Agent-Oriented Software Engineering

Franco Zambonelli

February 2005

Outline

- Part 1: What is Agent-Oriented Software Engineering (AOSE)
  - Why it is important
  - Key concepts.
- Part 2: Agent-methodologies
  - Key Concepts
  - The Gaia Methodology
  - Case Study
- Part 3: Implementing agents
  - Intra-agent vs. inter-agent issues
  - Multiagent infrastructures
- Part 4: Conclusions and Open Research directions.

Part 1

- What is Agent-Oriented Software Engineering

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;
  - The software:
    - Well-defined quality level (e.g., % of errors and performances);
    - Enabling reuse and maintenance.
- Requires:
  - Methodologies → Abstractions, and tools.

Software Engineering Abstractions

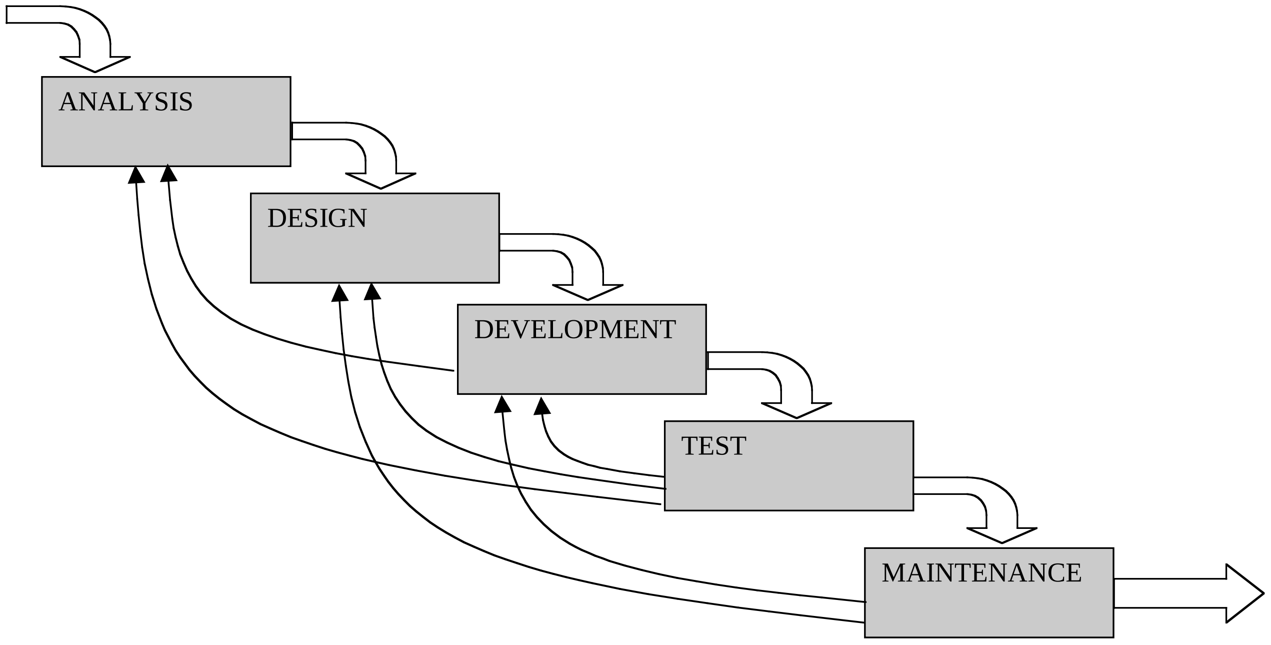
- Software deals with “abstract” entities, having a real-world counterpart:
    - Numbers, dates, names, persons, documents ...
- In what terms should we model them in software?
    - Data, functions, objects, agents ...
    - I.e., what are the **ABSTRACTIONS** that we have to use to model software?
- May depend on the available technologies!
    - Use OO abstractions for OO programming envs.;
    - Not necessarily: use OO abstractions because they are better, even for COBOL programming envs.

Methodologies

- A methodology for software development:
  - Is intended to give discipline to software development.
  - Defines the abstractions to use to model software:
    - Data-oriented methodologies, object-oriented ones …
    - Define the MINDSET of the methodology.
  - Disciplines the software process:
    - What to produce and when;
    - Which artifacts to produce.

The Classical “Cascade” Process

- The phases of software development:
  - Independent of programming paradigm;
  - Methodologies are typically organized around this classical process.
    - Inputs, outputs, internal activities of “phases”



Tools

- Notation tools:
  - To represent the outcomes of the software development phases:
    - Diagrams, equations, figures …
- Formal models:
  - To prove properties of software prior to development
    - Lambda and pi calculus, UNITY, Petri-nets, Z ….
- CASE (Computer Aided Software Engineering) tools:
  - Based on notations and models, to facilitate activities:
    - Simulators, rapid prototyping, code generators.

Example: Object-oriented Software Engineering (OOSE)

- Abstractions:
  - Objects, classes, inheritance, services.
- Methodologies:
  - Object-oriented analysis and design, RUP, OPEN, etc..;
  - Centered around the object-oriented abstractions.
- Tools (Modeling Techniques):
  - UML (standard), E-R, class lattices, finite state automata, visual languages ...

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!

What are Agents?

- There has been some debate
  - On what an agent is, and what could be appropriately called an agent
- Two main viewpoints (centered on different perspectives on autonomy):
  - The (strong) Artificial Intelligence viewpoint:
    - An agent must be, proactive, intelligent, and it must converse 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

What are Multiagent Systems?

- Again....
  - The (strong) artificial intelligence viewpoint
    - A multiagent system is a society of individuals (AI software agents) 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., the agents) interacting with each other in the context of a specific application viewpoint....

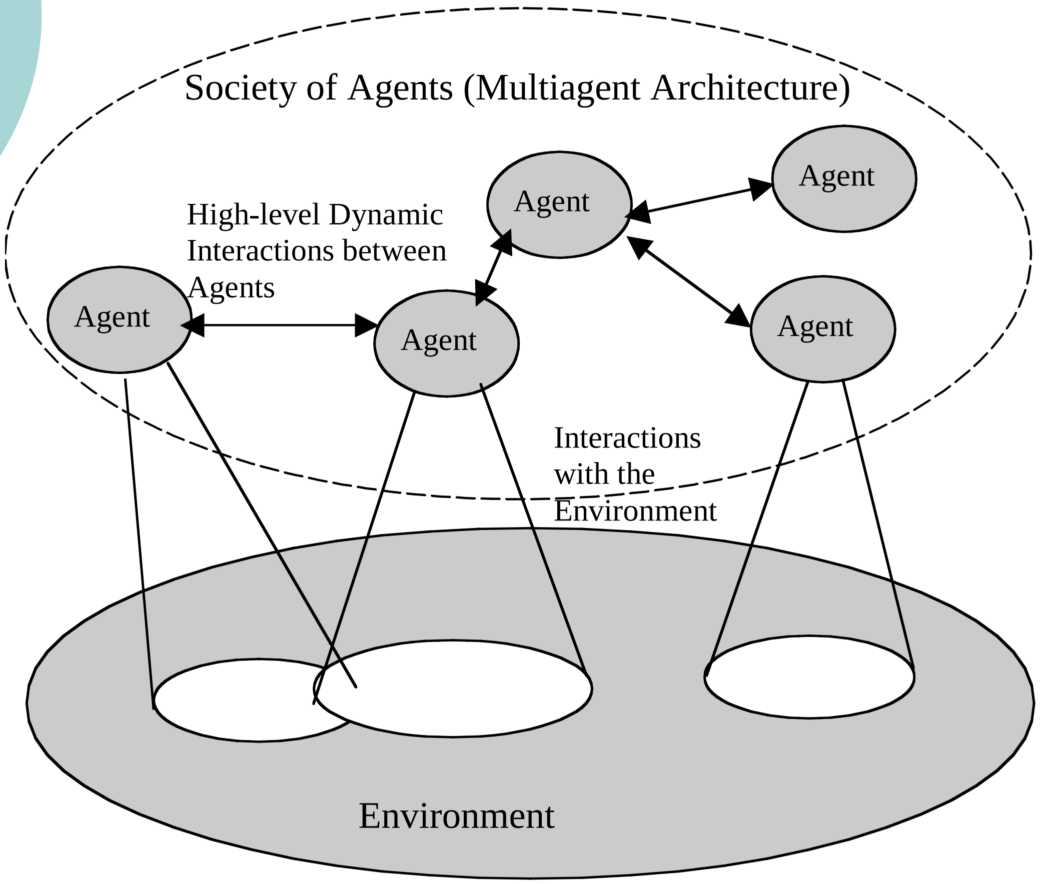
The SE Viewpoint on Agent-oriented Computing

