

# Agent-Oriented Software Engineering

Multiagent Systems LS

Sistemi Multiagente LS

Ambra Molesini

`ambra.molesini@unibo.it`

Ingegneria Due

ALMA MATER STUDIORUM—Università di Bologna a Cesena

Academic Year 2006/2007



## 1 What is Agent-Oriented Software Engineering (AOSE)?

## 2 Survey on AOSE methodologies

- Introduction to Methodologies
- Method Engineering and Agent
- Gaia
- PASSI
- Tropos
- ADELFE



# 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



# 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 outcomes to produce



# 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.



# Tools

- Notation tools:
  - To represent the outcomes of the software development phases:
    - diagrams, equations, figures. . .
- Formal models:
  - To prove properties of software prior to the development:
    - Lambda and pi calculus, Petri-nets. . .
- CASE tools:
  - Based on notations and models, to facilitate activities:
    - simulators



# 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 and asks for
  - Making the set of abstractions required clear
  - Adapting methodologies and producing new tools
- Novel, specific agent-oriented software engineering approaches are needed!



# Agents: Weak 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
- A *multi-agent 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. . .



# SE Viewpoint on Agent-Oriented Computing

- We commit to weak viewpoint 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 agent-based one, can be indeed characterised in terms of weak multi-agent systems



# 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

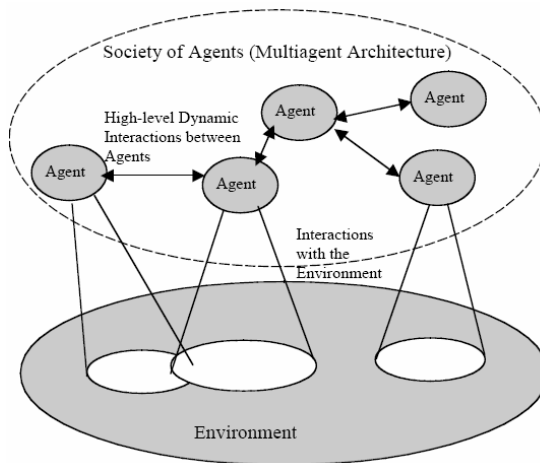


# SE Implications of Agent Characteristics

- Sociality
  - Not a single characterising protocol of interaction
  - Interaction as an additional SE dimension
- Openness
  - Controlling self-interested agents, malicious behaviours, and badly programmed agents
  - Dynamic re-organisation of software architecture
- Mobility and Locality
  - Additional dimension of autonomous behaviour
  - Improve locality in interactions



# MAS Characterisation



# Agent-Oriented Abstractions

- The development of a multi-agent system should fruitfully exploit *abstractions* coherent with the above characterisation
  - *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 among agents and between agents and resources of environment
- In addition, there may be the need of abstracting:
  - The *local context* where an agent lives (e.g., a sub-organisation of agents) to handle mobility & openness



# Why Agents and Multi-Agent Systems?

- Other lectures may have already outlined the advantages of (intelligent) agents and of multi-agent 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 features are very similar to those of agents and multi-agent systems
  - AOSE abstractions, methodologies, and tools are well suited for such software systems
- But of course. . .
  - AOSE may sometimes appear to be too “high-level”
  - There is a *gap* between the AOSE approach and the available technologies



# Agents and Multi-Agent 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 multi-agent systems:
  - Internet applications
  - P2P systems
  - Sensor networks
  - Pervasive computing systems



# Summarising

- 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 (in the following of the tutorial)
  - Appears to be applicable to a very wide range of distributed computing applications...



## 2 Survey on AOSE methodologies

- Introduction to Methodologies
- Method Engineering and Agent
- Gaia
- PASSI
- Tropos
- ADELFE



# What is a methodology?

- A methodology is a collection of methods covering and connecting different stages in a process. The purpose of a methodology is to prescribe a certain coherent approach to solving a problem in the context of a software process by preselecting and putting in relation a number of methods
- A methodology has two important components: one that describe the process elements of the approach, and a second that focuses on the work products and their documentation.



