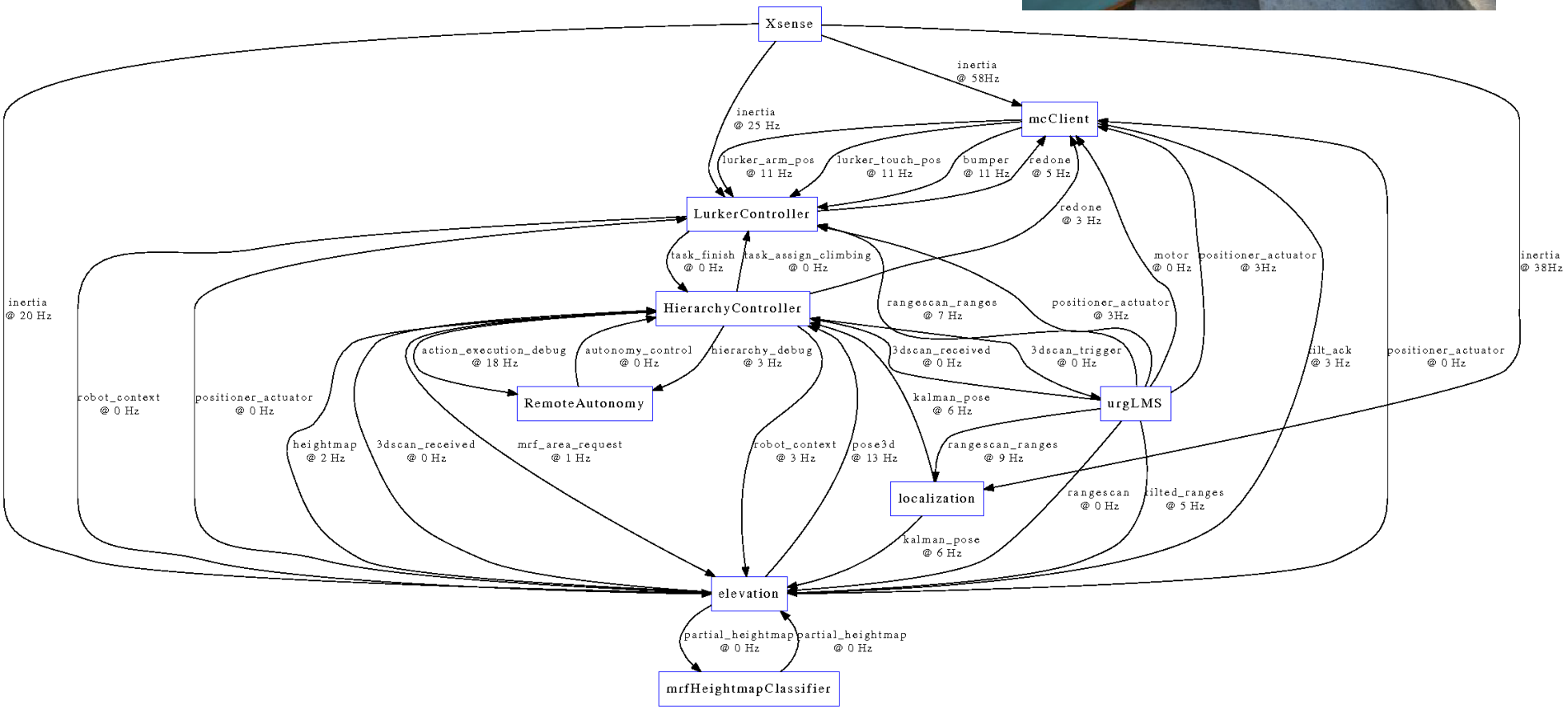
IPC Data Formats
Examples in C

```
#define RESCUE_BATTERY_STATUS_NAME  "rescue_battery_status"
#define RESCUE_BATTERY_STATUS_FMT   "{double, double, double, string}"
typedef struct {
   double level;                ///< [V]
   double capacityLeft;         ///< [0, 1] How full is the battery (estimated)
   double timestamp;
   char* host;
} rescue_battery_status_message;


#define RESCUE_JOYPAD_BUTTON_NAME  "rescue_joypad_button"
#define RESCUE_JOYPAD_BUTTON_FMT   "{int, double, string}"
//AUTOLOGGER LOGGER_PRINTF "Jb "
typedef struct {
   int button;
   double timestamp;
   char* host;
} rescue_joypad_button_message;

...
```

