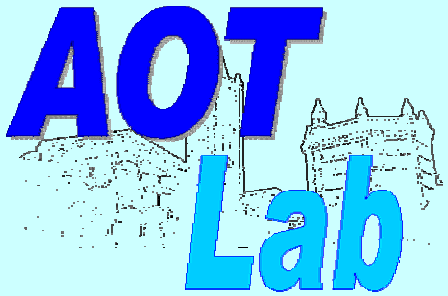


AOT Lab
Dipartimento di Ingegneria
dell'Informazione
Università degli Studi di Parma

Agent-Oriented Software Engineering

Federico Bergenti and Paola Turci

{bergenti, turci}@ce.unipr.it

With contributions of Massimo Cossentino, Onn Shehory and Franco Zambonelli

AOT Lab

Goal of the Lecture

- Understand and discuss
  - What Agent-Oriented Software Engineering (AOSE) is and why it is important
  - Key concepts
- Overview
  - Relevant AOSE methodologies
  - AOSE implementation tools
- Suggest interesting research directions

AOT Lab

Part 1

Key Concepts of Agent-Oriented Software Engineering

AOT Lab

What is Software Engineering

- Software is pervasive and critical
  - It cannot be built without a disciplined, engineered, approach
- There is a need to model and engineer both
  - The development process
    - Controllable, well documented, and reproducible ways of producing software
  - Software
    - Well-defined quality level (e.g., % of bugs and performances)
    - Enabling reuse and maintenance
- Requires
  - Abstractions, methodologies and tools

AOT Lab

Abstractions of Software Engineering

- Software deals with “abstract” entities, having a real-world counterpart
  - Numbers, dates, names, persons, documents, ...
- In what term shall we model them in software?
  - Data, functions, objects, agents, …
  - I.e., what are the abstractions that we have to use to model software?
- May depend on available technologies

AOT Lab

Methodologies

- A methodology for software development is intended discipline the development
- Defines the abstractions to use to model software
  - Data-oriented, flow-oriented, object-oriented, …
  - Define the mindset of the methodology
- Disciplines the software process
  - What to produce and when
  - Which artefacts to produce

AOT Lab

Tools

- Notation tools
  - To represent the outcome of the software development phases
    - Diagrams, equations, figures, …
- Formal models
  - To prove properties of software prior to development
    - Lamba calculus, Petri-nets, Z, ….
- CASE tools
  - To facilitate activities: rapid prototyping, code generators, …

AOT Lab

Why Agent-Oriented Software Engineering?

- Software engineering is necessary to discipline
  - Software systems and software processes
  - Any approach relies on a set of abstractions and on related methodologies and tools
- Agent-based computing introduces novel abstractions
  - Requires clarifying the set of necessary abstractions
  - Requires adapting methodologies and producing new tools
- Novel, specific agent-oriented software engineering approaches are needed

AOT Lab

What are Agents?

- There has been some debate
  - On what an agent is, and what could be appropriately called an agent
- Two main viepoints
  - The (strong) artificial intelligence viewpoint
    - An agent must be, proactive, intelligent, and it must conversate instead of doing client-server computing
  - The (weak) software engineering viewpoint
    - An agent is a software component with internal (either reactive or proactive) threads of execution, and that can be engaged in complex and stateful interactions protocols

AOT Lab

What are Multiagent Systems?

- Again….
  - The (strong) artificial intelligence viewpoint
    - A multiagent system is a society of individual that interact by exchanging knowledge and by negotiating with each other to achieve either their own interest or some global goal
  - The (weak) software engineering viewpoint
    - A multiagent system is a software systems made up of multiple independent and encapsulated loci of control (i.e., agents) interacting with each other in the context of a specific application viewpoint

AOT Lab

The Software Engineering Viewpoint

- We commit to it because
  - It focuses on the characteristics of agents that have impact on software development
    - Concurrency, interaction, multiple loci of control
    - Intelligence can be seen as a peculiar form of control independence
    - Conversations can be seen as a peculiar form of interaction
  - It is much more general
    - Does not exclude the strong AI viewpoint
    - Several software systems (even if never conceived as agents-based one) can be indeed characterised in terms of weak multi-agent systems

AOT Lab

Agent-Oriented Abstractions

- The development of a multiagent system should fruitfully exploit abstractions coherent with the above characterisation
  - Agents, autonomous entities, independent loci of control, situated in an environment, interacting with each others
  - Environment, the world of resources agents perceive
  - Interaction protocols, as the acts of interactions between agents

AOT Lab

Getting Deeply into AOSE...

- For the definition of a suitable methodology for multiagent systems development
  - There is need of better characterizing agents, multiagent systems, and the associated mindset of abstractions
    - How can we model agent autonomy, situatedness and sociality
  - There is need of understanding how the “traditional” cascade software engineering process maps into agent-oriented software development
    - What are analysis and design in AOSE?

AOT Lab

Characterizing Agents

- No agreement on the definition of agent
- Historically, two approaches to characterize "intelligent", i.e., rational, agents and multiagent systems
  - Operational, agents and multiagent systems are systems with particular features, i.e.
    - Particular structure
    - Particular behaviour
  - Based on system levels, agents and multiagent systems are new system levels
- These approaches are complementary

AOT Lab

Operational Characterization

- Particularly suited for rational agents because it is based on logics
- Rational agents (Wooldridge)
  - Described in terms of a belief, desires and intention
  - Beliefs, desires and intentions are structured so to make the agent behave rationally
  - Independent from the internal agent architecture
  - The whole work on *LORA* devoted to it

AOT Lab

Operational Characterization

- Simple control loop of a rational agent

```
1 forever
2   sense the environment
3   update the model of the environment
4   deliberate for a new goal
5   means-end reason to obtain a plan
        to achieve the goal
6   execute the plan
```

AOT Lab

Operational Characterization

- The operational characterization
  - Draws from well-founded logics
  - Does not depend on the internal architecture of the agents
- This approach has, at least, two problems
  - Does not justify reasonably why we should adopt agents instead of other technologies
  - Grounds rationality on the axioms of a logic
  - Could not make any accepted agreement

AOT Lab

System Levels

- System level

  Structured group of concepts that support the definition of an engineered model of a system

- Historically, introduced to hide details in hardware design, e.g.
  - A logic gate level design does not care about transistors
  - A register transfer level design does not care about gates
- System levels are levels of abstraction

AOT Lab

System Levels

- A system level is composed of
  - Medium, set of atomic concepts that the system level processes
  - Components, atomic concepts that we use to assembly the system
  - Composition laws ruling how components can be assembled to form a system
  - Behaviour laws determining how the behaviour of the system depends on
    - The behaviour of the single components
    - The architecture of the system

AOT Lab

Example: Logic Gate Level

Element	*Logic Gate Level Element*
System	Unit of a processor that manipulates registers
Components	Logic gates, lines
Medium	Single-bit signals
Composition Law	E.g., input and output of logic gates are connected through lines
Behaviour Law	The laws for composing truth tables of logic gates

AOT Lab

Knowledge Level

- At the beginning of the 80's the AI had the problem of defining knowledge
- Introduced a new system level, called knowledge level, to provide a scientific definition of knowledge (Newell)
- The knowledge level is used to model agents, i.e., rational systems that process knowledge

AOT Lab

Knowledge Level

Element	*Knowledge Level Element*
System	Agent
Components	Goal, action, body
Medium	Knowledge
Behaviour Law	Principle of rationality

- In order to model an agent, we need
  - A body, i.e., a means for the agent to interact with its environment
  - A set of actions the agent can perform on its environment. Each action has pre- and post-conditions
  - A set of goals

AOT Lab

Knowledge Level

- The knowledge level
  - Relies only on the principle of rationality to account for the behaviour of the agent
  - Focuses on modelling one single agent
- Today, we build systems in terms of
  - Agents that may not be proved to be rational at all
  - Interacting agents that are the unit of reuse and of encapsulation

AOT Lab

Social Level

Element	*Social Level Element*
System	Organization
Components	Agent, organizational relation, interaction channel, dependency
Medium	Obligation, influence mechanisms
Behaviour Law	Principle of organizational rationality

- Jennings introduced the social level on top of the knowledge level
- It allows to create organizational models of multiagent systems

AOT Lab

Social Level

- The social level
  - Moves all design towards social issues, does not care of how to design each agent
  - Cannot describe emerging organizations
- Best practice of architectural patterns suggests that organization is not enough to design a system, e.g., we need
  - Connectors for flexible composability
  - Contracts to support verifiable composability

AOT Lab

Agent Level

Element	*Agent Level Element*
System	Multiagent system
Components	Belief, goal, action, role, interaction rule
Medium	Representation of belief, goal and capabilities
Behaviour Law	Principle of rationality

- Between the knowledge and the social level
- Allows to model multiagent systems that
  - Rely on message passing and on the speech-act theory
  - Exploits the possibilities of the FIPA infrastructure

AOT Lab

Agents and Other Technologies

- Since FIPA, multiagent systems are often compared with object-oriented systems
  - Both rely on encapsulated units that interact
  - Both rely on message passing
  - For both we can define an architecture
  - … and many other similarities
- The comparisons found in the literature are often poor

AOT Lab

"Agents can say..."

- Use of autonomy to draw a line between agents and objects (Parunak)
  - *"Agents can say go"*, agents can take the initiative
  - *"Agents can say no",* agents can refuse to perform a requested service
- These seems relevant differences with object-oriented method invocation, but
  - Active objects have a long and honored history
  - Refusal is not useful per se

AOT Lab

Comparing the Meta-Models

Element	*Objects*	*Agents*
State	Properties and values	Knowledge base
Messaging	Request for service with certain parameters	Exchange of parts of the knowledge base
Reuse	Inheritance, mostly for composability	Composability
Delegation	Task delegation	Goal delegation
Responsibility	Design by contract	Pre-/post-conditions
Type system	Classes	None

AOT Lab

Comparing Granularity

- Objects have a highly dynamic lifecycle, they are
  - Created just for serving a request
  - Cloned just for performance reasons
  - Introduced to promote reusability
  - …often created and destroyed
- Agents are more coarse grained
  - Reason on their knowledge bases
  - Publish their capabilities to a DF
  - …they are rarely created and destroyed

AOT Lab

Part 2

Overview of Agent-Oriented Software Engineering Methodologies

AOT Lab

What is a Methodology ?

- A methodology is…
  - a collection of methods for solving a class of problems
  - a body of methods, rules, and postulates employed by a discipline: a particular procedure or set of procedures
- But…when referring to software
  - A methodology is the set of guidelines for covering the whole lifecycle of system development both technically and managerially
    - full lifecycle process
    - comprehensive set of concepts and models
    - full set of techniques (rules, guidelines, heuristics)
    - fully delineated set of deliverables
    - modeling language
    - set of metrics
    - quality assurance
    - coding (and other) standards
    - reuse advice
    - guidelines for project management

AOT Lab

Agent-Oriented Methodologies

- There is need for agent-oriented methodologies
  - Centered around specific agent-oriented abstractions
  - The adoption of OO methodologies would produce mismatches
    - Classes, objects, client-servers: little to do with agents
- Each methodology may introduce further abstractions
  - Around which to model software and to organize the software process
    - E.g., roles, organizations, responsibilities, belief, desire and intentions, …
  - Not directly translating into concrete entities of the software system
    - E.g. the concept of role is an aspect of an agent, not an agent

AOT Lab

Agent-Based Analysis

- Analysis aims to understand, at least
  - What are the main actors interacting with the system
  - How the system interacts with these actors
  - What the system is supposed to do
- The system is a closed entity and we do not look into it to avoid anticipating design issues and decisions
- Where do agents enter the picture?

AOT Lab

Agent-Based Analysis

- We associate agents with the entities of the scenarios we are analyzing
- Then, we associate accordingly
  - Roles, responsibilities and capabilities
  - Interaction patterns between agents
- This provides a neutral view of the problem.
- Methodologies, e.g., Tropos and GAIA, do not use the word agent to identify analysis-phase entities

AOT Lab

Agent-Based Design

- Design aims to engineer, at least
  - What are the main components interacting within the system
  - What are the responsibilities and the capabilities of each component in the system
  - How the components interact to implement the system, i.e., the architecture of the system
- Where do agents enter the picture?

AOT Lab

Agent-Based Design

- We associate agents with the components we use to build the system
- Then, we associate accordingly
  - Roles, responsibilities and capabilities
  - Interaction patterns between agents
- Differently from analysis: we need to choose on which agents to use and how they interact
- Agents at the design phase can have nothing to do with agents at the analysis phase

AOT Lab

Agent-Oriented Methodologies

- Several methodologies and approaches for designing MASs exist in literature. In general they tackle different aspects of the MAS and in some cases they are quite complementary:
  - **Gaia**
    - Encourages a developer to think of building agent-based systems as a process of *organisational design.*
  - **TROPOS**
    - It is founded on the concepts of goal-based requirements adopted from the i* and GRL (Goal-oriented Requirements Language). Its distinguishing feature is the emphasis on requirements analysis
  - **Prometeus**
    - It focuses mainly on the internal agent architecture; it is basically a methodology for designing BDI agent systems
  - **ADELFE**
    - It is a methodology for the development of adaptive multiagent systems
  - **MESSAGE**
    - It covers most of the fundamental aspects of the MAS development, focusing mainly on analysis and high-level design. The main objective was to combine the best features of the pre-existing approaches, but … the result was a too complex and farraginous methodology.
  - **PASSI**
    - It is a step-by-step requirement-to-code methodology. Integrates design models and concepts from both object oriented software engineering and artificial intelligence approaches
  - … and many others

AOT Lab

Gaia Methodology

- Gaia is appropriate for the development of systems with the following main characteristics:
  - Gaia is not intended for systems that admit the possibility of *true conflict*.
  - Gaia makes no assumptions about the delivery platform;
  - The organisation structure of the system is static, in that inter-agent relationships do not change at run-time.
  - The abilities of agents and the services they provide are static, in that they do not change at run-time.
  - The overall system contains a comparatively small number of different agent types (less than 100).