- The term “methodology” is often confused (particularly in the AOSE field) with the software development process
- In classical software engineering, there are substantial differences between the two things
- We will now introduce the definition of software development process and methodology as often adopted in the classical SE field





# SE definition about Methodology and Process

- A Software (Development) Process often refers to a *Software (Development) Process Model*:
  - It prescribes around which phases a process should be organised, in which order such phases should be executed, and when interactions and coordination between the work of the different phases should be occur
  - In other words, a process model defines a skeleton, a template, around which to organise and detail an actual process



# What is an AO methodology?

- AOSE methodologies mainly try to suggest a clean and disciplined approach to analyse, design and develop multi-agent systems, using specific methods and techniques
- AOSE methodologies, typically start from a *meta-model*, identifying the basic abstractions to be exploited in development
- On this base, they exploit and organise these abstractions so as to define guidelines on how to proceed in the analysis, design, and development, and on what output to produce at each stage



# Meta-model

- Meta-model enables checking and verifying the completeness and expressiveness of a methodology by understanding its deep semantics, as well as the relationships among concepts in different languages or methods
- *the process of designing a system (object or agent-oriented) consists of instantiating the system meta-model that the designers have in their mind in order to fulfill the specific problem requirements[1]*



# MAS Meta-model

- MAS meta-models usually include concepts like role, goal, task, plan, communication
- In the agent world the meta-model becomes a **critical element** when trying to create a new methodology because in the agent oriented context, to date, there are not common denominator
  - each methodology has its own concepts and system structure



# Agent-Oriented Methodologies

- A Variety of Methodology exists and have been proposed so far
  - Gaia (Zambonelli, Jennings, Wooldridge)
  - Tropos (Giorgini et all.)
  - PASSI (Cossentino)
  - ADELFE (Bernon et all.)
  - SODA (Omicini, Molesini)
  - Prometheus (Winokoff and Pagdam)
  - etc. . .



# Agent-Oriented Methodologies

- Exploiting abstractions that made them more suited to specific scenarios or to others . . .
- Should support mechanisms to manage the *complexity* of system description
- In this part we first introduce some concepts about Method Engineering
- then we show Gaia, PASSI, Tropos, ADELFE
- In part 3 (next lesson) we focus on SODA

Ok, I am not an impartial judge. . .



# Outline

- 1 What is Agent-Oriented Software Engineering (AOSE)?
- 2 Survey on AOSE methodologies
  - Introduction to Methodologies
  - Method Engineering and Agent
  - Gaia
  - PASSI
  - Tropos
  - ADELFE



# Method Engineering Goals

- Let the developer of a (multi-agent) system create his own methodology:
  - Suited for the specific problem/system to be built
  - Not conflicting with his (development) environmental constraints
  - Coherent with his (or his group) knowledge and skills
  - Supported by CASE tool
  - Using a standard modeling language
- This is the approach proposed within the IEEE-FIPA Methodology Working Group  
(<http://www.fipa.org/activities/methodology.html> )