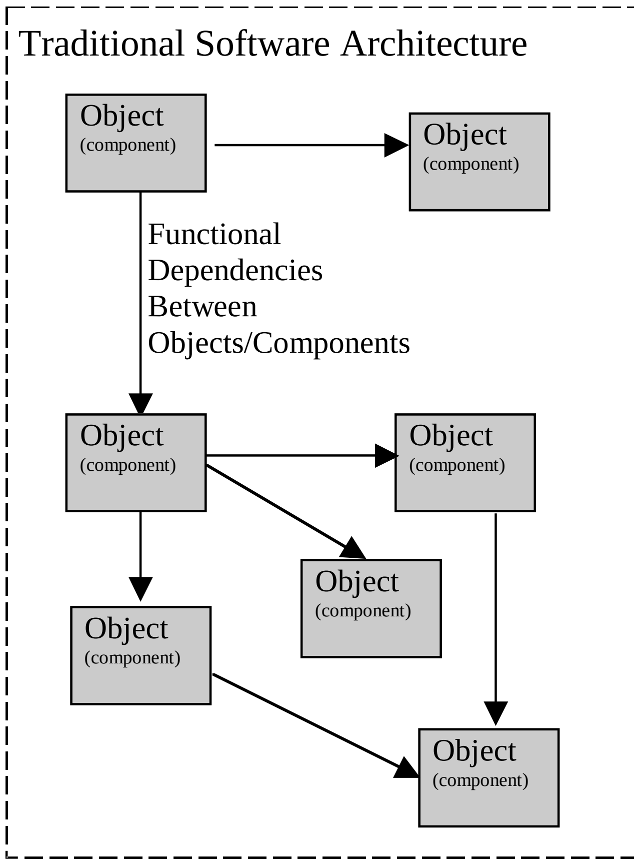
- We commit to it because:
  - It focuses on the characteristics of agents that have impact on software development
    - Concurrency, interaction, multiple loci of control
    - Intelligence can be seen as a peculiar form of control independence; conversations as a peculiar form of interaction
  - It 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
- Let's better characterize the SE perspective on agents...

SE Implications of Agent Characteristics

- Autonomy
  - Control encapsulation as a dimension of modularity
  - Conceptually simpler to tackle than a single (or multiple inter-dependent) locus of control
- Situatedness
  - Clear separation of concerns between:
    - the active computational parts of the system (the agents)
    - the resources of the environment
- Sociality
  - Not a single characterising protocol of interaction (e.g., client-server)
  - Interaction protocols as an additional SE dimension
- Openness
  - Controlling self-interested agents, malicious behaviors, and badly programmed agents
  - Dynamic re-organization of software architecture
- Mobility and Locality
  - Additional dimension of autonomous behavior
  - Improve locality in interactions

MAS vs. OOSE Characterisation


Society of Agents (Multiagent Architecture)
Agent
Agent
Agent
Agent
Agent
High-level Dynamic Interactions between Agents
Interactions with the Environment
Environment



Agent-Oriented Abstractions

- The development of a multiagent system should fruitfully exploit **abstractions** coherent with the above characterization:
  - **Agents,** autonomous entities, independent loci of control, situated in an environment, interacting with each other
  - **Environment**, the world of resources agents perceive
  - **Interaction protocols**, as the acts of interactions between agents
- In addition, there may be the need of abstracting:
  - The **local context** where an agent lives (e.g., a sub-organization of agents) to handle mobility & opennes
- Such abstractions translates into concrete entities of the software system

Agent-Oriented Methodologies

- There is need for SE methodologies
  - Centered around specific agent-oriented abstractions
    - E.g., Agents, environments, interaction protocols
  - 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, beliefs, desires 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

Agent-Oriented Tools

- SE requires tools to
  - represent software
    - E.g., interaction diagrams, E-R diagrams, etc.
  - verify properties
    - E.g., petri nets, formal notations, etc.
- AOSE requires
  - Specific agent-oriented tools
    - E.g., UML per se is not suitable to model agent systems and their interactions (object-oriented abstractions not agent-oriented ones)

Why Agents and Multiagent Systems?

- Other lectures may have already outlined the advantages of (intelligent) agents and of multiagent systems, and their possible applications
  - Autonomy for delegation (do work on our behalf)
  - Monitor our environments
  - More efficient interaction and resource management
- Here, we state that
  - **Agent-based computing, and the abstractions it uses, represent a new and general-purpose software engineering paradigm!**

There is much more to agent-oriented software engineering

- AOSE is not only for “agent systems.”
  - Most of today’s software systems have characteristics that are very similar to those of agents and multiagent systems
  - The agent abstractions, the methodologies, and the tools of AOSE suit such software systems
- AOSE is suitable for a wide class of scenarios and applications!
  - Agents’ “artificial Intelligence” features may be important but are not central
- But of course...
  - AOSE may sometimes be too “high-level” for simple complex systems...

Agents and Multiagent Systems are (Virtually) Everywhere!

- Examples of components that can be modelled (and **observed**) in terms of agents:
  - Autonomous network processes;
  - Computing-based sensors;
  - PDAs;
  - Robots.
- Example of software systems that can be modelled as multiagent systems:
  - Internet applications;
  - P2P systems;
  - Sensor networks;
  - Pervasive computing systems.

Summarizing

- A software engineering paradigm defines:
  - The mindset, the set of abstractions to be used in software development and, consequently,
  - Methodologies and tools
  - The range of applicability
- Agent-oriented software engineering defines
  - Abstractions of agents, environment, interaction protocols, context
  - Of course, also specific methodologies and tools (in the following of the tutorial)
  - Appears to be applicable to a very wide rage of distributed computing applications….

Part 2

- Agent-oriented Methodologies
- The Gaia Methodology

What is a methodology ?

To evaluate a methodology, we need to recall what a methodology is:

1: a body of methods, rules, and postulates employed by a discipline**:** a particular procedure or set of procedures

2 : the analysis of the principles or procedures of inquiry in a particular field

(Merriam-Webster)

- But when referring to software:
  - A methodology is the set of guidelines for covering the whole lifecycle of system development both technically and managerially.

Agent-oriented Methodologies

- They have the goal of
  - Guiding in the process of developing a multiagent systems
  - Starting from collection of requirements, to analisys, to design, and possibly to implementation

- An agent-oriented methodology defines the abstractions to use to model software:
  - Typically, agents, environments, protocols..
  - Plus additional methodology-specific abstractions

- And disciplines the software process:
  - What models and artifacts to produce and when
    - Model: an abstract representation of some aspect of interest of the software
    - Artifact: documents describing the characteristic of the software

Agent-oriented Methodologies

- A Variety of Methodology exists and have been proposed so far
  - Gaia (Zambonelli, Jennings, Wooldridge)
  - Prometeus (Winikoff and Pagdam)
  - SODA (Omicini)
  - ADELFE (Gleizes)
  - Etc.
- Exploiting abstractions that made them more suited to specific scenarios or to others..
- We focus on Gaia because is the reference one (i.e., the one any new proposal compares to) and the more general one
  - Ok, I am not an impartial judge…