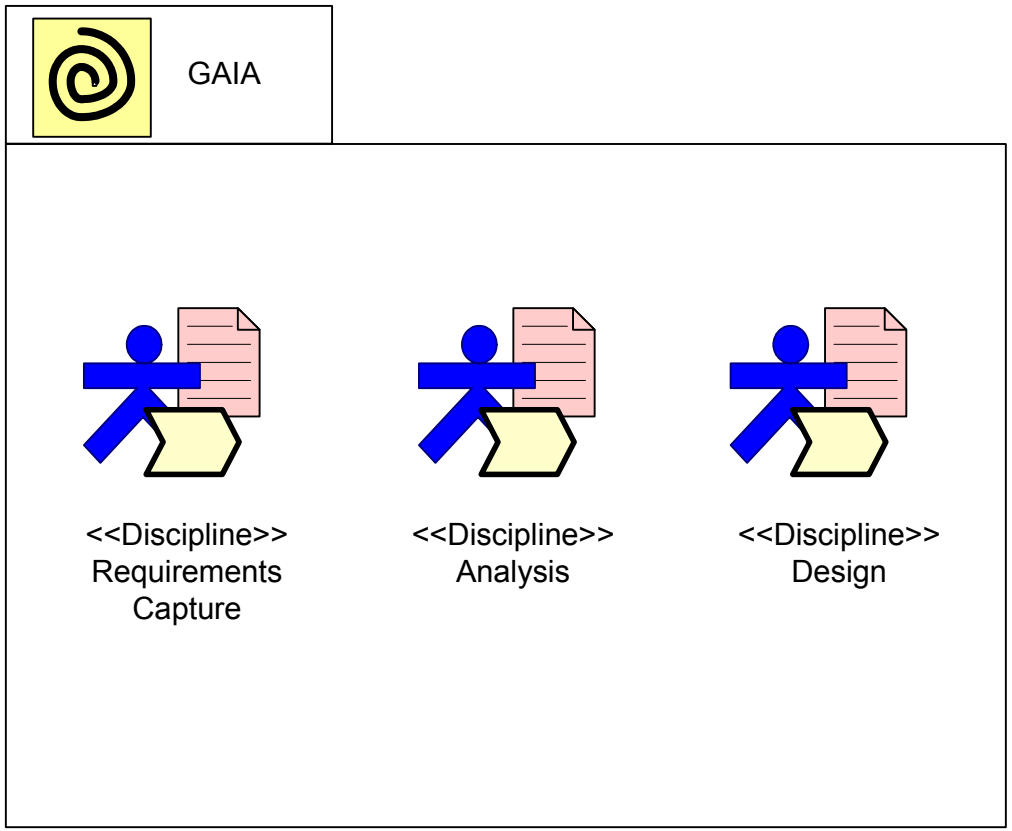
Case Study

Auction agent

1. The *configurator*: a GUI component, enables the user to control and monitor the agent's activity
2. The *parser*: translates retrieved information into an internal structure
3. The *bidder*: submits bids according to buying strategy. Implements two stages, bid and confirmation
4. The *manager*: controls the agent's activity, monitors the auction site, activates the parser, determines the next bid, activates the bidder and terminates the agent's purchasing activity

AOT Lab

Gaia Disciplines



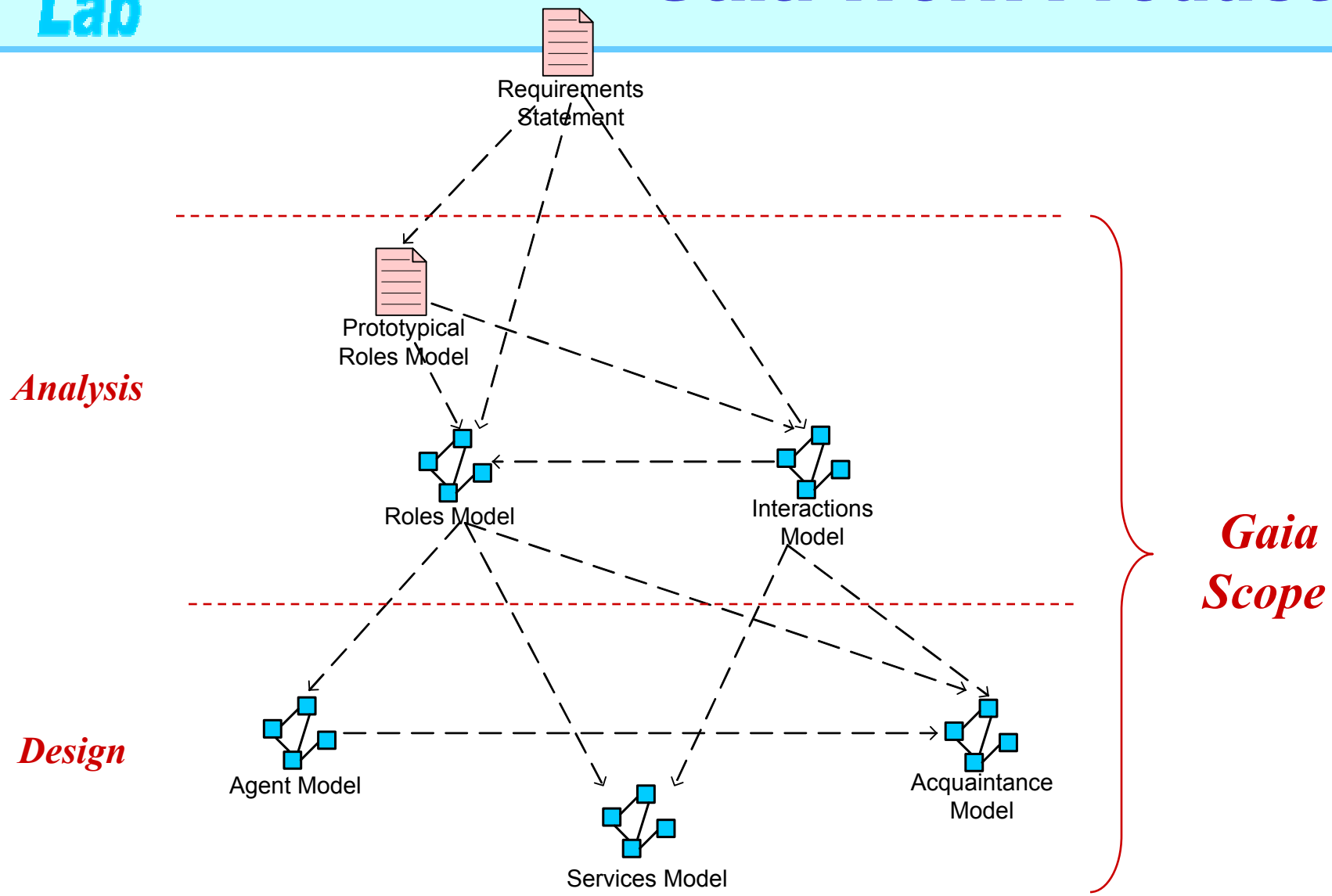
AOT Lab

GAIA Disciplines

- Requirements capture phase are considered independent of the paradigm used for analysis and design
  - For this reason Gaia does not deal with the requirements capture phase
- The analysis phase consists of the following models:
  - Role definition (permissions, responsibilities and protocols)
  - Interaction model (used for protocol description)
- The design phase consists of the following models:
  - Agent model
  - Service model (input, output, pre and post condition)
  - Acquaintance model

AOT Lab

Gaia Work Products



- Work products dependency diagram

AOT Lab

GAIA – Role Model

Role Schema:	*name of role*
Description	*short English description of the role*
Protocols and Activities	*protocols and activities in which the role plays a part*
Permissions	*"rights" associated with the role*
Responsibilities	
Liveness	*liveness responsibilities*
Safety	*safety responsibilities*

- Template for role schemata

- Protocols, state the interactions of the role with other roles. In addition state the internal activities of the role
- Permissions, state what resources may be used to carry out the role and what resource constraints the role's executor is subject to
- Responsibilities. determine the functionality of the role. This functionality is expressed in terms of safety and liveness properties

GAIA – Role Model

Role Schema: Manager (MA)

Description:
Controls the auction agent activities

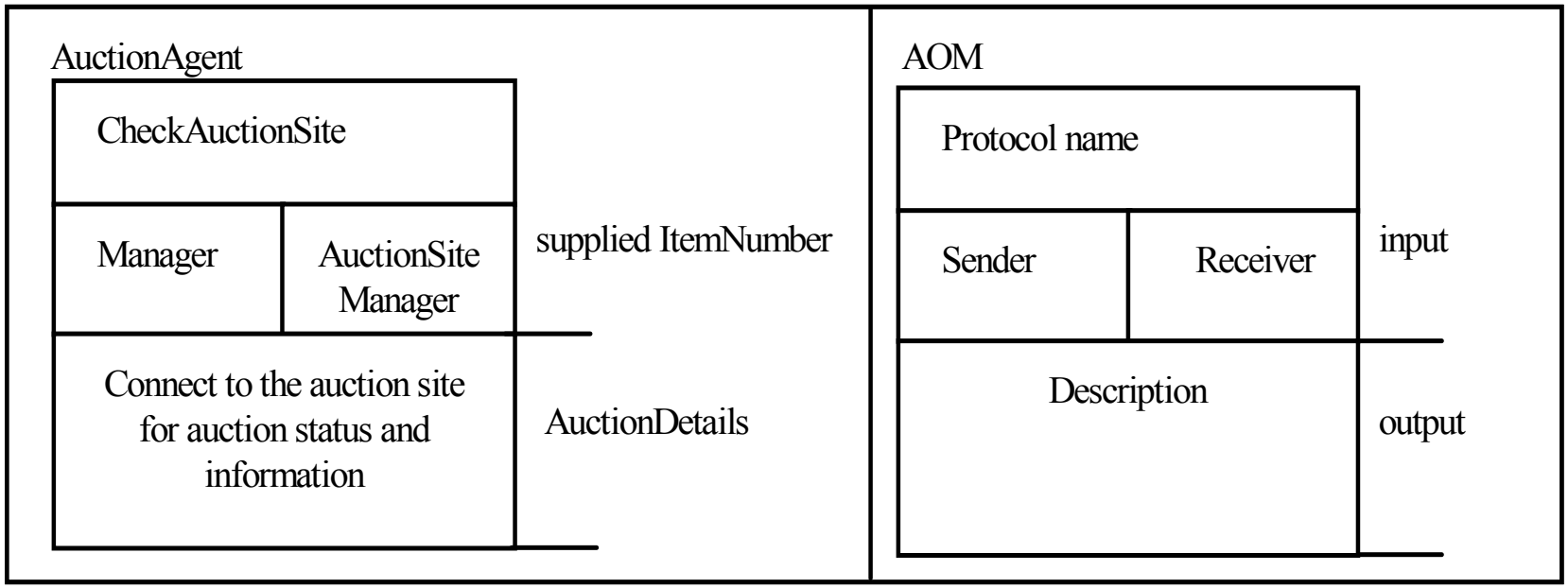
Protocol and Activities:
CheckAuctionSite , ActivateParser ,CheckForBid , Bid

Permission:
reads supplied ItemNumber // the item number in the auction site
AuctionDetails // the auction information

Responsibilities:
Liveness:
Manager = (CheckAuctionSite .ActivateParser .CheckForBid)$^{+}$[Bid]
Saftey :
true

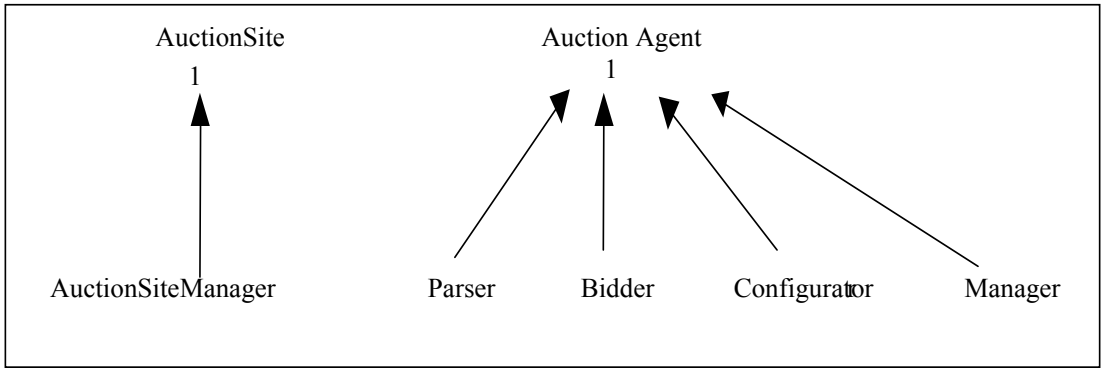
– *The Manager role scheme*

GAIA – Interaction Model



- The Interaction Model of the CheckAuctionSite protocol

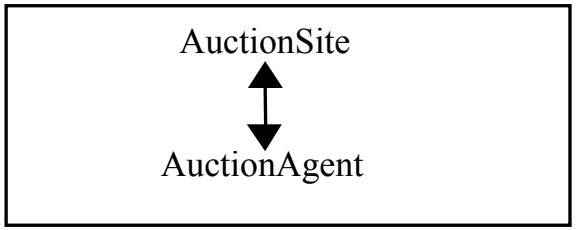
GAIA – Design Phase Models



- The Agent Model

Service	Input	Output	Pre-condition	Post-condition
Get auction details	ItemNumber	AuctionDetails	true	true
Validate user	User	Exists	true	(exists=true) ∨ (exists=false)
Bid	User, ItemNumber, Price	Success	user exists	(success=true) ∨ (success=false)

- The Service Model



- The Acquaintance Model

AOT Lab

Gaia v.2

- First version
  - Designed to handle small-scale, closed agent-based systems
  - Modelled agents, roles, interactions
  - Missed in modelling explicitly the social aspects of a MAS
- Official extension of Gaia
  - Thought for open agent systems
    - Significantly extends the range of applications to which Gaia can be applied
  - Focused on the social organization of the system
    - Three further abstractions, which can hardly be expressed in terms of individual roles or individual interaction protocols
      - Environmental model
      - Organizational rules
      - Organizational structures