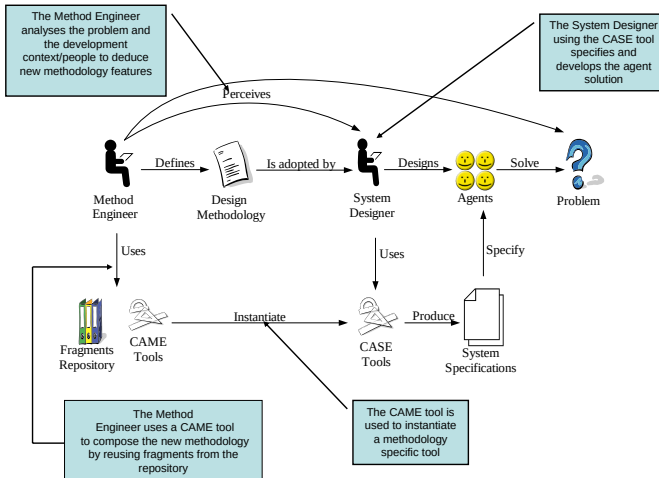


# The Proposed Approach

- Method Engineering
  - The development methodology is built by the developer assembling pieces of the process (*method fragments*) from a *method base*
  - The method base is composed of contributions coming from existing methodologies and other novel and specifically conceived fragments
- SPEM (Software Process Engineering Meta-model) is a standard from OMG (ver 1.1 is dated on 05-Jan-06)
  - It is a meta-model used to describe a software development process or a family of related software development processes



# Method Engineering



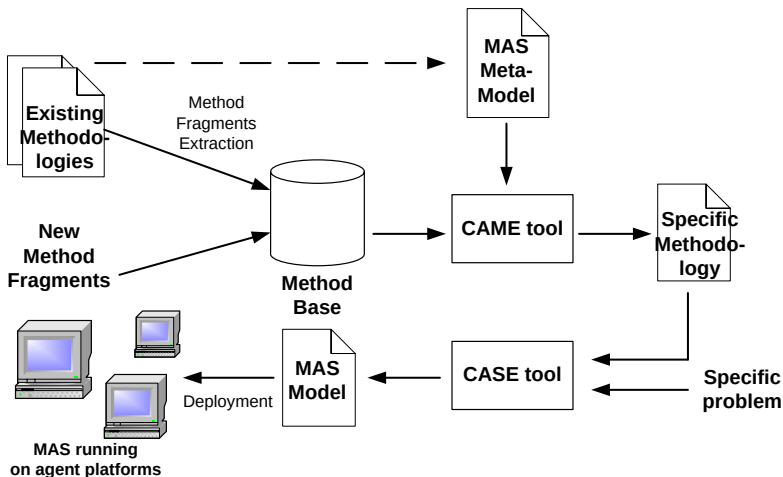
# What is a Method Fragment

A *fragment* is a portion of the development process, composed as follows:

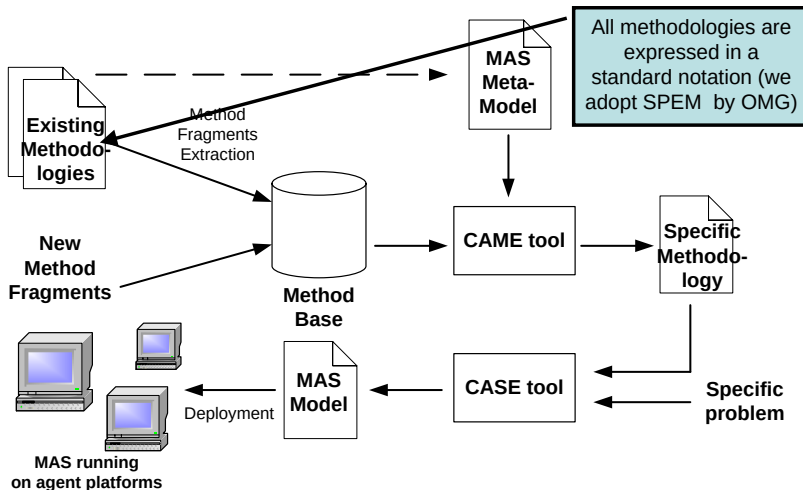
- A *portion of process* (what is to be done, in what order), defined with a *SPEM* diagram
- One or more *deliverables* (like (A)UML/UML diagrams, text documents and so on)
- Some *preconditions* (they are a kind of constraint because it is not possible to start the process specified in the fragment without the required input data or without verifying the required guard condition)
- A *list of concepts* (related to the MAS meta-model) to be defined (designed) or refined during the specified process fragment
- *Guideline(s)* that illustrates how to apply the fragment and best practices related to that
- A *glossary of terms* used in the fragment (in order to avoid misunderstandings if the fragment is reused in a context that is different from the original one)
- Other information (composition guidelines, platform to be used, application area and dependency relationships useful to assemble fragments) complete this definition.