IPC Example I

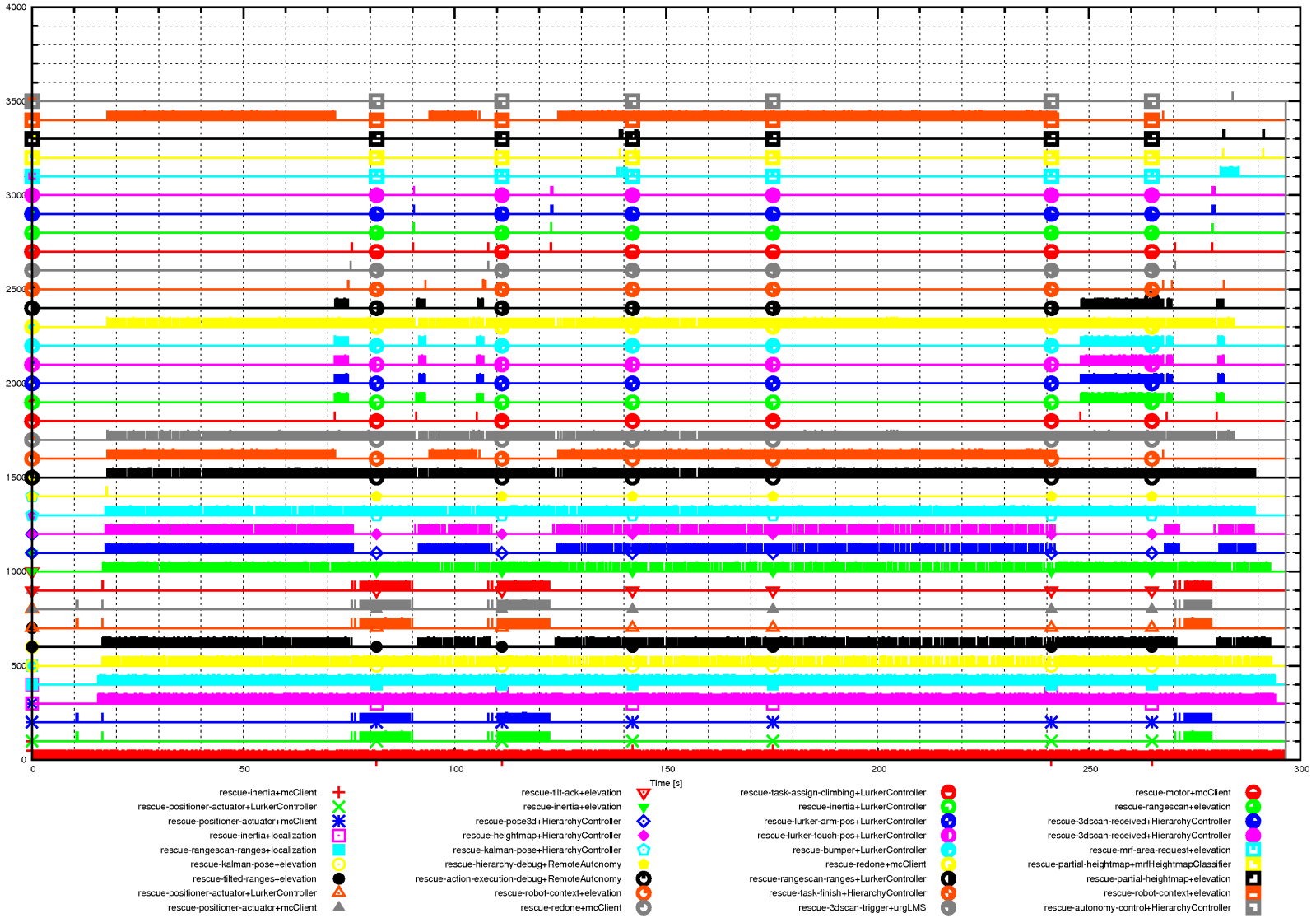
Autonomous Lurker Robot





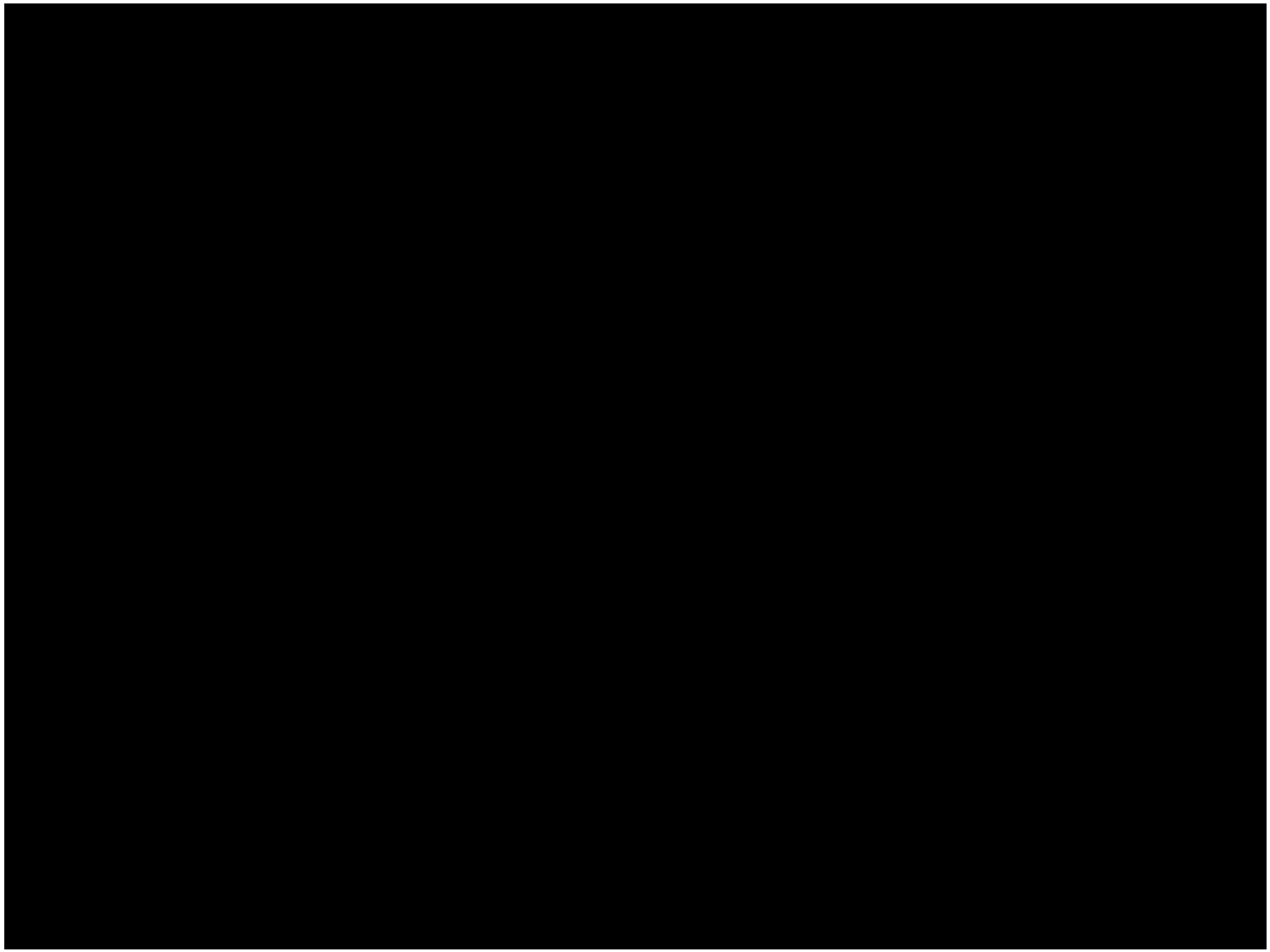