AOT Lab

Environmental Model

- The environment elected to a primary analysis and design abstraction
  - Represented in terms of abstract computational resources, such variables or tuples, made available to the agents for sensing, for effecting or for consuming
    - The simplest form can be a list of all the entities and resources a MAS may interact with, restricting the interactions by means of the permitted actions
      - Es. In a manufacturing pipeline the resources of interest can be the items being produced and made flow in each of the pipeline's "n" stages. The environmental model can be:

        changes *flux[i]*, i=1,n *//number of items flowing in each stage of the pipeline*

AOT Lab

Organizational Rules

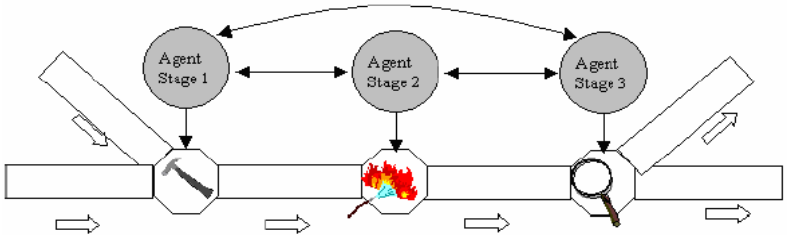
- Considered as *responsibilities* of the organization as a whole
  - In an open system, the organization should be able to enforce its internal coherency
  - Certain requirements are mostly simple expressed as global organizational rules rather than being replicated for each role in the organization
- Can be expressed by making use of the same formalism adopted for specifying liveness and safety rules for roles
  - Express constraints on the execution activities of roles and protocols
    - Es: In a manufacturing pipeline, the correct management of the pipeline requires each of the stage roles to be played only once. This can be expressed by the safety rule:

$$R^1,R = (STAGE[1], STAGE[2], \ldots, STAGE[N])$$

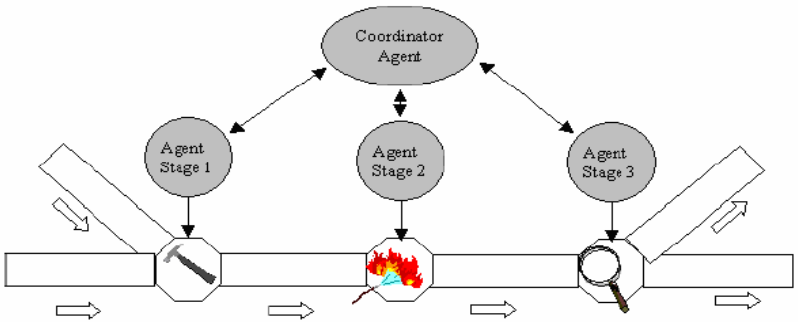
Organizational Structures

- The role model may define the organizational structure in an implicit way
- But …

  the structure of a MAS is more appropriately derived from the explicit choice of an appropriate organizational structure
  - Organizational structures viewed as first-class abstractions
    - Define the topology and the control regime
  - Opportunities for re-use and exploitation of organizational-patterns



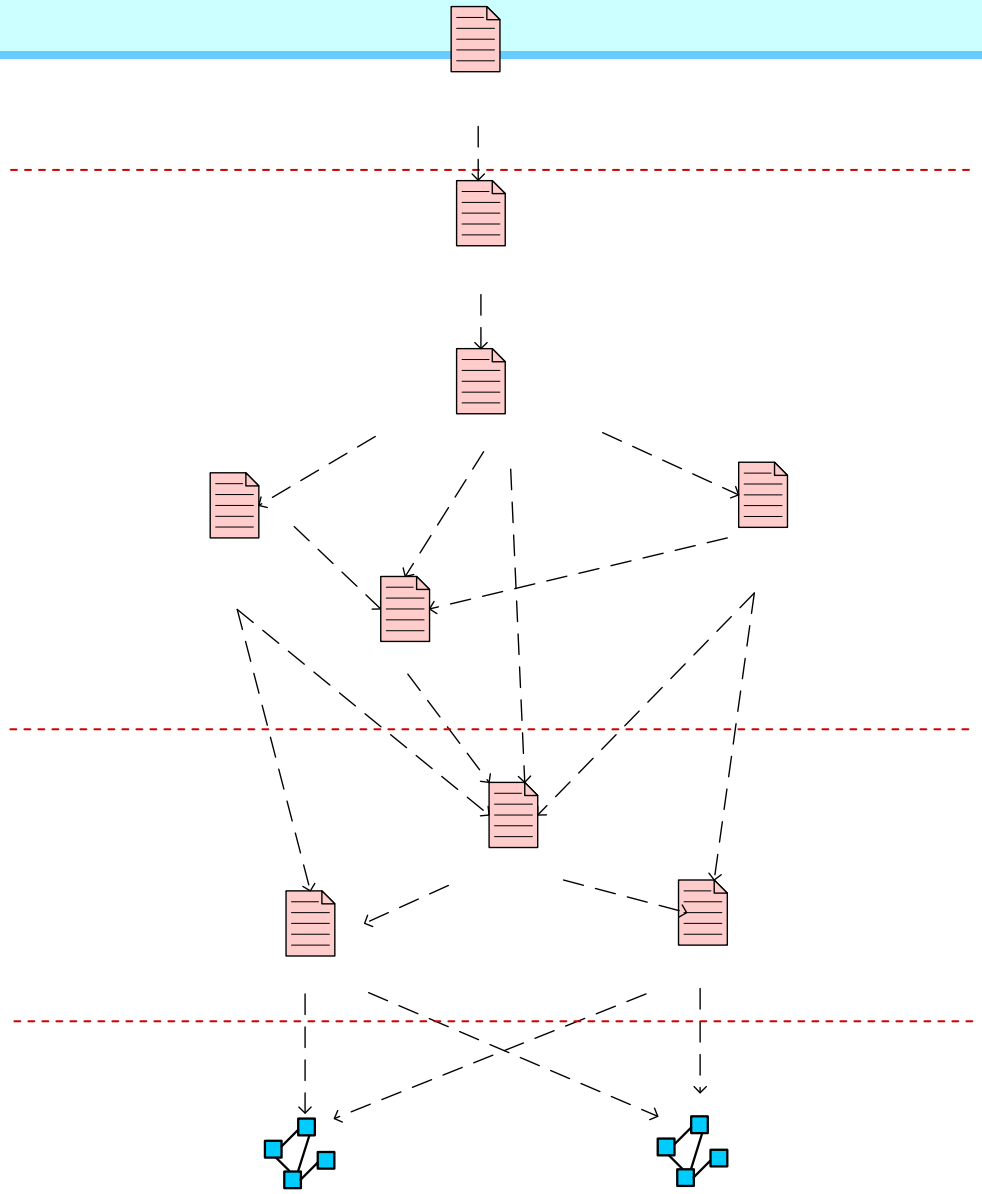
- Manufacturing pipeline: collective of peers organization



- Manufacturing pipeline: hierarchical organization

AOT Lab

Gaia v.2 - Work Products



Standardization Activities - FIPA

- FIPA is a world-wide, non-profit association of companies and organizations (http://www.fipa.org).
- At the beginning of the year 2003 two new Technical Committees were established, belonging to the software engineering area:
  - FIPA Modelling TC
  - FIPA Methodology TC
- Several researchers, already working in those areas, converged in these Technical Committees with the purpose of preparing the future standard in their specific field

AOT Lab

FIPA Modeling TC

- This TC was established to draw the specifications for the future ***FIPA agent-based unified modelling language*.**
  - Its work started from the existing experiences of UML and AUML.
  - The main objective is to facilitate advances in the state of the art of agent-based modelling.
  - The TC activities are in the first phase of a multiphase effort. The first phase is devoted to defining three diagrams:
    - Class diagram superstructure metamodel (basic foundational elements required in multi-agent systems)
    - Class diagram
    - Interaction diagrams
  - In addition, the TC participants have identified other modelling areas useful for representing and specifying agent-based systems

- **Extension Mechanisms**
  - Constraint - is a semantic relationship among model elements that specifies conditions and propositions that must be maintained as true; otherwise, the system described by the model is invalid
  - Stereotype - is a new class of metamodel element that is introduced at modelling time
  - Tag - a tag definition specifies the tagged values that can be attached to a kind of model element
  - Comment - is a text string attached directly to a model element

AOT Lab

FIPA Modeling - Agent Class Diagram

- The FIPA AUML Agent Class diagram is based on UML class diagram
  - UML class diagram:

    "A diagram that shows a collection of declarative (static) model elements such as classes, types, and their contents and relationships" [Booch, 1999]
  - Due to the relative complexity of agent design and the many differences that exist between an agent and an object, FIPA AUML class diagram extends UML class diagram to cover needs for agent design
    - FIPA AUML class diagram describes the agents and their architectures
  - Very preliminary form!!

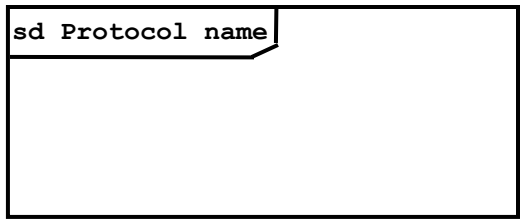
AOT Lab

FIPA Modeling: Sequence Diagram

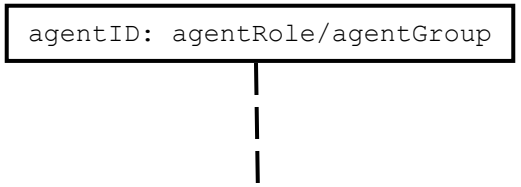
- A diagram that shows agent interactions arranged in time sequence. In particular, it shows the agents participating in the interaction and the sequence of messages exchanged
- Two dimensions:
  - Vertical dimension represents the time ordering
  - Horizontal dimension represents different roles or agents playing specific roles.
- Based on UML 2.0
- More structured than previous AUML sequence diagram
- (graphical) notation (lack of semantics)
- Still in draft form!

AOT Lab

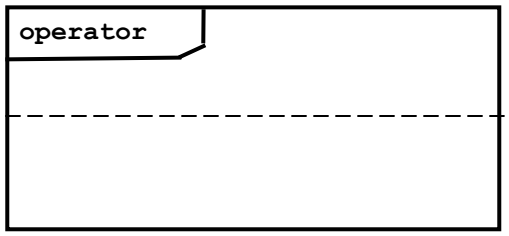
Graphical Notation



- Defines the frame of the sequence diagram. It contains the message sequence, the agent role, and the constraints on the interaction



- Lifelines shape the presence of agents in interaction. In Fig. is reported the complete form for the label: an agent identifier, a role and the group of the agent.

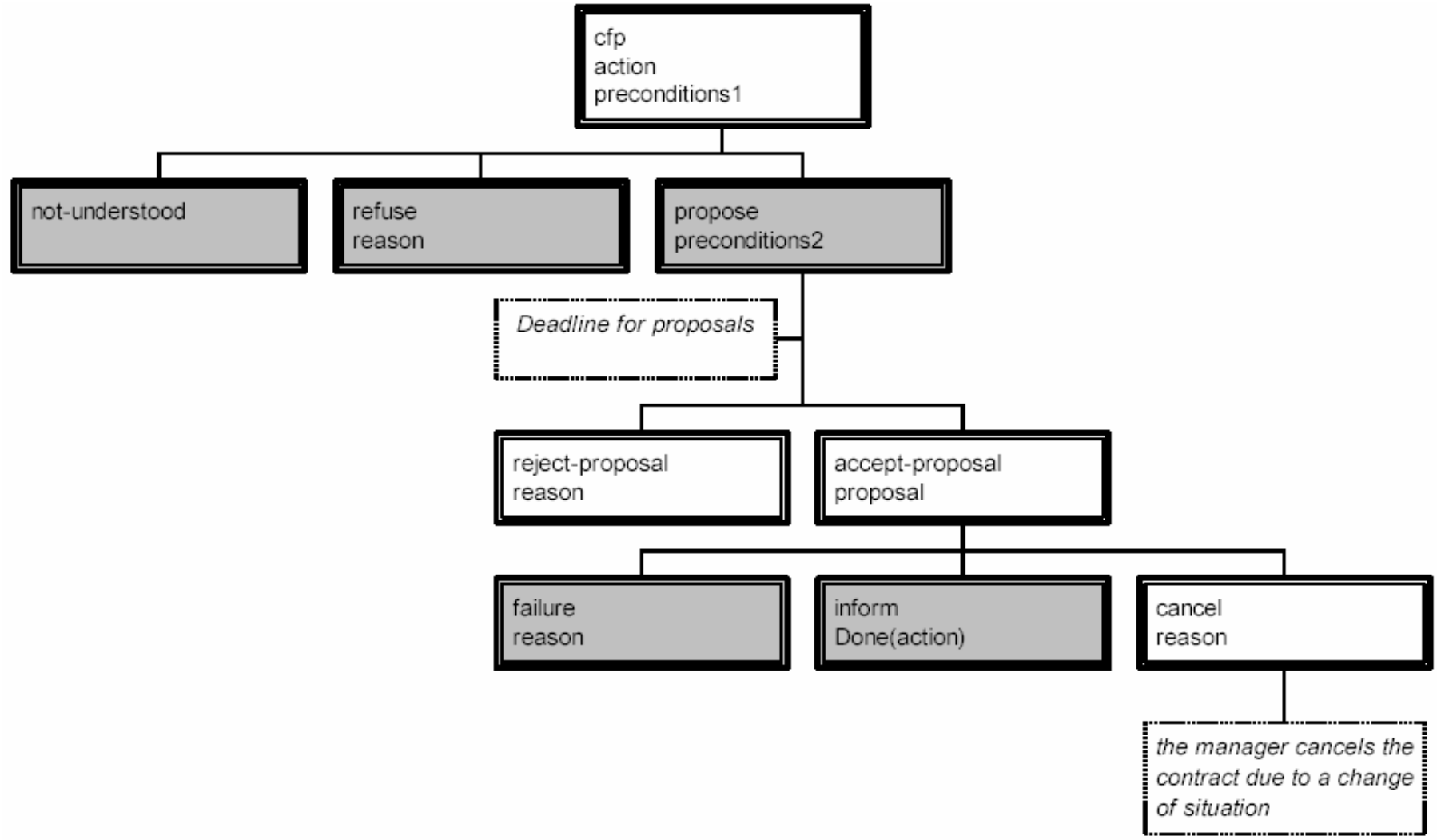


- The splitting operator is written inside the “snipped” corner pentagon. It must be alternative, option, break, parallel, loop ... A dashed vertical line separates each path from others