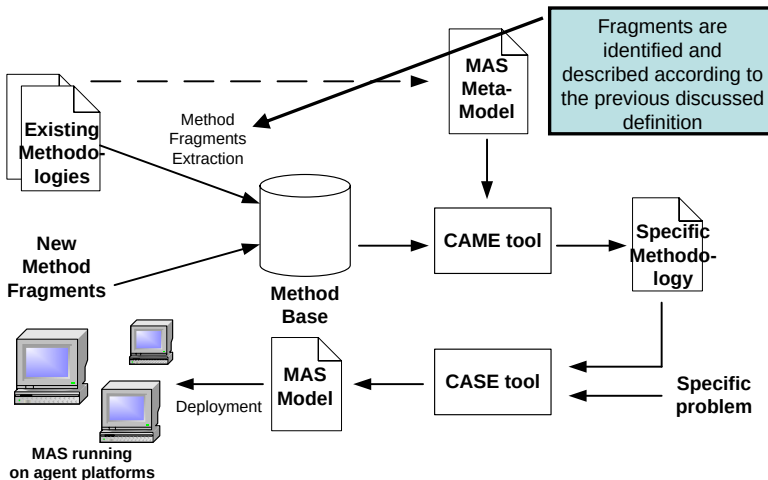
# The IEEE-FIPA Methodology Production Process



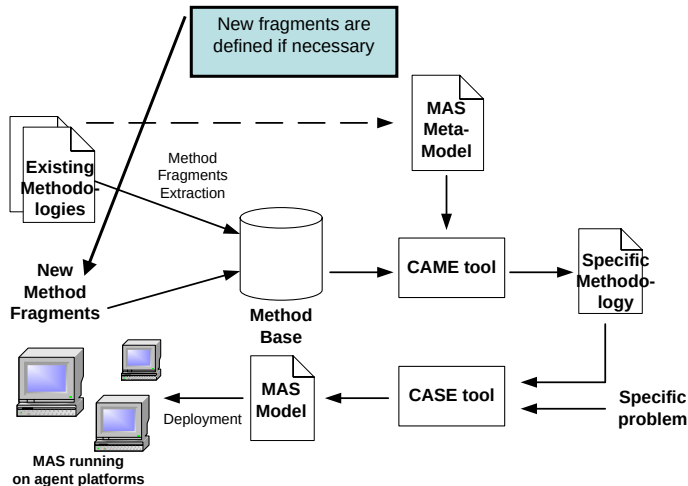
# The IEEE-FIPA Methodology Production Process



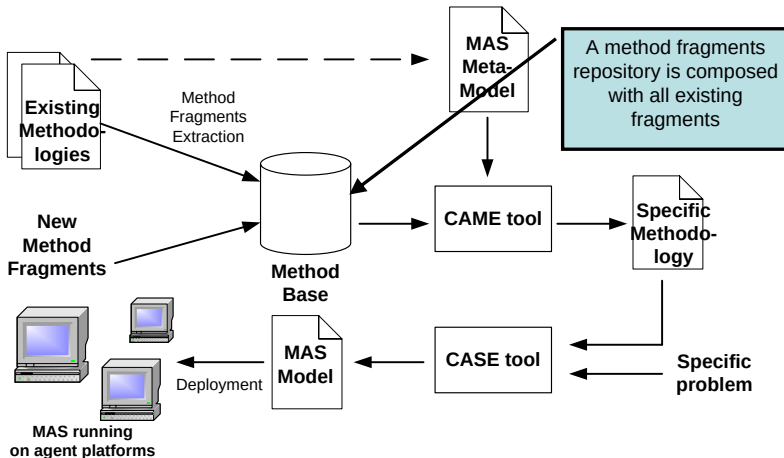
# The IEEE-FIPA Methodology Production Process