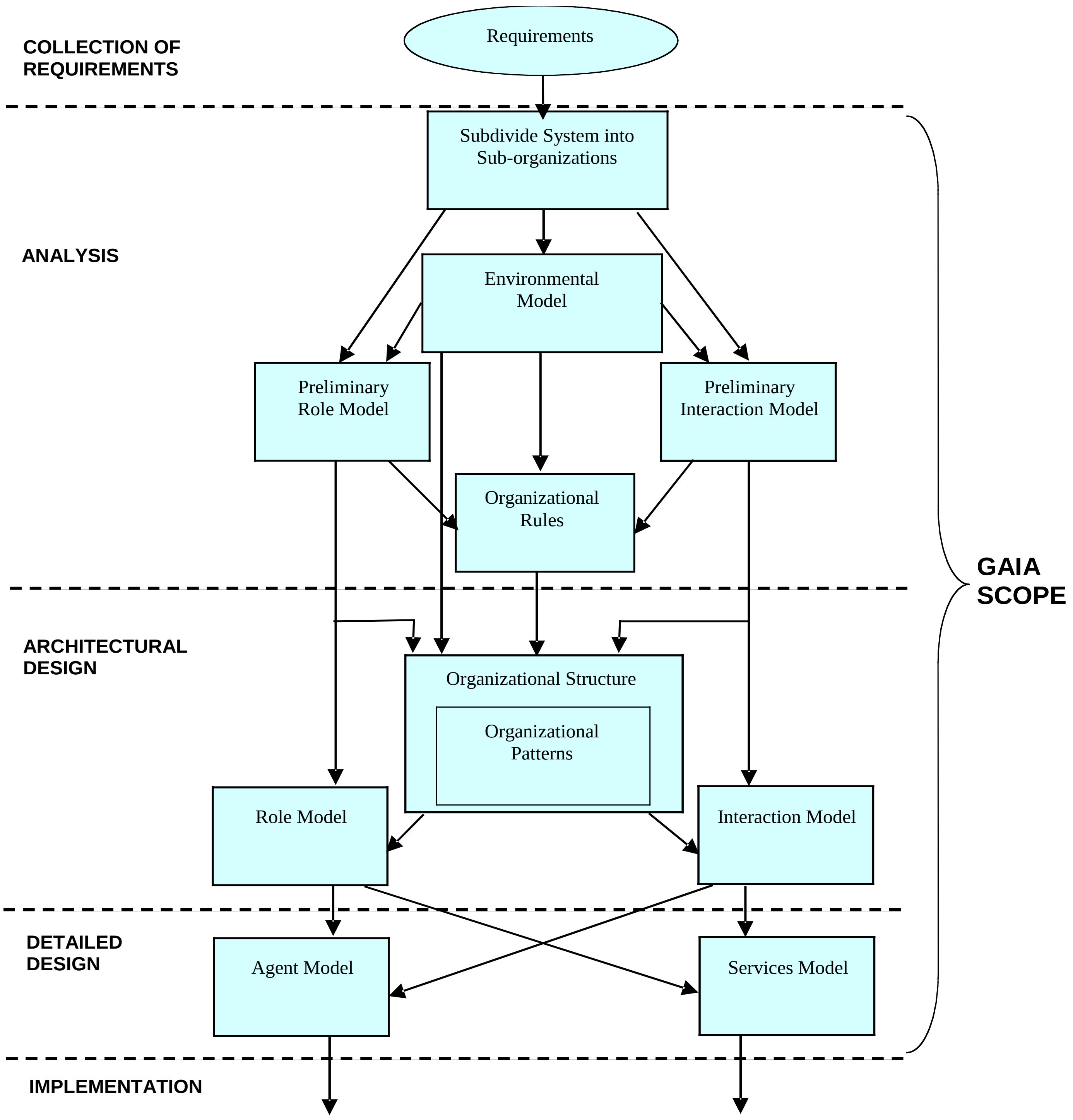
The Gaia Methodology

- It is “THE” AOSE Methodology
  - Firstly proposed by Jennings and Wooldridge in 1999
  - Extended and modified by Zambonelli in 2000
  - Final Stable Version in 2003 by Zambonelli, Jennings, Wooldridge
  - Many other researchers are working towards further extensions...
- Key Goals
  - Starting from the requirements (what one wants a software system to do)
  - Guide developers to a well-defined design for the multiagent system
  - The programmers can easily implement
  - Able to model and deal with the characteristics of complex and open multiagent systems

Key Characteristics of Gaia

- Exploits organizational abstractions
  - Conceive a multiagent systems as an organization of individual, each of which playing specific roles in that organization
  - And interacting accordingly to its role
- Introduces a clear set of abstractions
  - Roles, organizational rules, organizational structures
  - Useful to understand and model complex and open multiagent systems
- Abstract from implementation issues

Structure of Gaia Process



A Case Study:
Distributed Project Review

- The ministry for research publish a call for funding research
  - Scientists must “submit” a research proposal, e.g., in the form of a scientific article (paper)
- A number of scientists (called reviewers or referees”) review the papers and give marks
  - It has to complete a document called “review form”
  - To ensure fairness, the reviewers must be anonymous, expert, and must be willing to do the review,
  - Also, each project should receive a minimum number of review from different scientists
- Eventually, all accepted project/papers will sign a contract, will receive the funds, and will publish the results on a book

The Case Study: Why Agents?

- It is a typical case of distributed workflow management
  - There are actions to do on common documents
  - According to specific rules
- Each of the human actors involved in the process
  - Could be supported by a personal agents
  - Helping him to submit documents, filling in, respect deadlines, etc.

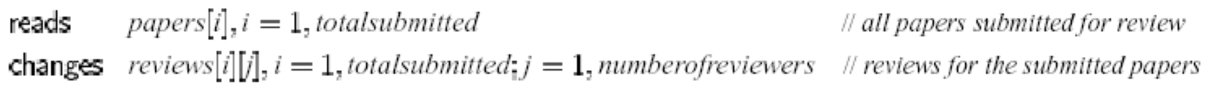
- Let’s see how we could develop this using the Gaia methodology..

Gaia Analysis (1)

- Once we know what the problem to solve is
- **First: Sub-organizations**
  - See if it can easily conceived as a set of loosely interacting problems
  - To be devoted to different sub-organization
  - And let’s focus on the different sub-organizations
  - “Divide et impera”
- **Second: Environment**
  - Analyze the operational environment
  - See how it can be modeled in terms of an agent environment
  - Resources to be accessed and how
  - So as to obtain an “environmental” model

Case Study Analisys (1)

- First: Sub-organizations
  - There are clearly different organizations in time
    - The submission of paper,
    - The review of paper
    - The Contractual phase for accepted ones
- Second: Environment
  - The environment is clearly a computational environment of digital resources
  - Filled in with papers and review forms
    - And possible with “user profiles” describing the attitudes, expertises, and possibly the conflicts of interest of scientists



```
reads    papers[i], i = 1, totalsubmitted                                   // all papers submitted for review
changes  reviews[i][j], i = 1, totalsubmitted; j = 1, numberofreviewers     // reviews for the submitted papers
```

Gaia Analysis (2)

- **Third: Roles**
  - See what “roles” must be played in the organization
  - A role defines a “responsibility” center in the organization, with a set of expected behaviors
  - So that its goals can be achieved
  - Defines the attributes and the responsibility of each role, reasoning in terms of “sub-goals”
  - So as to define the “role model”, i.e., the list specifying the characteristics of the various roles
- **Fourth: Protocols**
  - See how roles must interact with each other so as to fulfill expectations
  - Analyze these interaction protocols
  - So as to define an “interaction model”, i.e., the list specifying the characteristics of the various protocols

Case Study Analysis (2)

- **Third: Roles**
  - There are clearly such roles such as
    - “chair” (who received submissions and control the review process)
    - “author” (who send submissions)
    - “reviewer” (who receive papers to review and send back review forms)
  - Each with different permissions related to the environment (e.g., authors cannot access review forms) and with different responsibilities (reviewers must fill in the review form in due time)
- **Fourth: Protocols**
  - Protocols can be easily identified
    - “submit paper FROM author TO chair”
    - “send paper to review FROM chair TO”
    - Etc.

Gaia Analysis (3)

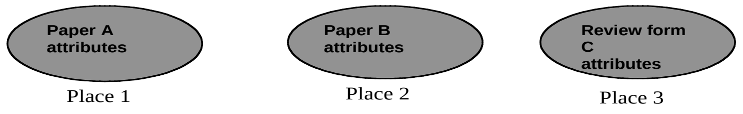
- **Fifth: Organizational Rules**
  - Analyze what “global” rules exists in the system that should rule all the interactions and the behavior between roles
  - These defines sorts of “social rules” or “laws” to be enacted in the organization
  - The list of all identified rules, that we call “organizational rules”, define the last model of the analysis

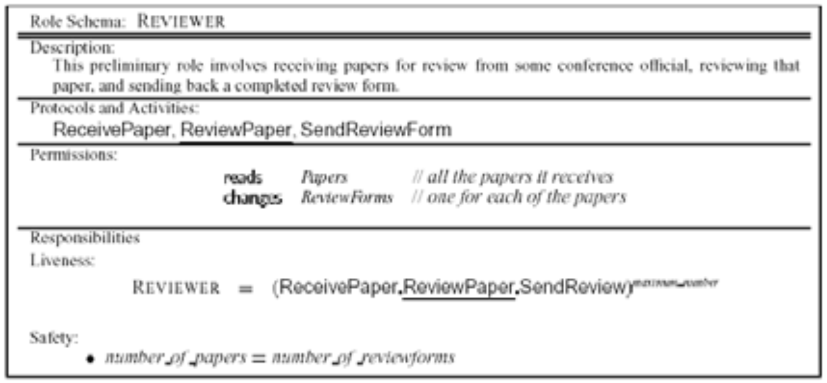
Case Study Analysis (3)

- **Fifth: Organizational Rules**
  - The process should clearly occur according to some rules ensuring fairness of the process
  - An author should not also act as reviewer for his own projects, or for those of his “friends”
  - A reviewer should not give two review for the same project
  - Each project should receive the same minimal number of review
  - And other you may think of…

Gaia Analysys: Graphical Representation of Models

- Environment
- Roles
- Interactions
- Organizational Rules





Role Schema: REVIEWER

Description:
This preliminary role involves receiving papers for review from some conference official, reviewing that paper, and sending back a completed review form.