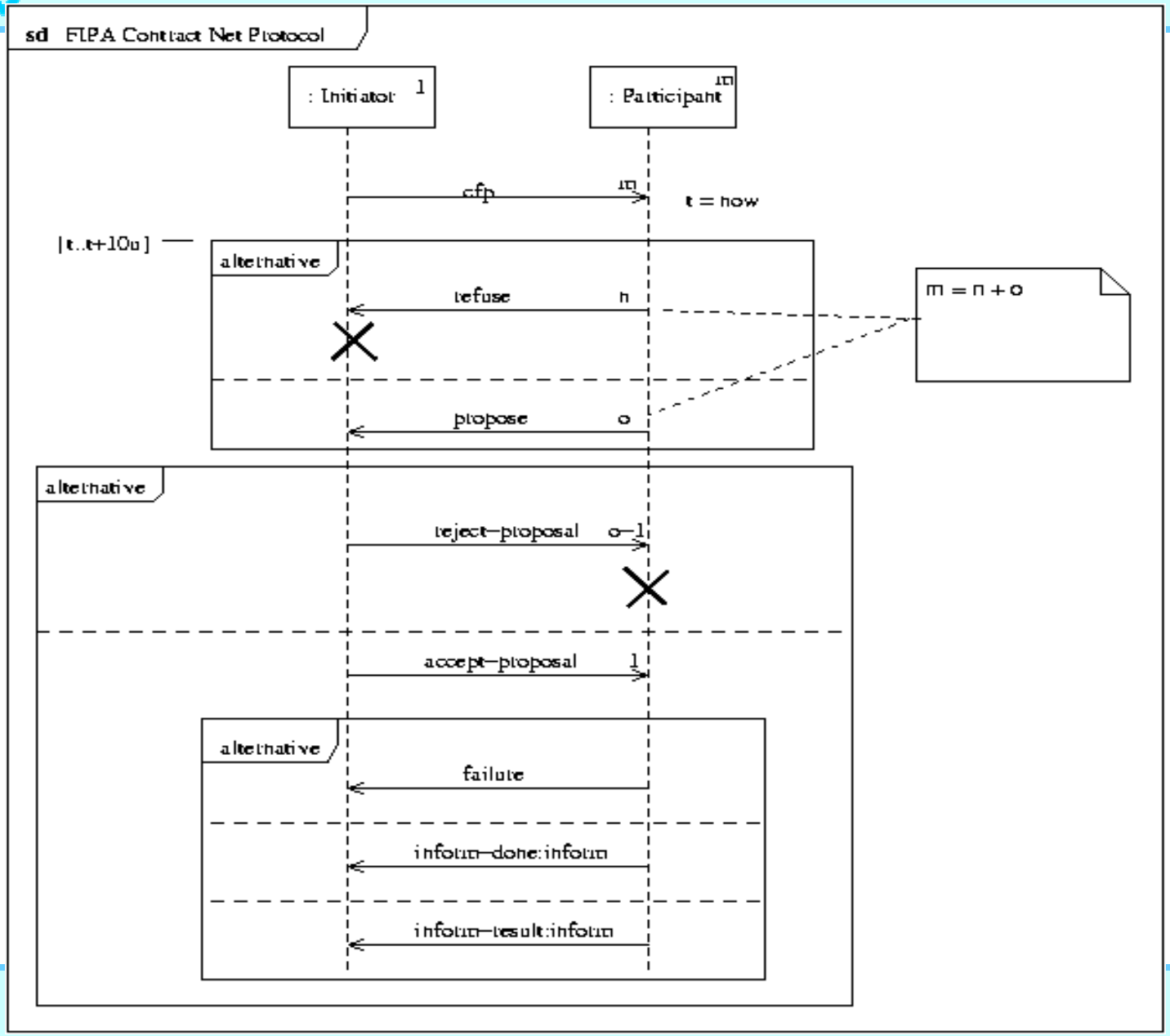
- Termination; the message path is stopped after this operator

AOT Lab

FIPA Contract Net Protocol



Sequence Diagram - Example



AOT Lab

Method Engineering

- From a quick overview of the major methodologies for the development of MASs, a crucial question arises:

"Can we have a universal methodology that is suitable for every problem domain?"

… the answer is probably "NO!"

- The best and feasible solution could be for developers to compose suitable methods for their problem domains and development environments, using phases or models or elements coming from different methodologies.
  - In the the object-oriented community the need for systematic principles to develop situation-specific methods has led to the emergence of the ***method engineering***

AOT Lab

FIPA Methodology TC

- Objective: identifying a methodology for the development of MAS that may fit the greatest number of needs.
  - Based on the *method engineering*, it consists in the definition of a sort of meta-methodology that could be instantiated for each specific problem.
    - The TC aims at collecting experiences coming from the existing contributions and re-organize them at a meta-model level.
  - Goals:
    - ***Creation of the meta-model***, It is necessary to formally represent method fragments in order to store them in the method base
    - ***Identification of the method base architecture***
    - ***Collection of method fragments***, the repository will be initially populated with method fragments extracted by the most diffused methodologies
    - ***Description of techniques for method integration***

AOT Lab

MAS Meta-Model

- Design a system ≅ Instantiate a meta-model
  - OO context - design rely on a common denominator
    - Universally accepted concept of object and related meta-model of object-oriented systems
  - AO context - to date, no common denominator
    - Each methodology has its own concepts and system structure

In the agent world
*the meta-model is the **critical element***
*when applying the **method engineering** paradigm*

AOT Lab

Part 3

Agent-Oriented Tools

AOT Lab

Agent-Oriented Tools

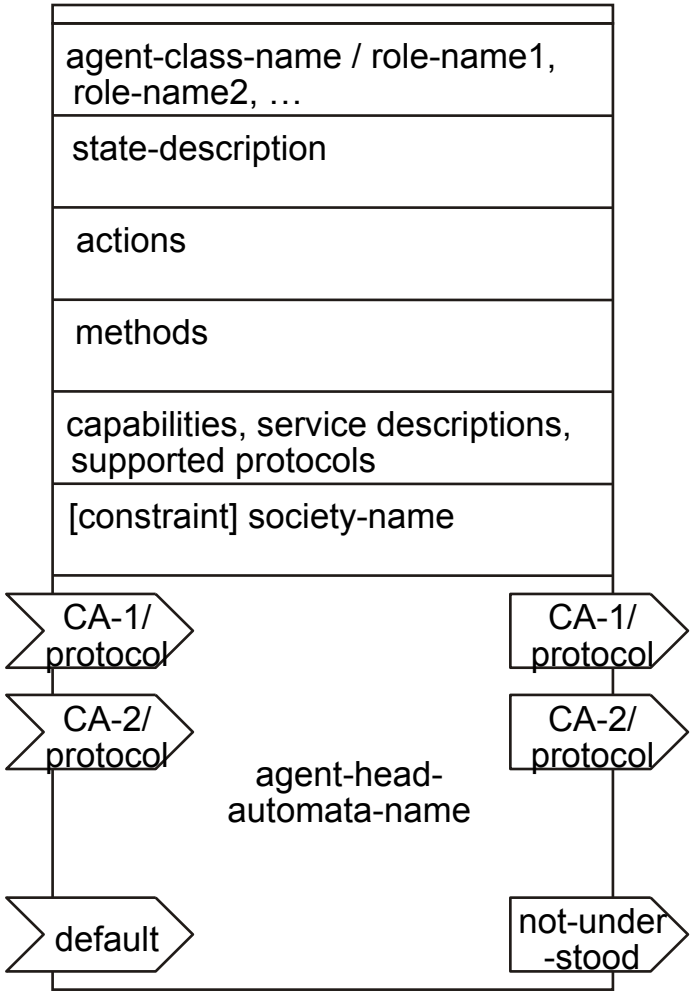
- Software engineering requires tools to
  - Represent software
    - E.g., interaction diagrams, E-R diagrams, …
  - Verify properties
    - E.g., Petri nets, formal methods, …
- Agent-oriented software engineering requires
  - Specific agent-oriented tools
    - E.g., UML is not suitable to model agent systems and their interactions

AOT Lab

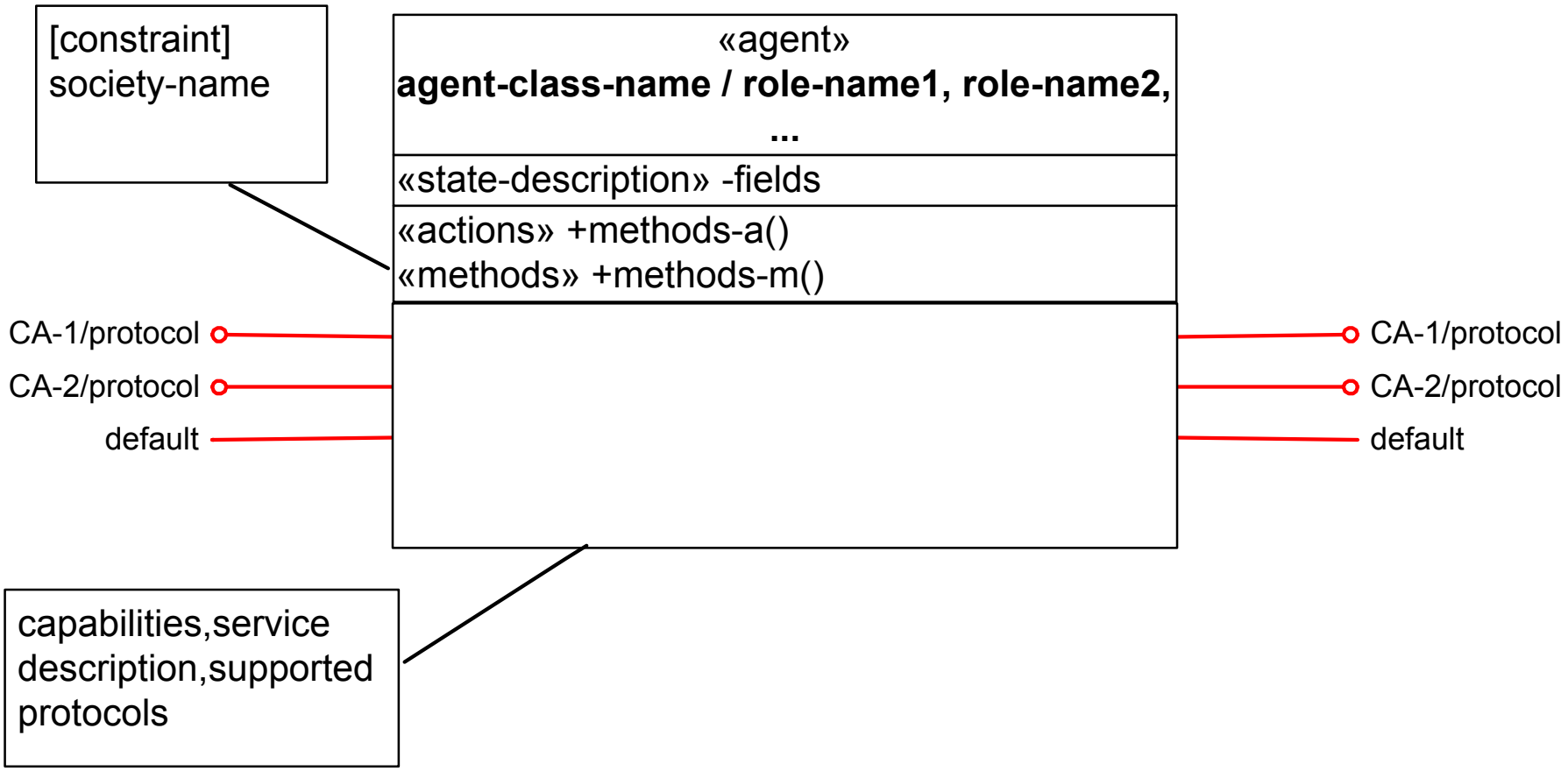
AUML

- Agent Unified Modeling Language is based on UML
  - Revised by FIPAModelingTC
- AUML is not a language yet, it is a proposal
- Extended with the following
  - Organized special agent class
  - New concept of role
  - New Agent Interaction Protocol Diagrams