IPC Example I

Lurker Communication Graph



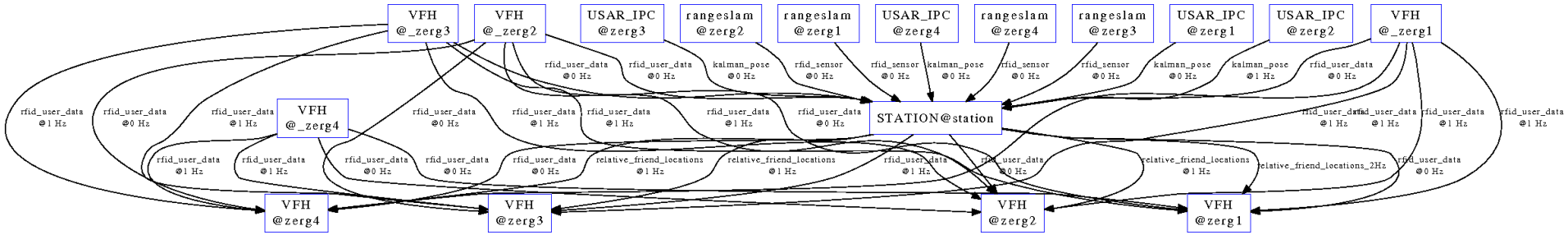
IPC Example I

Video Lurker Exploration (IROS`07)