Protocols and Activities:
ReceivePaper, ReviewPaper, SendReviewForm

Permissions:

reads	*Papers*	*// all the papers it receives*
changes	*ReviewForms*	*// one for each of the papers*

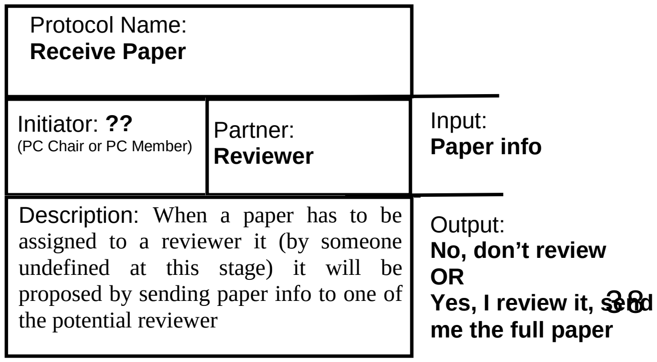
Responsibilities
Liveness:

$$\text{REVIEWER} = (\text{ReceivePaper}.\underline{\text{ReviewPaper}}.\text{SendReview})^{maximum\_number}$$

Safety:
- $number\_of\_papers = number\_of\_reviewforms$

$$\neg(\text{REVIEWER}(paper(x)) \mid \text{AUTHOR}(paper(x)))$$

$$\text{REVIEWER}(paper(i))^{3+}, i = 1, \ldots, number\_of\_submitted\_papers$$



Protocol Name: **Receive Paper**		
Initiator: **??** (PC Chair or PC Member)	Partner: **Reviewer**	Input: **Paper info**
Description: When a paper has to be assigned to a reviewer it (by someone undefined at this stage) it will be proposed by sending paper info to one of the potential reviewer		Output: **No, don't review OR Yes, I review it, send me the full paper**

From Analysis to Design

- Once all the analysis model are in place
  - We can start reasoning at how organizing them into a concrete architecture
- An “agent architecture” in Gaia is
  - A full specification of the **structure of the organization**
  - With full specifications on all the roles involved
  - With full specification on all interaction involved
- It is important to note that in Gaia
  - Role and Interaction models are “preliminar”
  - They cannot be completed without choosing the final structure of the organization
    - Defining all patterns of interactions
    - Introducing further “organizational” roles
    - Arranging the structure so that the organizational rules are properly enacted

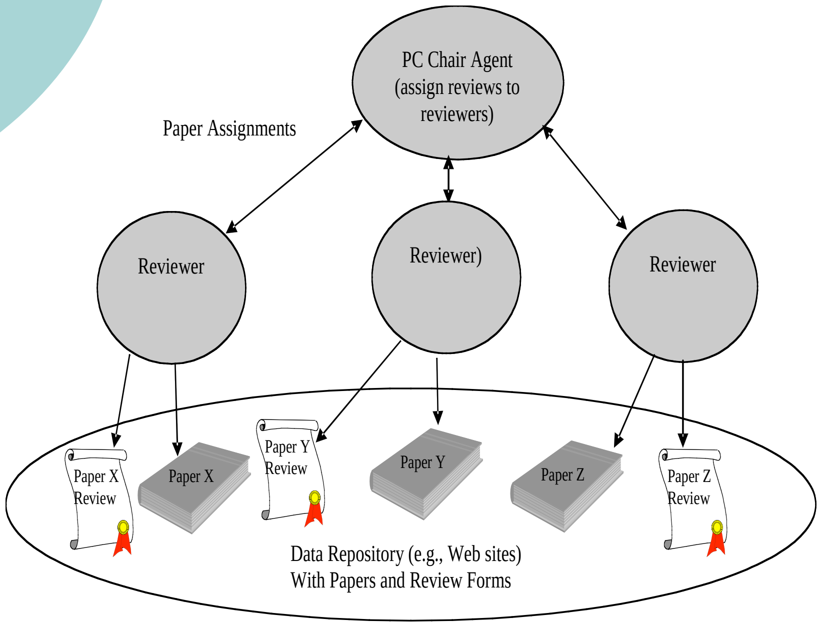
From Analysis to Design in the Case Study

- The final organizational of the review process may imply
  - Multi-level hierarchies to select papers (if there are a lot of submissions the “chair” must be supported by “co-chairs”)
  - A Negotiation process to select reviewers (it is a difficult process, and agent could help in that to march papers with appropriate reviewers)
  - A structure that avoid cheating (where an authors is somehow allowed to act as reviewer of its own project)
- Then, it is clear that the analysis could not have determines the final structure and a definitive listing of roles and protocols

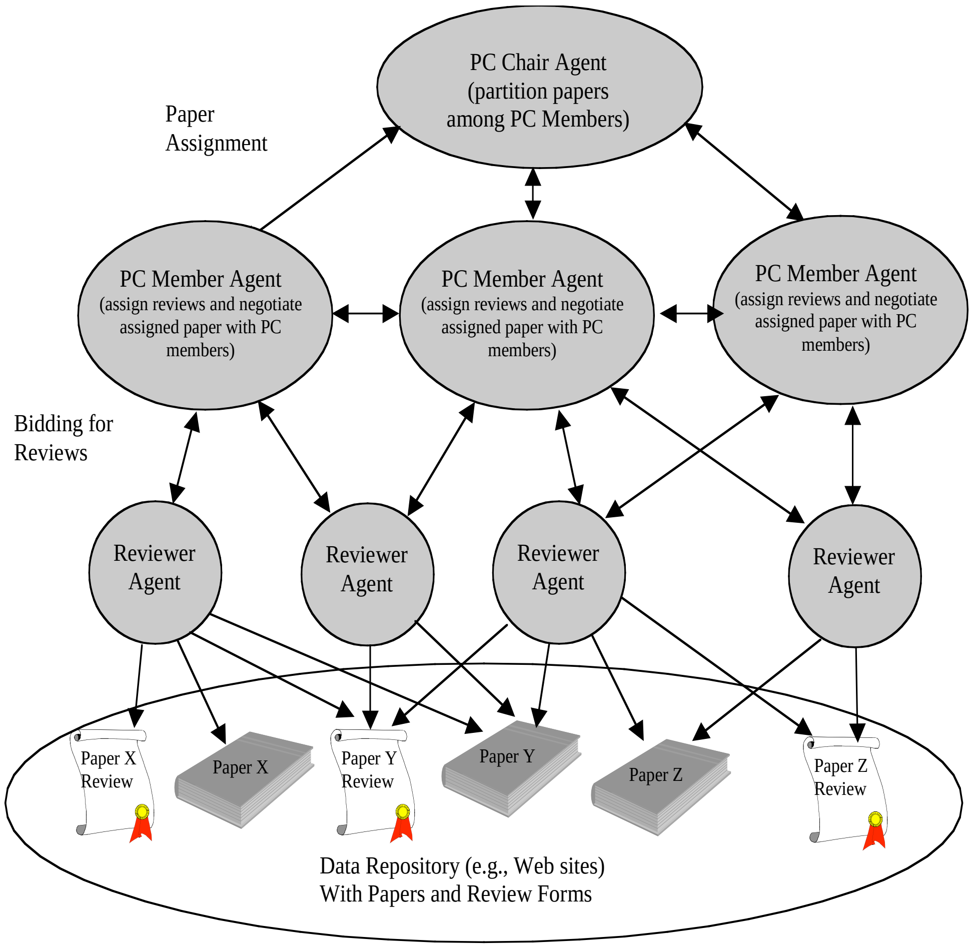
Gaia Architecture Design (1)

- Aimed at determining the final architecture of the system
- The architecture, i.e., the organizational structure consists in
  - The **topology** of interaction of all roles involved
    - Hierarchies, Collectives, Multilevel...
    - Which roles interact with which
  - The “**control regime**” of interactions
    - What type of interactions? Why?
    - Control interactions, Work partitioning, work specialization, negotiations, open markets, etc.