AUML – Agent Notation

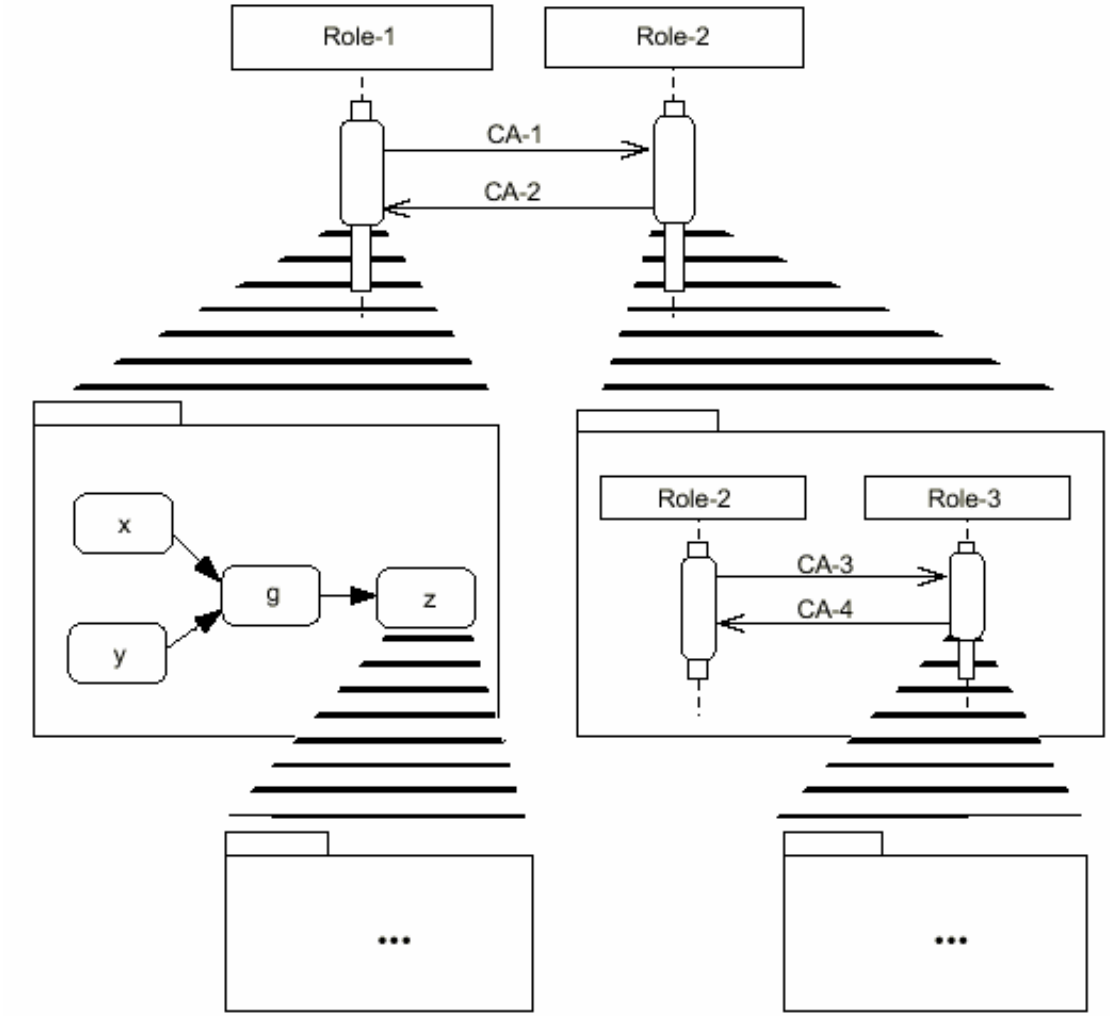


AUML – Agent Notation



AOT Lab

Agent Interaction Protocols

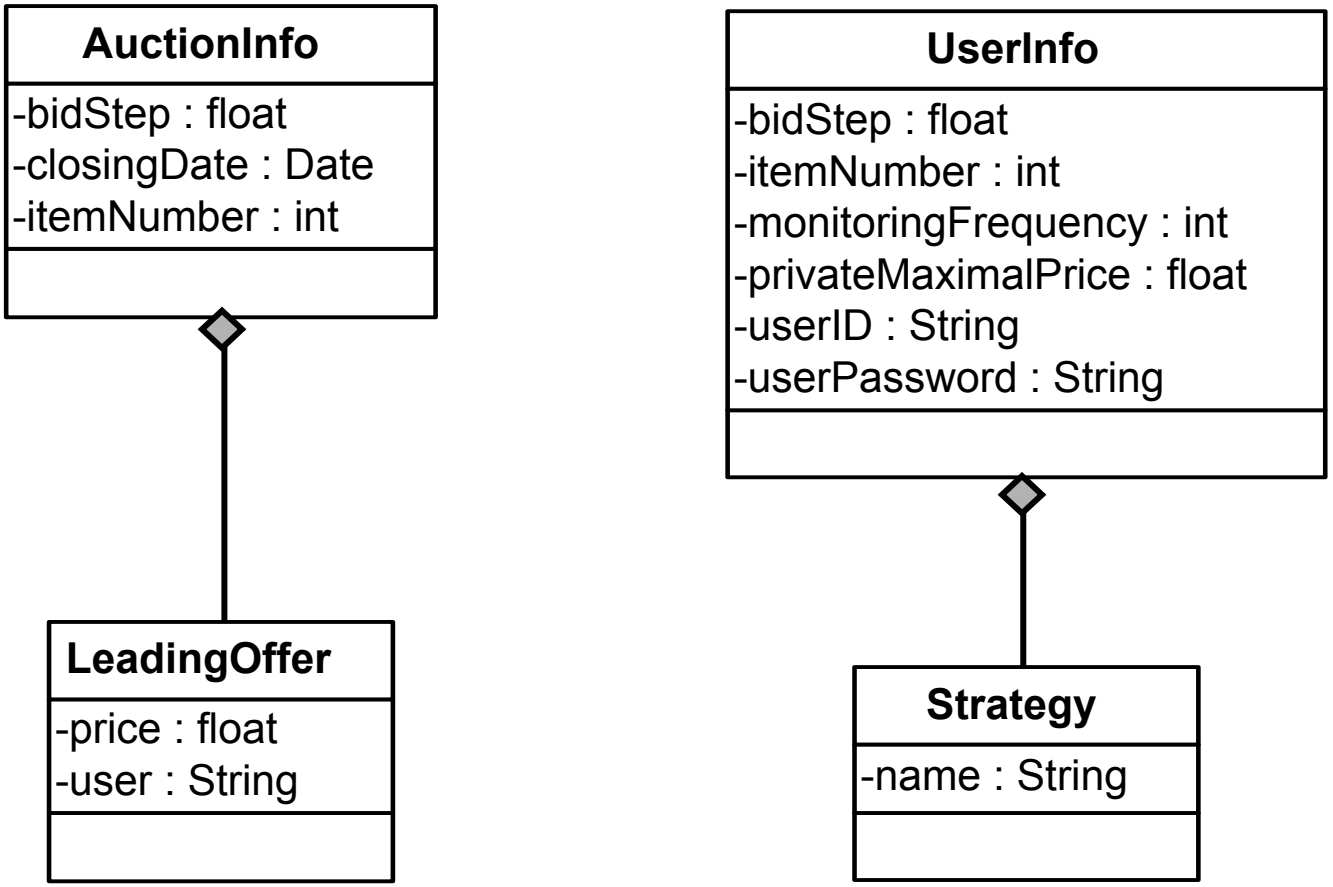


AOT Lab

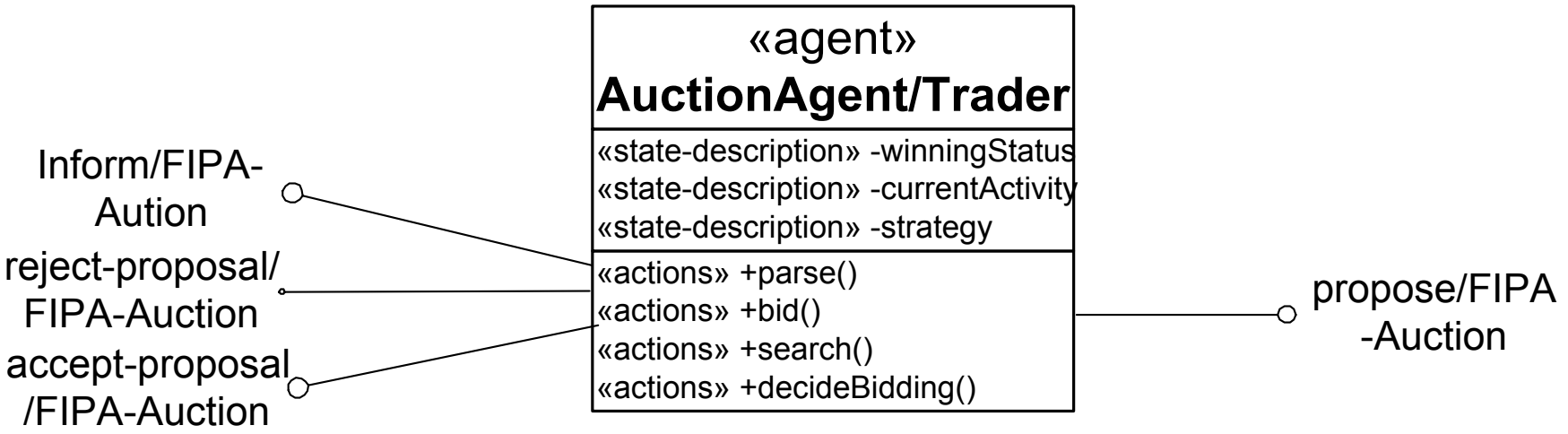
AUML – Extensions

- Agent Interaction Protocol
  - Layered protocol
  - Nested protocol
  - Interleaved protocol
- Extending the behavioral diagrams to be fitted to the concept of role

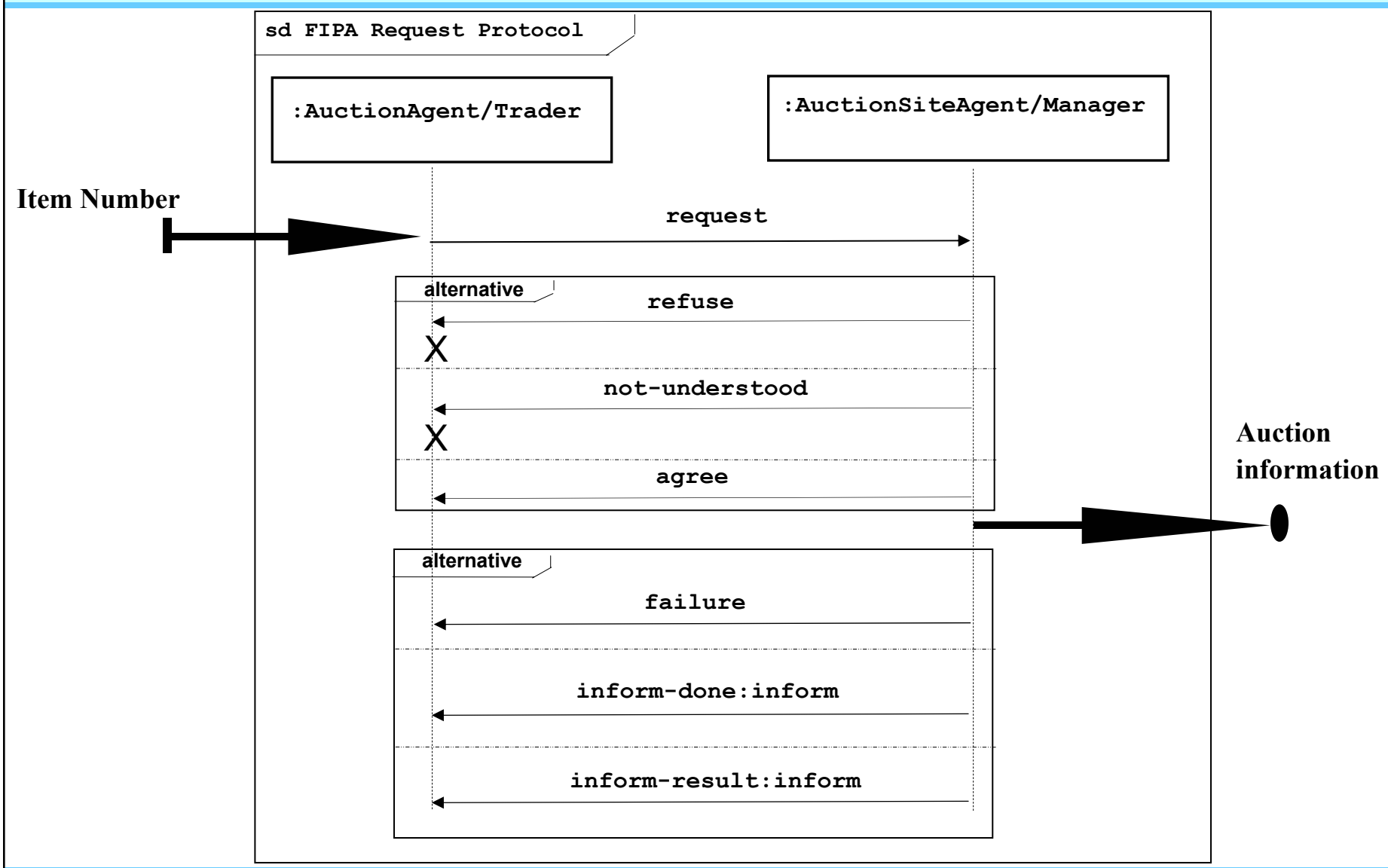
AUML – Knowledge Structure



AUML – System Structure



AUML – Agent Interaction Protocol



AOT Lab

Issues in Implementing Agents and Multiagent Systems

- How can we move from agent-based design to concrete agent code?
- Methodologies should abstract from
  - Internal agent architecture
  - Communication architecture
  - Implementation tools
- However, depending on tools the effort from design to implementation changes
  - It depends on how much abstractions are close to the abstractions of agent-oriented design

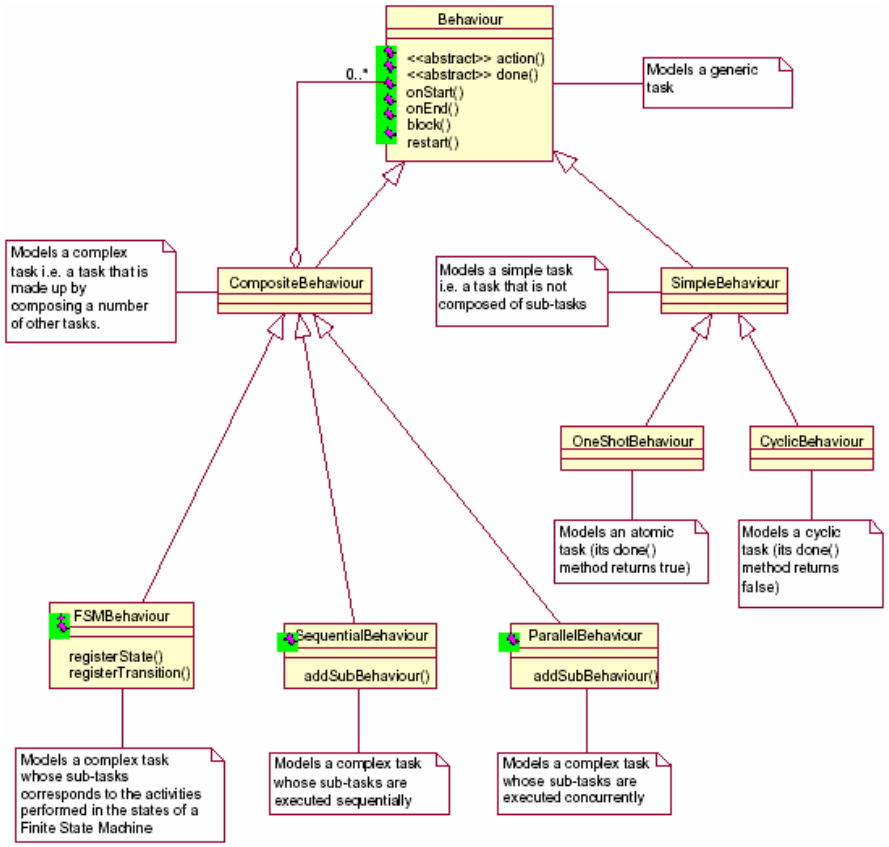
AOT Lab

Implementing Agents

- We have two categories of tools to implement agents
  - Object-oriented tools: are very much related to the object-oriented approach, e.g., frameworks
  - BDI toolkits: are based on the BDI model
- The choice of the tool to adopt is hard and there is no general answer
  - Performances
  - Maintenance
  - …and many other issues