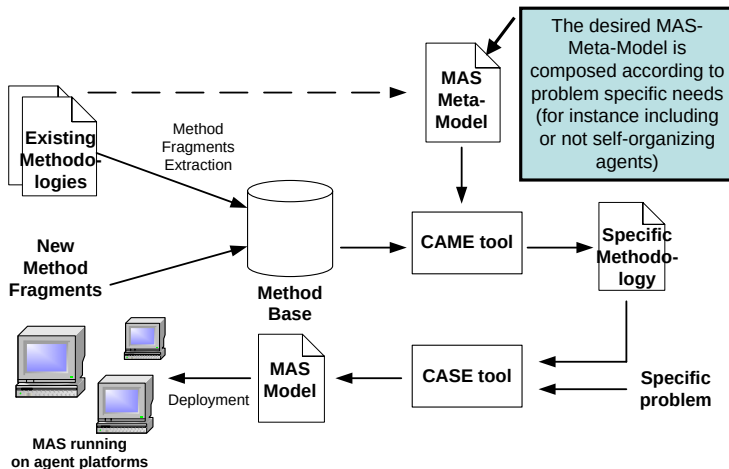
# The IEEE-FIPA Methodology Production Process



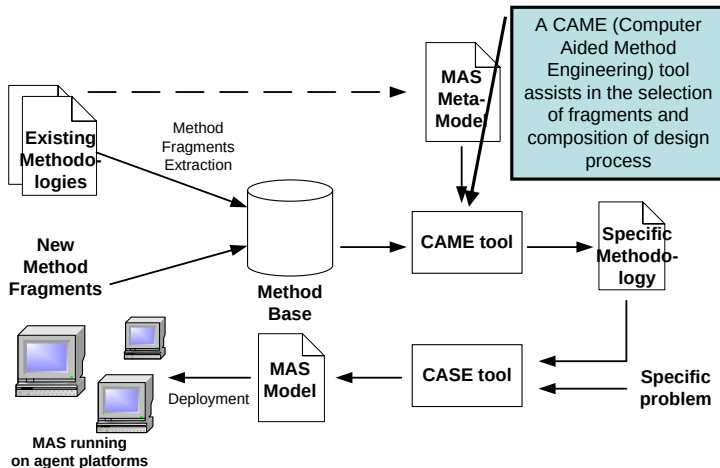
# The IEEE-FIPA Methodology Production Process



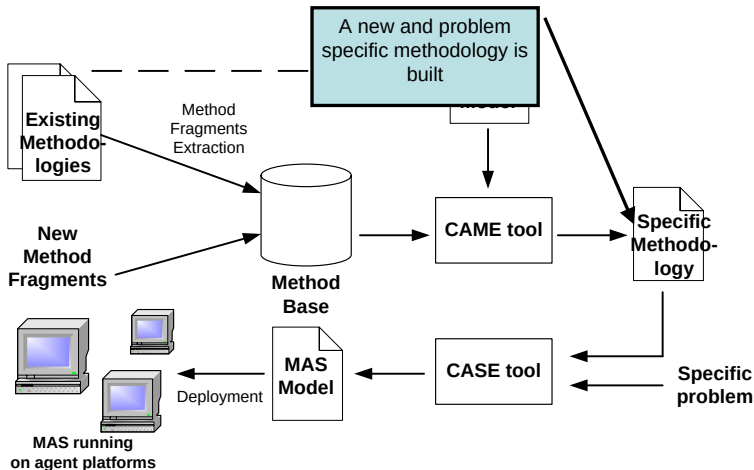
# The IEEE-FIPA Methodology Production Process