Case Study: Possible Organizational Structures



PC Chair Agent (assign reviews to reviewers)
Paper Assignments
Reviewer
Reviewer)
Reviewer
Paper X Review
Paper X
Paper Y Review
Paper Y
Paper Z
Paper Z Review
Data Repository (e.g., Web sites)
With Papers and Review Forms

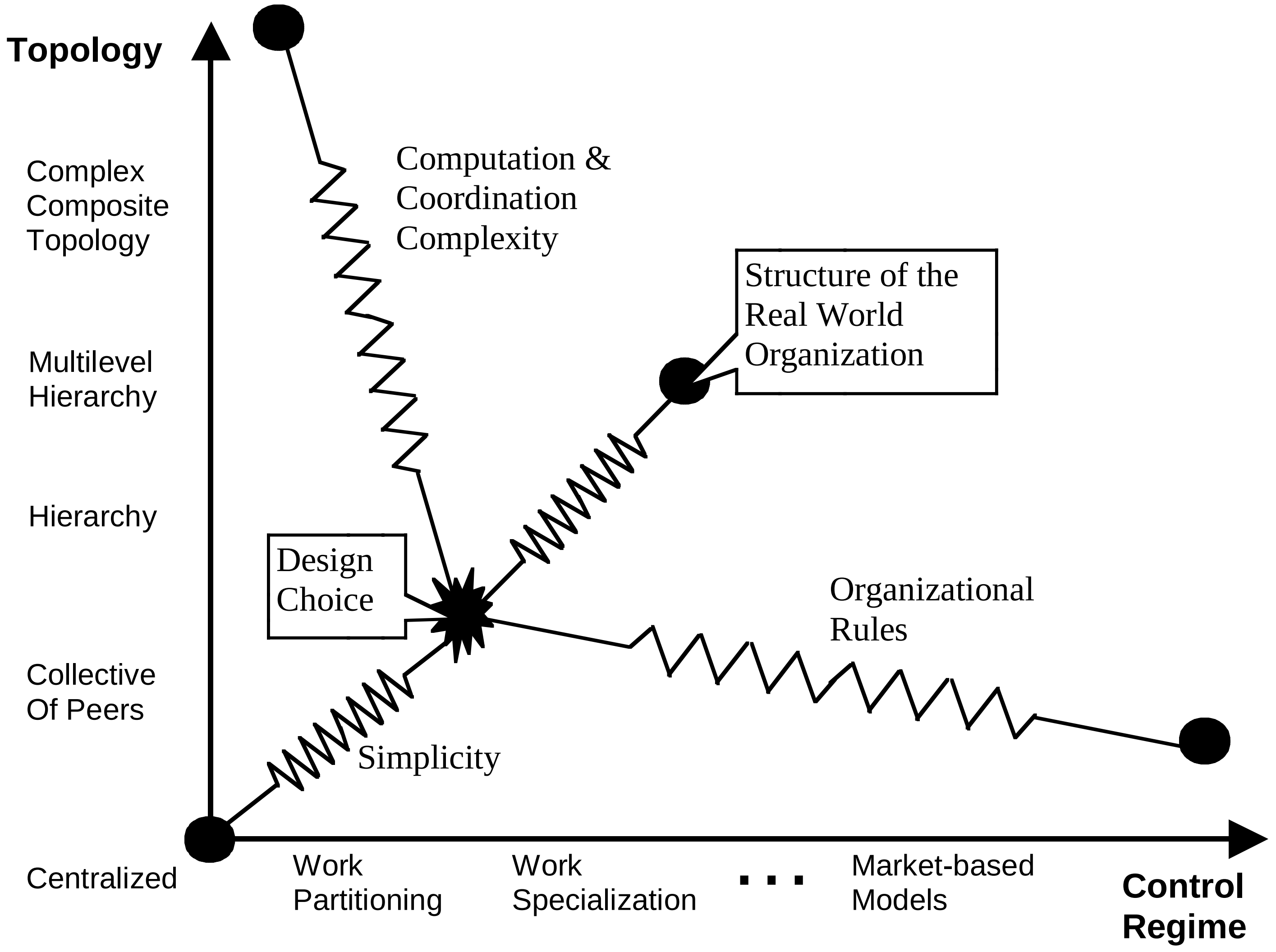


PC Chair Agent (partition papers among PC Members)
Paper Assignment
PC Member Agent (assign reviews and negotiate assigned paper with PC members)
PC Member Agent (assign reviews and negotiate assigned paper with PC members)
PC Member Agent (assign reviews and negotiate assigned paper with PC members)
Bidding for Reviews
Reviewer Agent
Reviewer Agent
Reviewer Agent
Reviewer Agent
Paper X Review
Paper X
Paper Y Review
Paper Y
Paper Z
Paper Z Review
Data Repository (e.g., Web sites)
With Papers and Review Forms

Gaia Architecture Design (2)

- What “forces” determines/influence the organizational structure?
- Simplicity
  - Simple structures are always preferable
- The Real-World organization
  - Trying to mimic the real-world organization minimizes conceptual complexity
- Complexity of the problem
  - Calls for distributed structures, with many components involved
- The need to enact organizational rules with small effort
  - Calls for exploiting negotiations as much as possible,
  - Also to deal with open systems,

Choosing the Organizational Structure



Gaia Architecture Design (3)

- It is important to note that in the definition of the organizational structure
  - This can be composed from a set of known “organizational patterns”
  - So that previous experiences can be re-used
- Once the organizational structure is decided
  - Complete the role model
  - Additional roles may have been introduced due to the specific structure chosen
- Complete the interaction model
  - To account for all interactions between all roles in a detailed way

Gaia Detailed Design

- Devoted to transform “roles” and “interaction protocols” into more concrete components, easy to be implemented
- Roles becomes agents
  - With internal knowledge, a context, internal activities, and services to be provided
  - Sometimes, it is possibly thinking at compacting the execution of several roles into a single agent
  - Clearly, we can define “agent classes” and see what and how many instances for these classes must be created
- Interaction protocols becomes sequence of messages
  - To be exchanged between specific agents
  - Having specific content and ontologies
- And the final specifications go to the programmers...

About Gaia Notations

- Gaia adopt a custom notation for its models
  - However, Gaia does not prescribe this
  - Any other graphical or textual notations (e.g. UML or whatever) can be used or can complement the Gaia one

Part 3:

- Implementation Issues and Multiagent Infrastructures

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
  - The methodology could strongly invite to exploit a specific infrastructure

Intra-agent Issues: Implementing Agents

- We have two main categories of tools to implement agents:
  - Object-oriented tools: are very much related to the object-oriented approach, e.g., Aglet;
  - BDI toolkits: are based on the BDI model (e.g., Jade).
- The choice of the tool to adopt is hard and there is no general answer:
  - Performances;
  - Maintenance;
  - … and many other issues.
- We have already discussed about Aglets and JADE agent implementation models, so we skip them now…

Inter-agent Issues: 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?