Object-Oriented Tools: JADE

- JADE (Java Agent DEvelopment framework) implements a FIPA platform. It
  - Is distributed across the network in terms of containers
  - Provides management facilities, e.g., RMA
  - Provides advanced development facilities, e.g., Sniffer
- The agent architecture is based on behaviours that implement the tasks of the agent
  - One agent runs in one thread
  - Cooperative scheduling of prioritized behaviours
- Different type of behaviours, e.g.
  - FSM
  - Cyclic



AOT Lab

Example – CD Seller

```
import musicShopOntology.*;
import ecommerceOntology.*;
…other imports
public class CDSeller extends Agent {
  …declare private variables
  protected void setup() {
    …setup language and ontology
    …create initial knowledge base
    addBehaviour(new HandleRequestBehaviour(this));
  }

  class HandleRequestBehaviour
    extends CyclicBehaviour {
    public HandleRequestBehaviour(Agent a) {
      super(a);
    }
    public void action() {
      ACLMessage msg = receive(MessageTemplate.
        MatchPerformative(ACLMessage.REQUEST));
      try {
        ContentElement ce =
          manager.extractContent(msg);
        Sell           sell = null;
        AgentAction    toNotify = null;

        if (ce instanceof Sell) {
          sell = (Sell) ce; toNotify = sell
        } else { …unknown action }

        addBehaviour(new InformDoneBehaviour(
          myAgent, toNotify));
      } catch(Exception e) { e.printStackTrace(); }
    }
  }
```

```
class InformDoneBehaviour
  extends OneShotBehaviour {
  private AgentAction act;

  public InformDoneBehaviour(Agent a, AgentAction
    act) {
    super(a); this.act = act;
  }

  public void action() {
    try {
      ACLMessage msg = new
        ACLMessage(ACLMessage.INFORM);
      AID receiver = new AID(receiver, false);

      msg.setSender(getAID());
      msg.addReceiver(receiver);
      msg.setLanguage(codec.getName());
      msg.setOntology(ontology.getName();
      Done d = new Done(act);

      manager.fillContent(msg, d);
      send(msg);
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

AOT Lab

BDI Toolkits: ParADE

- ParADE (Parma Agent Development Environment) is a toolkit for the development of BDI FIPA agents

- Agent level
  - Agents are atomic components
  - UML is used to build models of single agents and of the multiagent system
- Object level, exploits the generated code
  - Each agent is an object-oriented system
  - ParADE provides is a framework on top of JADE

AOT Lab

ParADE – Characteristics

- ParADE agents
  - Integrate reactive and goal-directed behaviours to balance autonomy and efficiency
  - Exploit the FIPA ACL with a minimalist semantics
- ParADE generates Java code from
  - Ontology diagram, models the part of the ontology that support the communication
  - Architecture diagram, defines the architecture and the interaction protocols

AOT Lab

Example – Song Seller

```
…ParADE imports
public class Shop extends ShopAgent {
  protected void init() {
    …set the agent model
    Agent anyAgent = new AgentVariable("y");
    Song  anySong  = new SongVariable("w");

    // Plans to achieve intentions
    // If 'anyAgent' requests for a song and the song is available, then execute 'ActionBody'
    plan(available(me, anySong),              // precondition
         done(sell(anyAgent, anySong)),       // intention to achieve
         new ActionBody() {                   // the action to perform to achieve the intention
           public void body(Goal g) {
             Done done = (Done)g;
             Sell sell = (Sell)done.getAction();

             sell.perform();

             forget(intend(sell.getAgent(), done));
             achieved(done);
           }
         });

         Song OneHeadlight = new ConcreteSong("One Headlight", 1000);
         believe(available(me, OneHeadlight));