IPC Example II

Autonomous team of Zerg Robots



IPC Example III

Fast integration with Micro Aerial Vehicle (MAV)

We integrated our robot system within only one day with a MAV developed by another team from Sweden

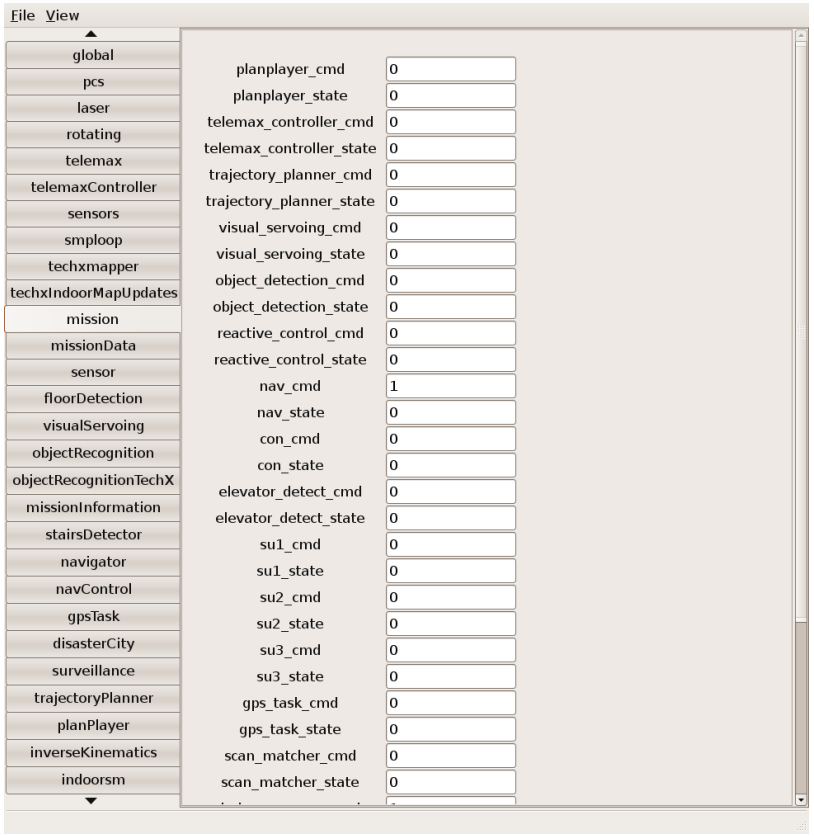
IPC Communication Models II

Parameter Daemon

- In a complex system composed of various modules, global parameters have to be handled somehow
- A parameter daemon is a separate module that reads parameters from a single configuration file
  - Stores specific parameters (typically fixed during runtime), but also module status information and commands (changing during runtime)
- Communication through "parameter changes"
  - Can be considered as blackboard system
  - Modules can install handler for parameter changes
- Implemented by publish/subscribe

Parameter Daemon

Examples



Interface for mission control: each module‘s action state can be set and the status read

Specific parameters of “stairsDetector”

Summary

- ACLs provide standards for communication among selfish agents, e.g. within an open systems
- Motivated from the theory of speech acts, communication is implemented in terms of actions
- The FIPA ACL can be considered as the de facto standard for agent communication
  - The JADE framework implements it in JAVA
- IPC/ROS offers all necessary functionality within fully cooperative and distributed environments
  - It is very efficient and simple to use

Literature

- M. Woolridge: **An Introduction to Multi-Agent-Systems**, Wiley, 2001, 294 pages
- Searle, J.R., **Speech Acts** *Cambridge University Press*, 1969
- FIPA:
  - Website http://www.fipa.org
  - Agent Interaction Protocols (http://www.fipa.org/repository/ips.php3)
- JADE
  - Website http://sharon.cselt.it/projects/jade/
  - Tutorial: http://www.iro.umontreal.ca/~vaucher/Agents/Jade/ JadePrimer.html
- IPC:
  - Website http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html