Multiagent Infrastructures

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

Communication vs. Coordination Infrastructures

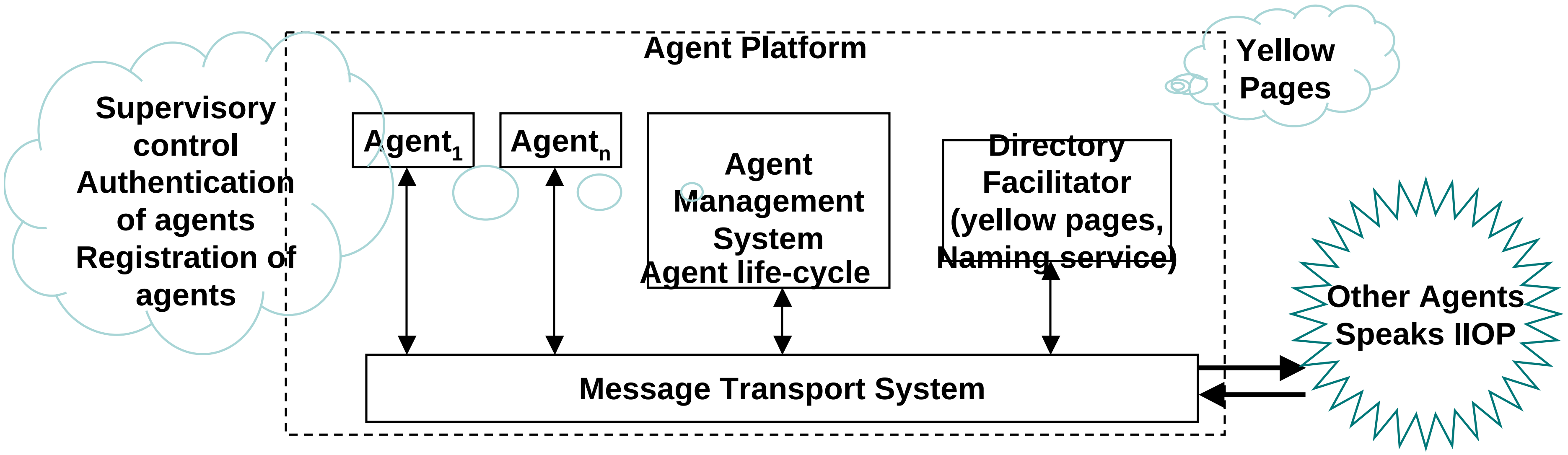
- Communication Infrastructures
  - Middleware layer mainly devoted to provide communication facilities
    - Routing messages, facilitators, etc.
    - FIPA defines a communication infrastructure
  - Communication enabling
- Coordination Infrastructure
  - Middleware layer mainly devoted to orchestrate interactions
    - Synchronization, and constraints on interactions
    - MARS and Tucson are coordination infrastructures
  - Activities ruling

Communication Infrastructure

- Agent in a MAS have to 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

FIPA Specifications for Communication Infrastructures

- The Foundation for Intelligent Physical Agents
- Specifies STANDARDS for multiagent infrastructures
  - to interoperate and be managed
- Formally specified ACL
  - Specifies encoding, semantics, and pragmatics of messages
- Includes: mobility, security, ontology, Human-Agent comm.
- FIPA reference architecture (see below)


Agent Platform
Yellow Pages
Supervisory control Authentication of agents Registration of agents
Agent$_1$
Agent$_n$
Agent Management System Agent life-cycle
Directory Facilitator (yellow pages, Naming service)
Other Agents Speaks IIOP
Message Transport System

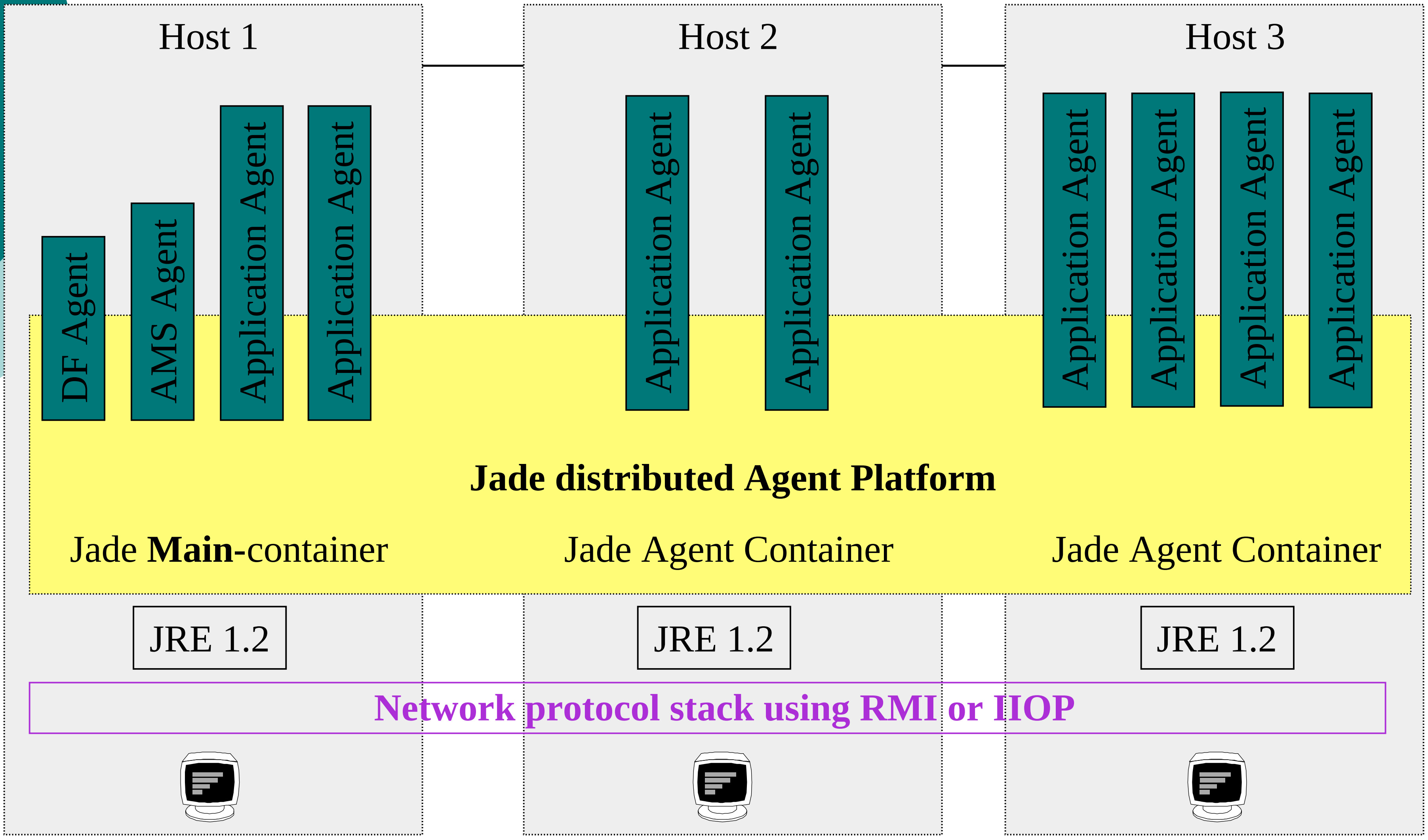
JADE (Java Agent DEvelopment Framework)

- JADE – A FIPA-compliant Agent Framework
  - http://sharon.cselt.it/projects/jade/
- Is a software framework
  - simplifies the implementation of multi-agent systems
  - Attempts to be very efficient
  - Fully implemented in Java and fully distributed under LGPL
  - Mostly oriented to ***AGENT COMMUNICATIONS*** (via ACL)
- Definitely the most used systems
  - AND IT IS ITALIAN!!!
  - Developed by UNIPR and TELECOM-IT