         ConcreteAgent receiver = new ConcreteAgent(receiver);
         schedule(inform(receiver, available(me, OneHeadlight)));
  }
}
```

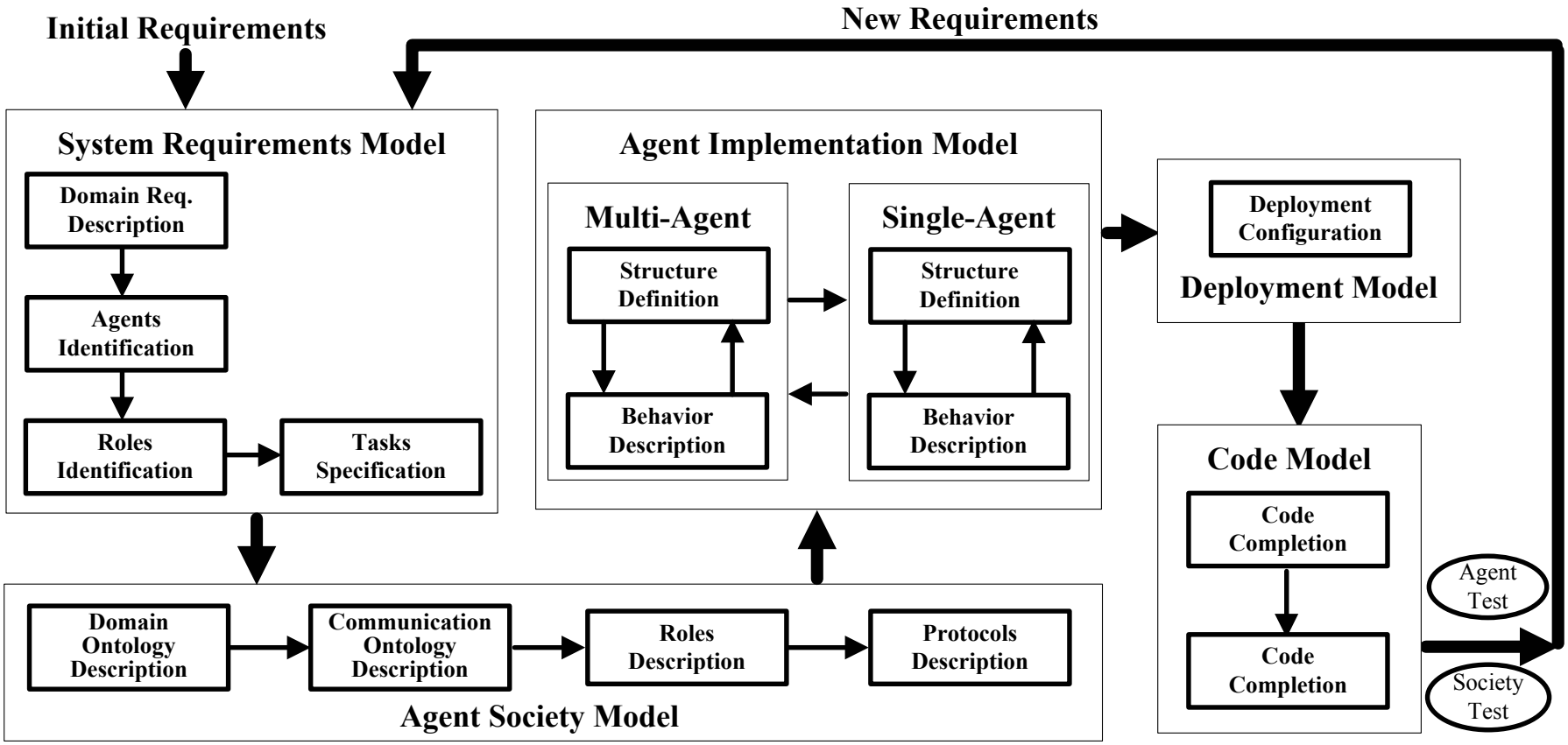
AOT Lab

CASE Tools: PASSI Toolkit

- PASSI is a step-by-step requirements-to-code method for developing multiagent
  - integrates design models and philosophies from both object-oriented software engineering and MAS using UML notation
  - The modeling language is an extension of UML
- PASSI is conceived to be supported by PTK, an agent-oriented CASE tool
  - The functionalities of PTK include:
    - Automatic (total or partial) compilation of some diagrams
    - Automatic support to the execution of recurrent operations
    - Check of design consistency
    - Automatic compilation of reports and design documents
    - **Access to a database of patterns**
    - Generation of code and Reverse Engineering

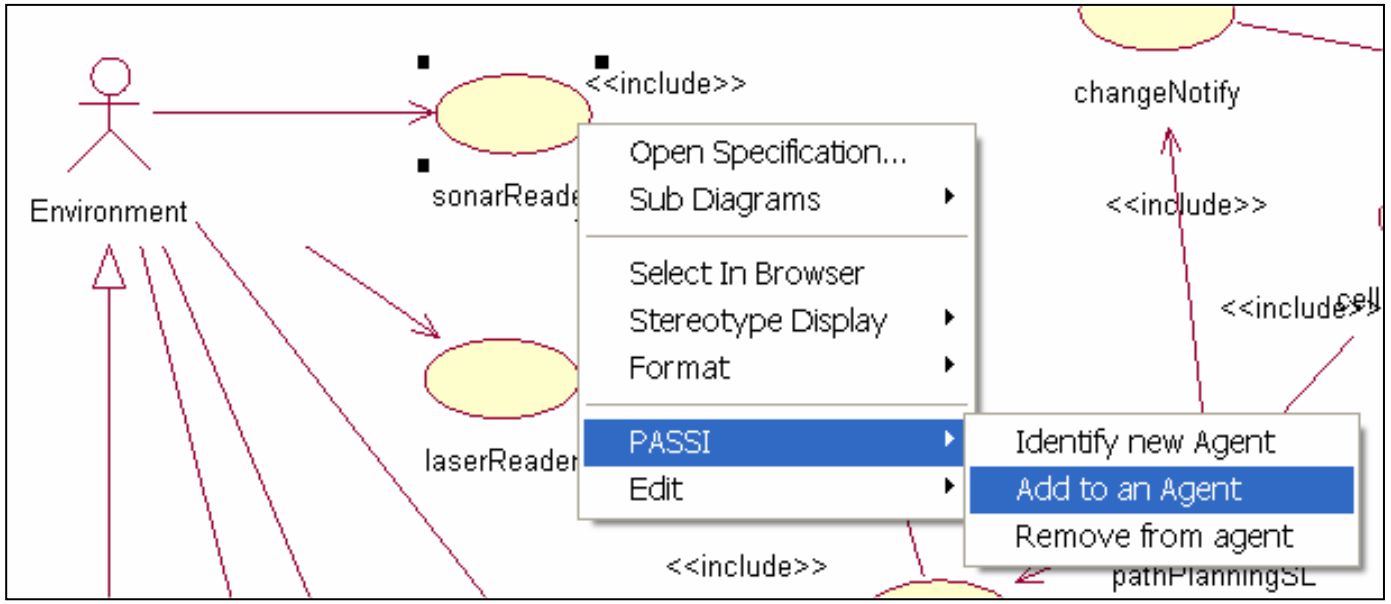
PASSI

(Process for Agent Societies Specification and Implementation)



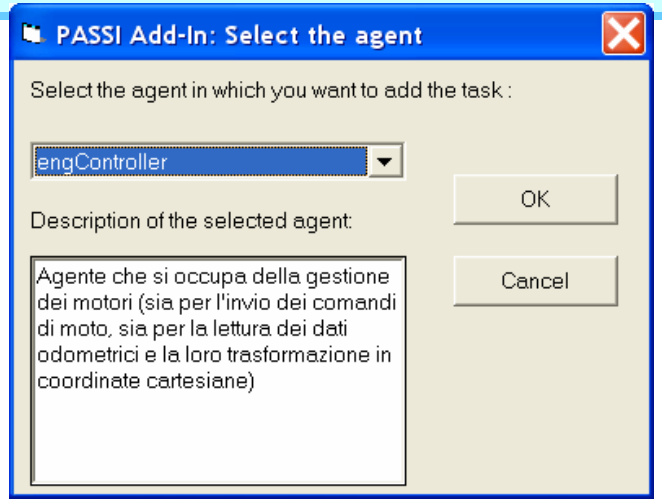
PTK - Sample

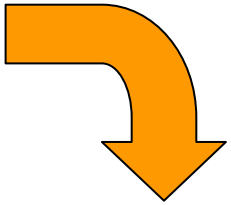
- Agent Identification Diagram automatically composed by the tool
  - The designer creates new agents and select their use cases operating in the Domain Description diagram

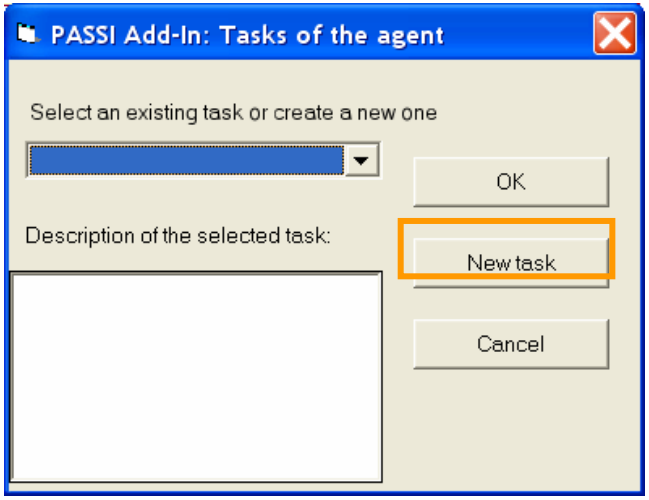


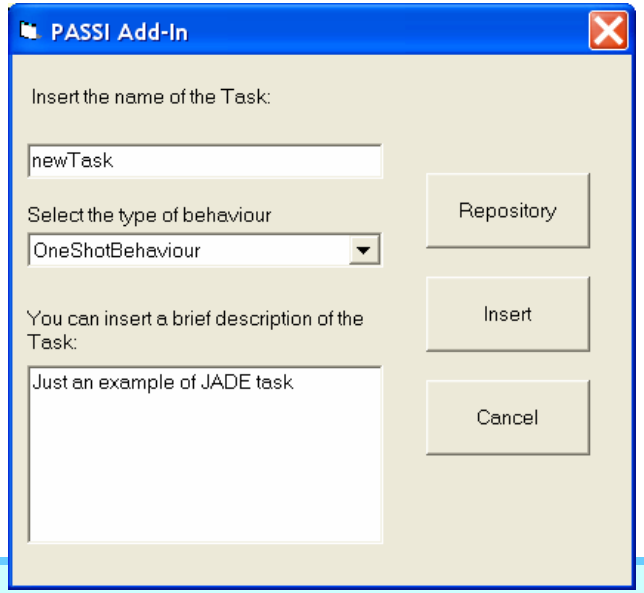
AOT Lab

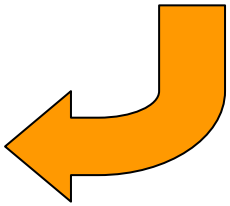
PTK - Sample





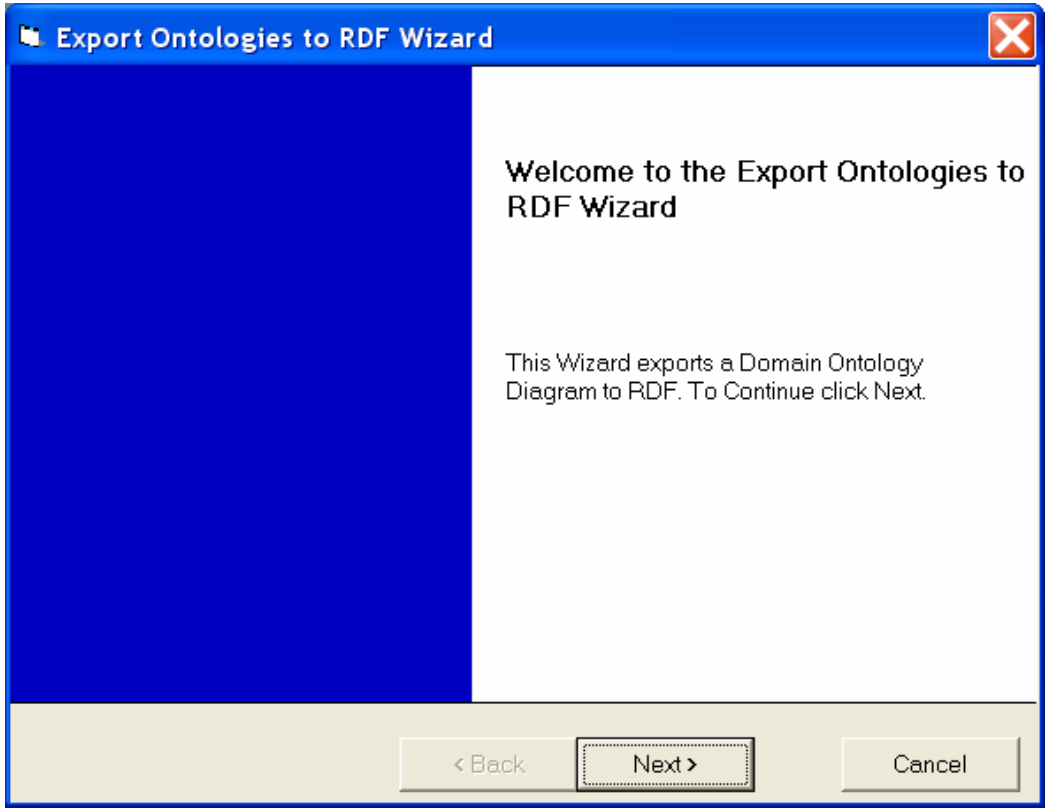






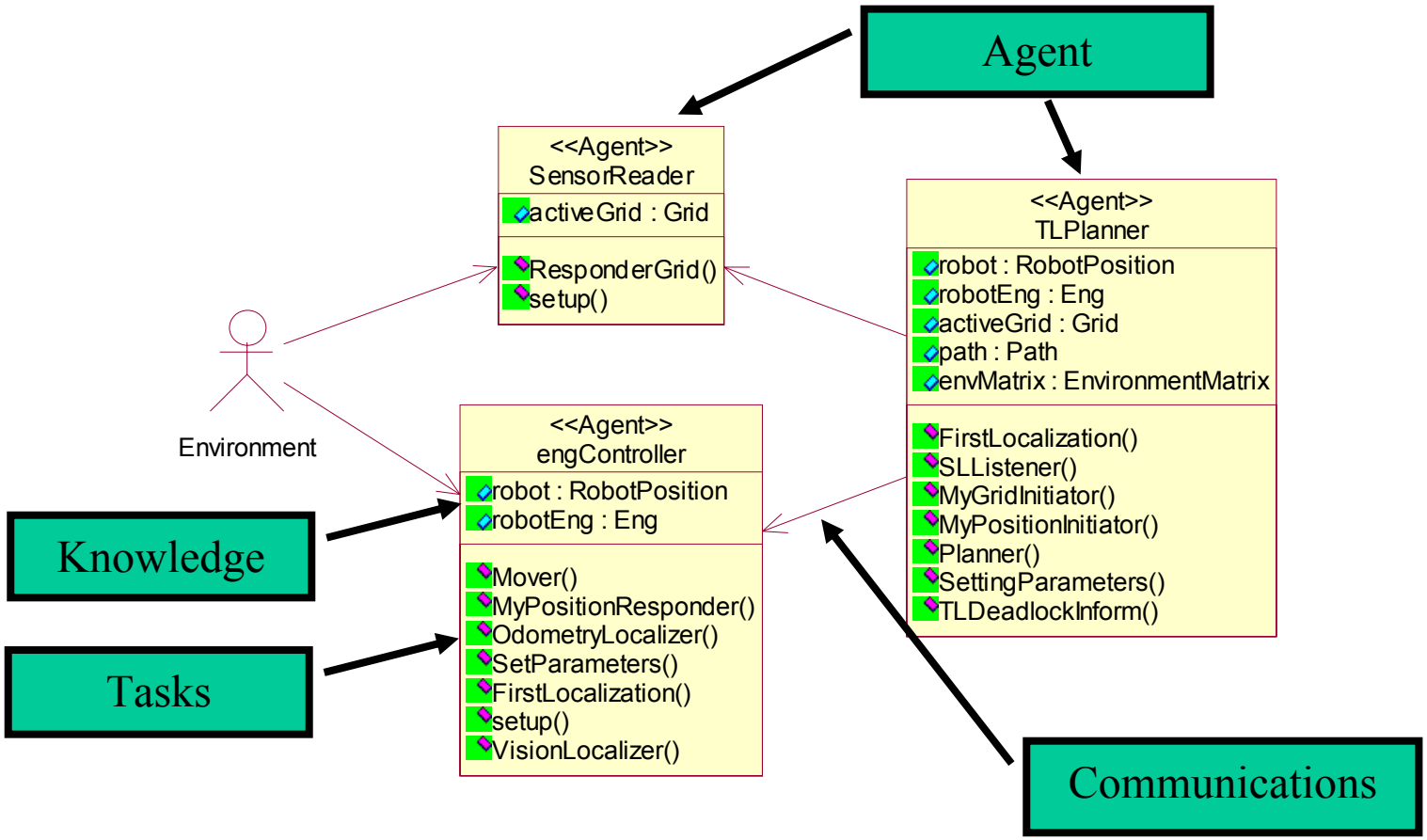
PTK - Sample

- Starting from the Domain Ontology Description diagram, PTK exports the RDF description of the ontology



PTK - Sample

- Multiagent Structure Definition Diagram automatically compiled by the tool



AOT Lab

Implementing Multiagent Systems

- Inter-agent implementation aspects are orthogonal to intra-agent ones
  - Given a set of agents
    - With internal architecture
    - With specified interaction patterns
  - How can we glue them together?
    - Letting agents know each other
  - How to enable interactions?
    - Promoting spontaneous interoperability
  - How to rule interactions?
    - Preventing malicious or self-interested behaviours?

AOT Lab

Multiagent Infrastructures

- Enabling and ruling interactions is mostly a matter of the infrastructure
- The middleware supporting communication and coordination activities
  - Not simply a passive layer
  - But a layer of communication and coordination services
    - Actively supporting the execution of interaction protocols
    - Providing for helping agents move in unknown worlds
    - Providing for proactively controlling, and possibly influencing interactions

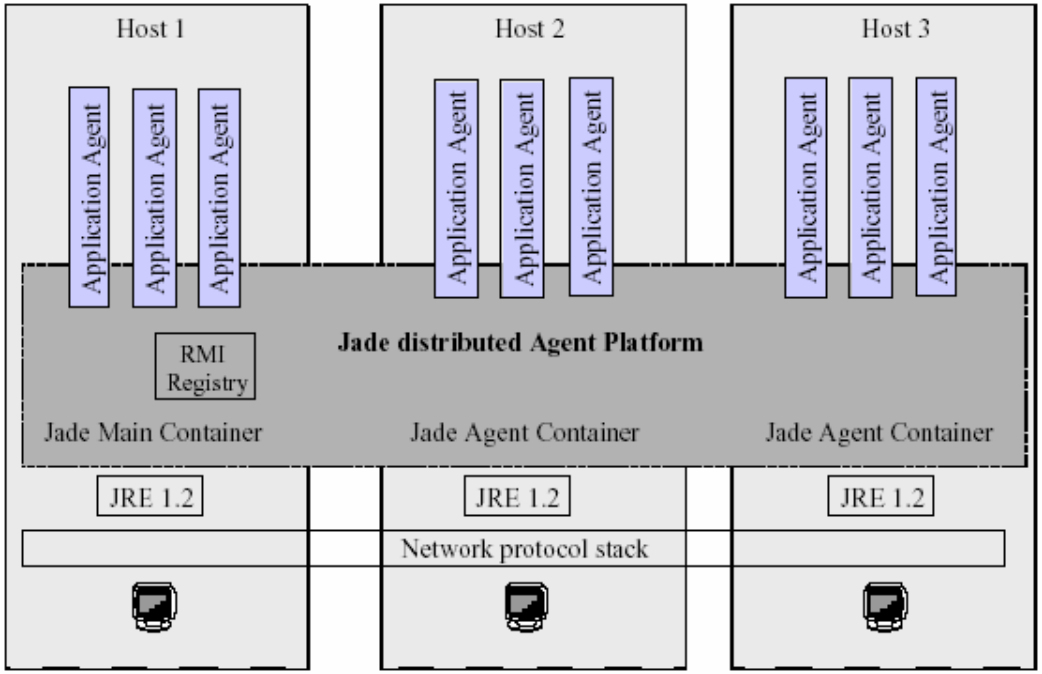
AOT Lab

Communication Infrastructure

- Agent in a multiagent system interact with each other, requiring
  - Finding other agents
    - Directory services in the infrastructure keep track of which agents are around, and what are their characteristics (e.g., services provided)
  - Re-routing message
    - Facilitator agents (parts of the infrastructure) can receive messages to be delivered to agents with specific characteristics, and re-route them
  - Control on ACL protocols
    - The execution of a single protocol can be controlled in terms of a finite state machine

AOT Lab

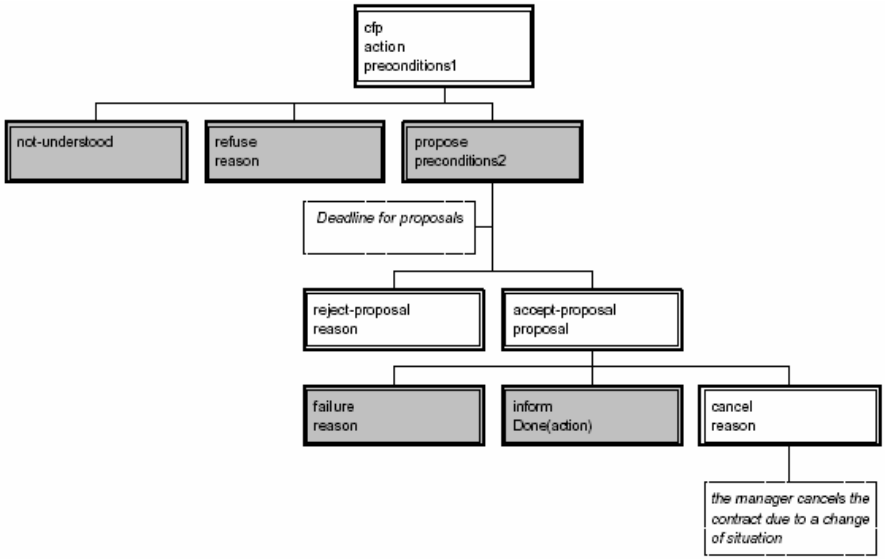
Example of Communication Infrastructures: JADE (1)



- Implements a FIPA platform with all necessary services, e.g., DF
- JADE
  - Is distributed across the network in terms of containers
  - Provides management facilities, e.g., RMA
  - Provides advanced development facilities, e.g., Sniffer

Example of Communication Infrastructures: JADE (2)

- Interaction protocols are the FIPA way to manage interactions
- JADE provides support for FIPA generic interaction protocols, e.g.
  - FIPA Contract net
  - FIPA English and Dutch auctions
- JADE implements interaction protocols as FSM behaviours



AOT Lab

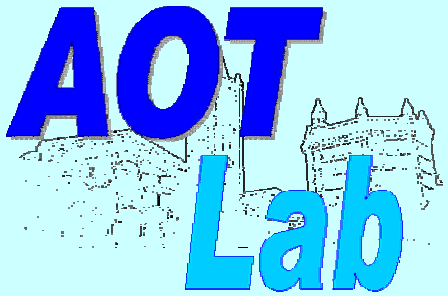
Features and Limitations of Communication Infrastructures

- There is not application intelligence in the infrastructure
  - The service provided are
    - Of a very general-purpose nature
    - Not re-configurable to meet the need of specific applications
- There is no global orchestration
  - The only proactive control is on individual protocols
    - There is no way of controlling and influencing the global behaviour of a multiagent system
    - How to control self-interested behaviour, unpredictable dynamics, programming errors?

AOT Lab

Software Engineering with Communication Infrastructures

- All application problems are to be identified and designed in terms of
  - Internal agent behaviours and inter-agent interaction protocols
  - These include, from the intra-agent engineering viewpoint
    - Controlling the global interactions
    - Controlling self-interested behaviours
- Advantages
  - All in the system is an agents
  - The engineering of the system does not imply the engineering of the infrastructure
  - A standard has already emerged (FIPA)
- Drawbacks
  - The design is hardly re-tunable
  - Global problems spread into internal agents' code



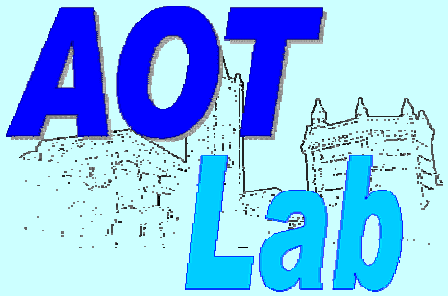
AOT Lab
Dipartimento di Ingegneria
dell'Informazione
Università degli Studi di Parma

Thank You!

Questions?

Comments?

Ideas?



AOT Lab
Dipartimento di Ingegneria
dell'Informazione
Università degli Studi di Parma

Addendum

Selected References

AOT Lab

Selected References

- Introductory to Agents and Multiagent Systems
  - A. Newell, “The Knowledge Level”, Artificial Intelligence, 18(1):87-127, 1982.
  - P. Wegner, “Why Interaction is More Powerful than Algorithms”, Communications of the ACM, 40(5):80–91, 1997.
  - M. Wooldridge, “Reasoning About Rational Agents”, MIT Press, 2000.
  - M. Wooldridge, N. Jennings, “Intelligent Agents: Theory and Practice”, The Knowledge Engineering Review, Vol. 10, No. 2, 1999.
  - D. Chess, C. Harrison, A. Kershenbaum, “Mobile Agents: are They a Good Idea?”, Mobile Object Systems, Lecture Notes in Computer Science, No. 1222, Springer-Verlag (D), pp. 25-45, February 1997.
  - V. Parunak, “Go to the Ant: Engineering Principles from Natural Agent Systems”, Annals of Operations Research, 75:69-101, 1997.
  - N. R. Jennings, "An Agent-Based Approach for Building Complex Software System", Communications of the ACM, 44(4):35:41, 2001.

AOT Lab

Selected References

- Introductory to AOSE
  - N.R. Jennings, “On Agent-Based Software Engineering”, Artificial Intelligence, 117:227-296, 2000.
  - N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, B. Odgers, “Autonomous Agents for Business Process Management”, Int. Journal of Applied AI, Vol. 14 (2), pp. 145-189, 2000.
  - M. J. Wooldridge and N. R. Jennings, "Software Engineering with Agents: Pitfalls and Pratfalls", IEEE Internet Computing, Vol.3, No. 3, May-June 1999.