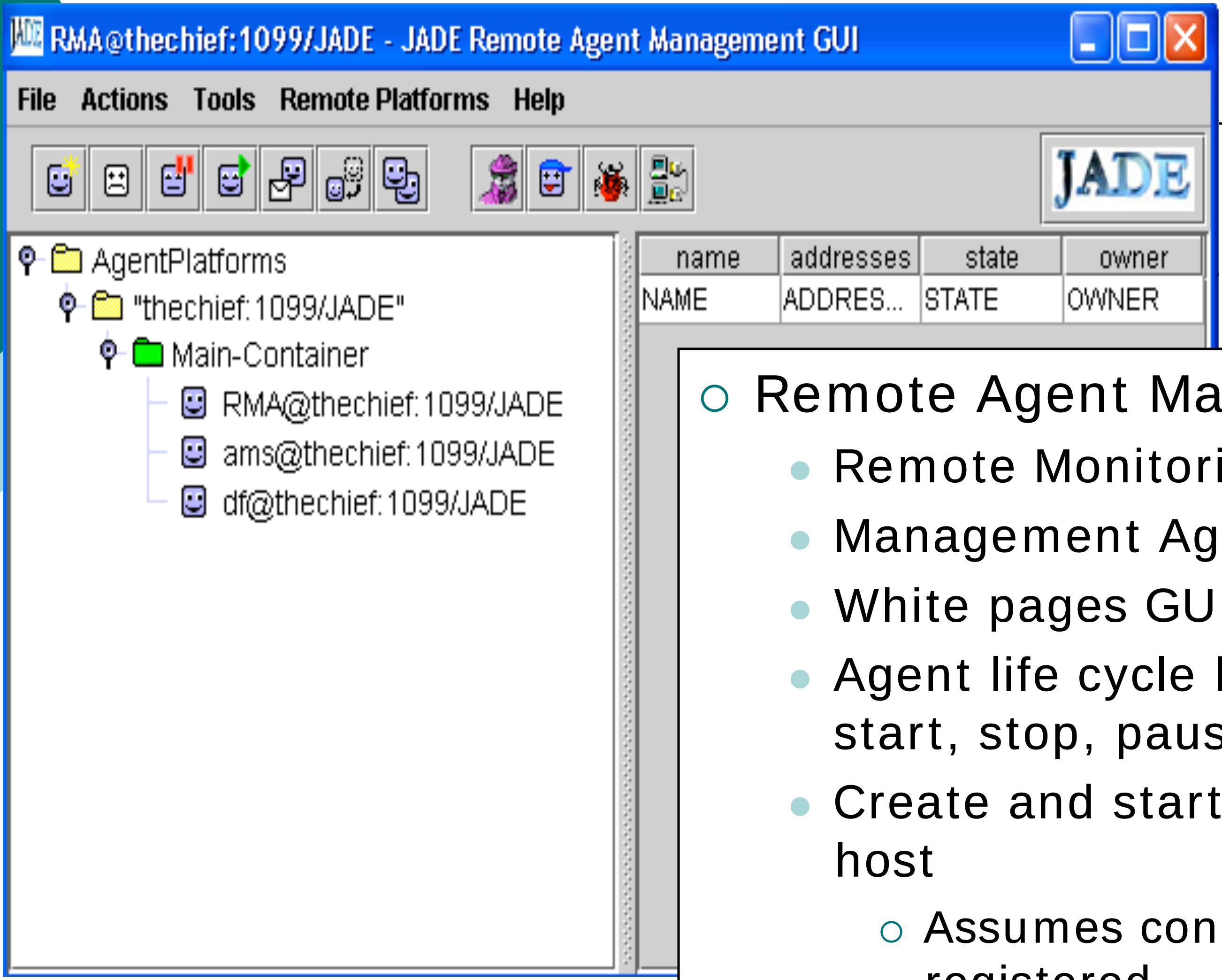
JADE continued

- Is the middleware for MAS (Multi-Agent Systems)
  - Target users: agent programmers for MAS
  - Agent services
    - life-cycle (to handle creation and death of agents), yellow-pages (naming service), message transport (to have different platforms interoperate)
  - Agent Communication Languages
    - Support for Speech Act and Negotiation protocols
    - Support for Shared Ontologies
  - 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

Distributed architecture of a JADE Agent Platform



JADE Agent Platform - GUI


RMA@thechief:1099/JADE - JADE Remote Agent Management GUI

AgentPlatforms
"thechief:1099/JADE"
Main-Container
RMA@thechief:1099/JADE
ams@thechief:1099/JADE
df@thechief:1099/JADE

name	addresses	state	owner
NAME	ADDRES...	STATE	OWNER

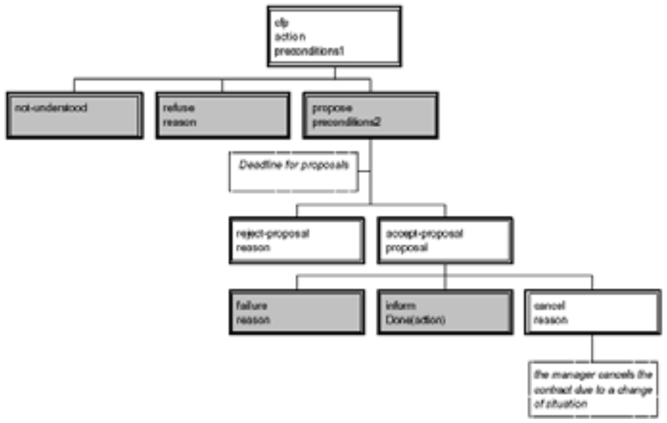
- Remote Agent Management
  - Remote Monitoring Agent
  - Management Agent
  - White pages GUI – to find agents
  - Agent life cycle handling allowing start, stop, pause, migrate, etc.
  - Create and start agents on remote host
    - Assumes container already registered
  - Naturally uses ACL for communication

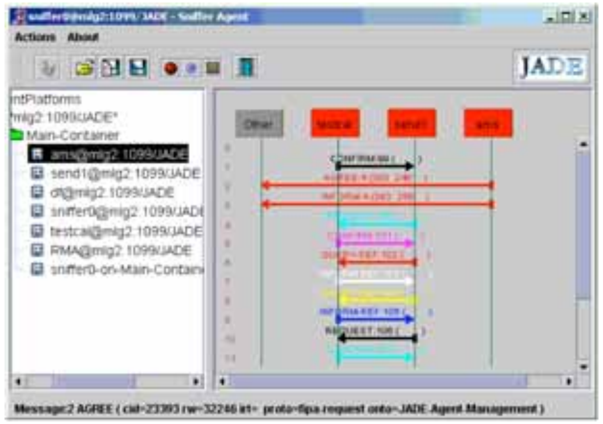
JADE Communication Sub-system

- 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
- If you send a message to another agent and the sub-system can't find target, then it sends it to the AMS to handle
- Graphics tools to analyse agent communications

JADE Interaction Protocols

- 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 behaviors.
- Graphics Tools to Analyse Protocols





Message:2 AGREE (cid=23393 rw=32246 irt= proto=fipa-request onto=JADE-Agent-Management)

Software Engineering with Communication Infrastructures

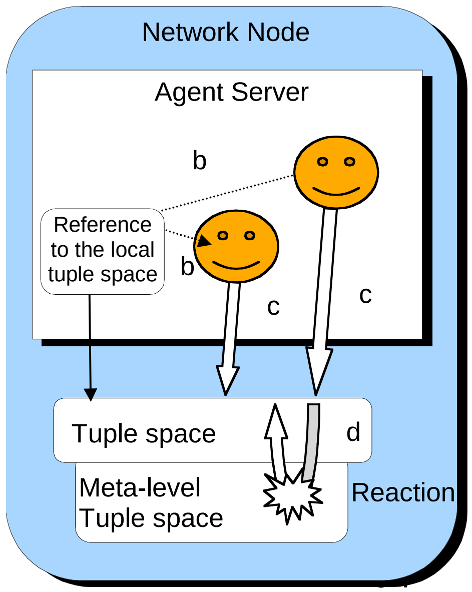
- All application problems are to be identified and designed in terms of
  - Internal agent behaviors and inter-agent interaction protocols
  - These include, from the intra-agent engineering viewpoint:
    - Controlling the global interactions
    - Controlling self-interested behaviours
- Advantages:
  - 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

Coordination Infrastructures

- The infrastructure is more than a support to communication
  - Other than enabling interactions...
  - It can embed the "laws" to which interaction must obey
    - E.g., to specify which agents can execute which protocols and when
    - E.g., Gaia organizational rules
  - It can control the adherence of the MAS behavior to the laws
    - E.g., to prevent malicious behaviors
  - Such laws can be re-configured depending on the application problem
    - E.g., English vs. Vickery auctions have different rules

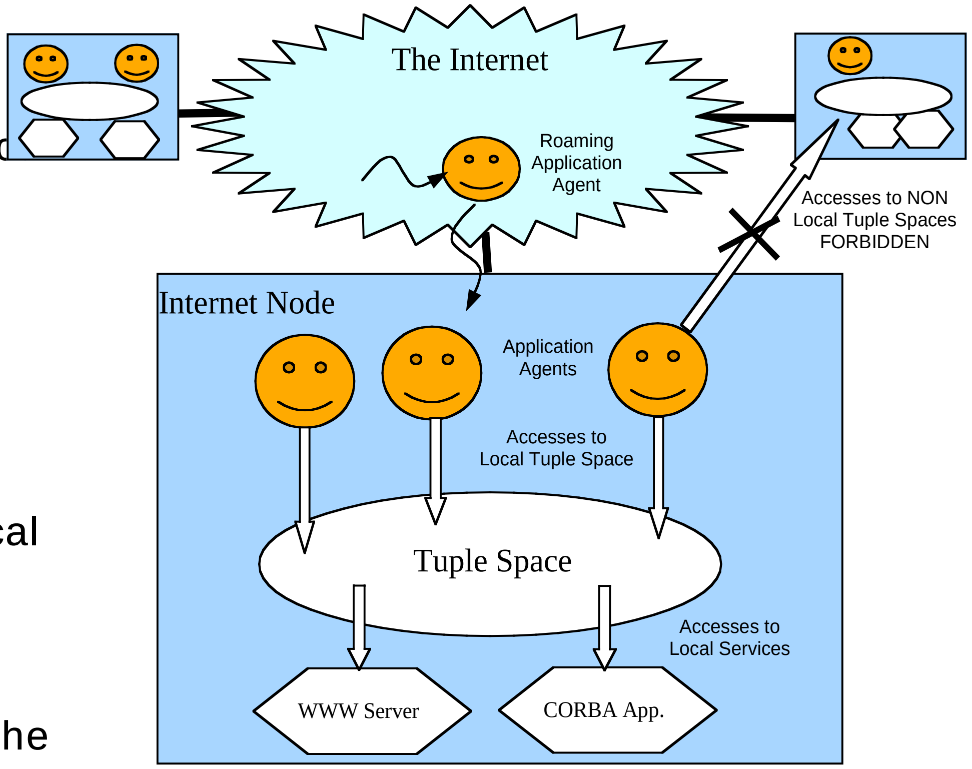
The MARS Coordination Infrastructure

- Mobile Agent Reactive Spaces
  - Developed at the University of Modena e Reggio Emilia
  - Ported on different agent systems (*Aglets*, *Java2Go*, *SOMA, JADE*)
  - Strictly related to TUCSON
- One shared data space on each node
- **“Tuple spaces”**
  - Attributed-based access to local resources
- Programmable tuple spaces
  - Based on the original idea of programmable coordination media (Omicini & Denti 98)
  - A “meta-level” can control and monitor all agent interactions



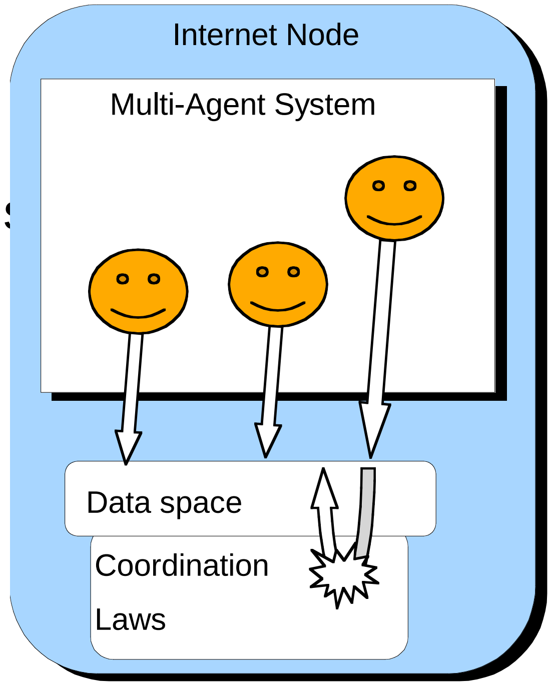
MARS Features

- Mobile agents roam the Internet
  - On each node, they connec to a local tuple space
- They can access it to retrieve/put data
  - Data can be accessed via attributes
  - Mediated interactions between agents via the local tuple space
  - Coordination and various interactions protocols as sequences of accesses to the tuple space
- Access to local resources
  - appears to agents as access to data in the tuple space


The Internet
Roaming Application Agent
Accesses to NON Local Tuple Spaces FORBIDDEN
Internet Node
Application Agents
Accesses to Local Tuple Space
Tuple Space
Accesses to Local Services
WWW Server
CORBA App.

Programmable Coordination in MARS

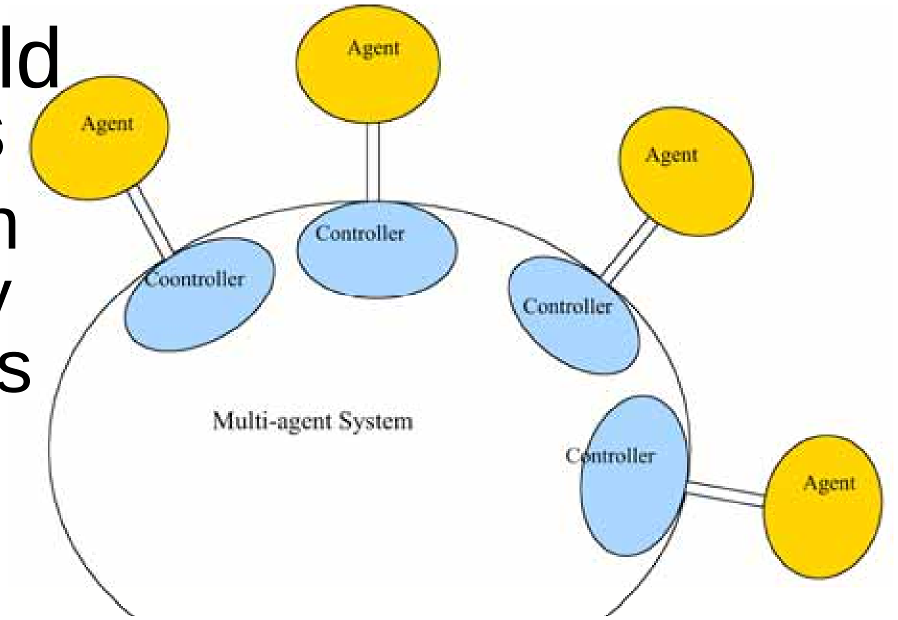
- The Tuple space of MARS is fully programmable
  - It can control and influence all interactions
- The data space can embed the coordination laws
  - Ruling, other than enabling, interactions
- Global control on the behavior of the MAS can be enacted
  - Interaction actions can be influenced and constrained
  - Control of self-interested behavior and errors
- Ease of maintenance
  - To change the behavior of the MAS, no need of changing agents, only coordination laws
  - e.g., from English to Vickery auction



Example of Coordination Infrastructures: Fishmarket

- Each agents in a MAS
  - Is dynamically attached a controller module
  - In charge of controlling its external actions (i.e., protocol execution)

- Inspired by real-world fish market auctions
  - Fishers participate in auctions by implicitly respecting local rules
  - There is an implicit (institutional) control



Software Engineering with Coordination Infrastructure (1)

- Clear separation of concerns
  - Intra-agent goals
  - Global MAS goals and global rules of the organizations
  - Such separation of concerns has to reflect in analysis and design
- Example: the ***Gaia methodology version 2***
  - Explicitly tuned to open MAS
  - Implicitly assuming the presence of a coordination infrastructure
    - Identification of global organizational rules as a primary abstraction in the software process

Software Engineering with Coordination Infrastructure (2)

- Advantages
  - Separation of concerns reduces complexity in analysis and design
    - Inter-agent issues separated from intra-agent ones
  - Design for adaptivity perspective
    - Agents and rules can change independently
  - Intelligence in the infrastructure
    - A trend in the scenario of distributed computing
- Drawbacks
  - Implement both agents and infrastructural programs
  - Agents are no longer the only active components of the systems
    - No longer homogeneous
  - Lack of standardization

Institutions

- May basic researches in the area of MAS recognize that:
  - Agents do not live and interact in a virgin world
    - Agents live in a society, and as that they have to respect the rules of a society
    - Agents live in an organization, which can effectively executed only in respect of organizational patterns of interactions
- In general: Multiagent systems represent ***institutions***
  - Where agents must conform to a set of expected behavior in their interactions
  - Such an approach requires the introduction of a conceptual coordination infrastructure during analysis and design (as in Gaia v. 2)

Part 4

- Conclusions and Open Issues

Open Issues in AOSE

- Engineering MAS for Mobility & Ubiquity
  - What models and methodologies? What infrastructures?
- Emergent Behavior: Dynamic systems & Complexity
  - Relations between MAS and complex systems
  - Exploiting emergence behavior in MAS
- MAS as Social Systems
  - Relations with social networks and social organizations
  - Self-organization
  - Performance models
- Performance models for MAS
  - How to “measure” a MAS
  - In terms of complexity and efficiency?

Conclusions

- In our humble opinion, agents will become the dominant paradigm in software engineering
  - AOSE abstractions and methodologies apply to a wide range of scenarios
- Several assessed research works already exist
  - Modeling work
  - Methodologies
  - Implementation Tools
- Still, there are a number of fascinating and largely unexplored open research directions…
  - Ubiquity, self-organization, performance….