  - Y. Shoham, “An Overview of Agent-Oriented Programming”, in J. M. Bradshaw, editor, Software Agents, pages 271–290. AAAI Press / The MIT Press, 1997.
  - K. Siau and M. Rossi, “Evaluation of Information Modeling Methods – A Review”, Proceedinga 31st Annual Hawaii International Conference on System Sciences, pp. 314-322, 1998.
  - F. Zambonelli, N. Jennings, M. Wooldridge, “Organizational Abstractions for the Analysis and Design”, 1st International Workshop on Agent-oriented Software Engineering, LNAI No. 1957, Springer, 2001.

AOT Lab

Selected References

- Surveys on Methodologies
  - C. Iglesias, M. Garijo, J. C. Gonzales, “A Survey of Agent-oriented Methodologies”, Intelligent Agents V, LNAI No. 1555, 1999.
  - M. Wooldridge, P. Ciancarini, “Agent-Oriented Software Engineering”, in Agent-Oriented Software Engineering, LNCS No. 1957, 2001.
  - O. Shehory and A. Sturm, “Evaluation of Modeling Techniques for Agent-Based Systems”, Proceedings of The Fifth International Conference on Autonomous Agents, pp. 624-631, 2001.
- The GAIA Methodology
  - M. Wooldridge, N. Jennings, D. Kinny,”The Gaia Methodology for Agent-Oriented Analysis and Design”, Journal of Autonomous Agents and Multi-agent Systems, 3(3), 2000.
  - F. Zambonelli, N. Jennings, M. Wooldridge, “Organizational Rules as an Abstraction for the Analysis and Design of Multiagent Systems”, Journal of Software and Knowledge Engineering, 11(3), 2001.
  - F. Zambonelli, N. Jennings, M. Wooldridge, “Developing Multiagent Systems: the Gaia Methodology”, ACM Transactions on Software Engineering and Methodology, 12(3):417-470, July 2003

AOT Lab

Selected References

- Other Relevant Methodologies
  - P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, “A Knowledge Level Software Engineering Methodology for Agent Oriented Programming”, Proceedings of the 5th International Conference on Autonomous Agents, Montreal (CA), June 2001.
  - G. Caire, F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, J. Pavon, P. Kearney, J. Stark, P. Massonet, “Agent Oriented Analysis using MESSAGE/UML”, 2nd International Workshop on Agent-Oriented Software Engineering, LNCS No. 2222, Springer-Verlag, pp. 119-135, 2001.
  - M. Cossentino, C. Potts, “A CASE tool supported methodology for the design of multi-agent systems”. In Proc. SERP'02 - June 24 - 27, 2002 - Las Vegas (NV), USA
  - M. Cossentino, “Different Perspectives in Designing Multi-Agent System”, AgeS’02 (Agent Technology and Software Engineering) Workshop at NodE’02, Erfurt, Germany, October 2002
  - S. A. DeLoach, M. F. Wood, Cl. H. Sparkman, “Multiagent Systems Engineering”, The International Journal of Software Engineering and Knowledge Engineering, Vol. 11 (3), pp. 231-258, 2001.

AOT Lab

Selected References

- AUML
  - B. Bauer, J.P. Muller, J. Odell, “Agent UML: A Formalism for Specifying Multiagent Software Systems”, The International Journal of Software Engineering and Knowledge Engineering, Vol. 11 (3), pp. 207-230, 2001.
  - B. Bauer, “UML Class Diagrams: Revisited in the Context of Agent-Based Systems”, Proceedings of Agent-Oriented Software Engineering (AOSE),pp.1-8, 2001.
- FIPA
  - FIPA Modeling Technical Committee – Home Page – Available at http://www.fipa.org/activities/modeling.html
  - FIPA Methodology Technical Committee – Home Page – Available at http://www.fipa.org/activities/methodology.html

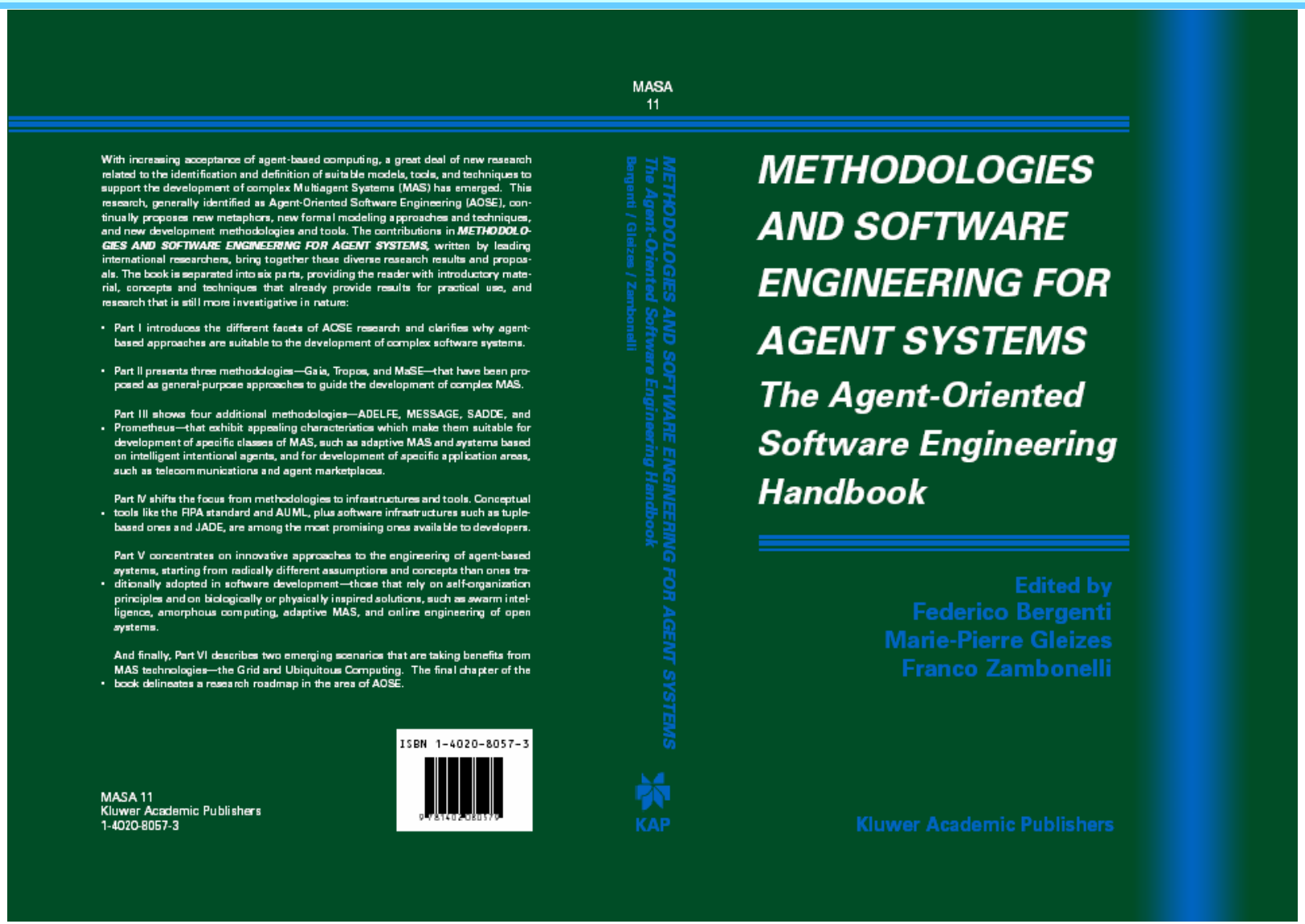
AOT Lab

Selected References

- Multiagent Systems Infrastructures
  - F. Bellifemine, A. Poggi, G. Rimassa, “Developing Multi-Agent Systems with a FIPA-Compliant Agent Framework”, Software Practice and Experience, 31:103–128, 2001.
  - S. Poslad, P. Buckle, and R. Hadingham, “The FIPA-OS Agent Platform: Open Source for Open Standard”, available at http://fipa-os.sourceforge.net.
  - P. Busetta, R. Rönnquist, A. Hodgson, A. Lucas, “JACK Intelligent Agents: Components for Intelligent Agents in Java”, Agentlink News Letter, 1999.
  - P. Ciancarini, A. Omicini, F. Zambonelli, “Coordination Technologies for Internet Agents”, Nordic Journal of Computing, 6(1), 2000.

AOT Lab

Selected References



MASA
11

With increasing acceptance of agent-based computing, a great deal of new research related to the identification and definition of suitable models, tools, and techniques to support the development of complex Multiagent Systems (MAS) has emerged. This research, generally identified as Agent-Oriented Software Engineering (AOSE), continually proposes new metaphors, new formal modeling approaches and techniques, and new development methodologies and tools. The contributions in ***METHODOLOGIES AND SOFTWARE ENGINEERING FOR AGENT SYSTEMS***, written by leading international researchers, bring together these diverse research results and proposals. The book is separated into six parts, providing the reader with introductory material, concepts and techniques that already provide results for practical use, and research that is still more investigative in nature:

- Part I introduces the different facets of AOSE research and clarifies why agent-based approaches are suitable to the development of complex software systems.
- Part II presents three methodologies—Gaia, Tropos, and MaSE—that have been proposed as general-purpose approaches to guide the development of complex MAS.
- Part III shows four additional methodologies—ADELFE, MESSAGE, SADDE, and Prometheus—that exhibit appealing characteristics which make them suitable for development of specific classes of MAS, such as adaptive MAS and systems based on intelligent intentional agents, and for development of specific application areas, such as telecommunications and agent marketplaces.
- Part IV shifts the focus from methodologies to infrastructures and tools. Conceptual tools like the FIPA standard and AUML, plus software infrastructures such as tuple-based ones and JADE, are among the most promising ones available to developers.
- Part V concentrates on innovative approaches to the engineering of agent-based systems, starting from radically different assumptions and concepts than ones traditionally adopted in software development—those that rely on self-organization principles and on biologically or physically inspired solutions, such as swarm intelligence, amorphous computing, adaptive MAS, and online engineering of open systems.
- And finally, Part VI describes two emerging scenarios that are taking benefits from MAS technologies—the Grid and Ubiquitous Computing. The final chapter of the book delineates a research roadmap in the area of AOSE.

ISBN 1-4020-8057-3

MASA 11
Kluwer Academic Publishers
1-4020-8057-3

METHODOLOGIES AND SOFTWARE ENGINEERING FOR AGENT SYSTEMS
The Agent-Oriented Software Engineering Handbook
Bergenti / Gleizes / Zambonelli

KAP

METHODOLOGIES AND SOFTWARE ENGINEERING FOR AGENT SYSTEMS
The Agent-Oriented Software Engineering Handbook

Edited by
Federico Bergenti
Marie-Pierre Gleizes
Franco Zambonelli

Kluwer Academic Publishers