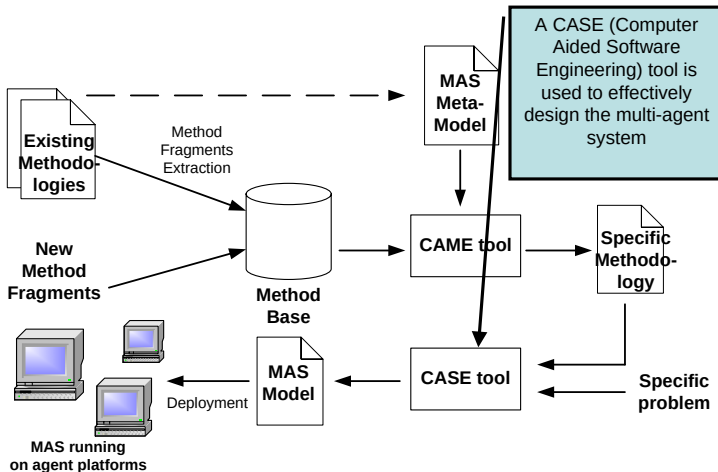
# The IEEE-FIPA Methodology Production Process



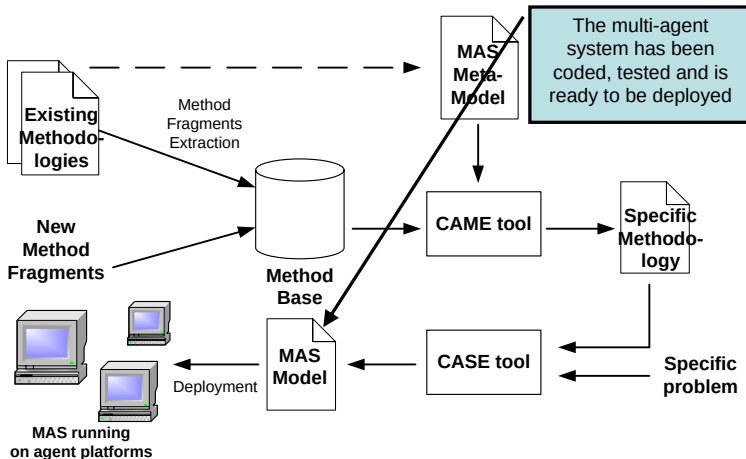
# The IEEE-FIPA Methodology Production Process



# The IEEE-FIPA Methodology Production Process



# The IEEE-FIPA Methodology Production Process



# CAME Tool

- This tool is based on the method meta-model and it is responsible for method fragment specification, i.e. their product and process parts definition.
- Method fragment specification can be done “from scratch”, by assembly or by modification.
- In the first case product and process models of the fragments are defined by instantiating the method meta-model used by the tool.
- In the second case fragments are assembled in order to satisfy some specific situation.
- In the third case fragments are obtained by modification of other fragments in order to better satisfy the method goal.
- Depending to the method meta-model, the CAME tool should offer graphical modelling facilities and special features.



# OO vs AO Method Engineering

- Method Engineering has been introduced in the object oriented (OO) context some years ago
- It could seem that introducing the method engineering paradigm in the agent oriented (AO) context is a plain operation.
- It is not so, because in the OO context the construction of method fragments (pieces of methodology), the assembling of the methodology with them and the execution of the design rely on a common denominator, **the universally accepted concept of object and related model of the object oriented system**
- In the agent context, there is not an universally accepted definition of agent nor it exists any very diffused model of the multi-agent system



# Outline

- 1 What is Agent-Oriented Software Engineering (AOSE)?
- 2 Survey on AOSE methodologies
  - Introduction to Methodologies
  - Method Engineering and Agent
  - **Gaia**
  - PASSI
  - Tropos
  - ADELFE



# The Gaia Methodology

- It is the most known 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 multi-agent system
- Model and dealing with the characteristics of complex and open multi-agent systems
- Easy to implement



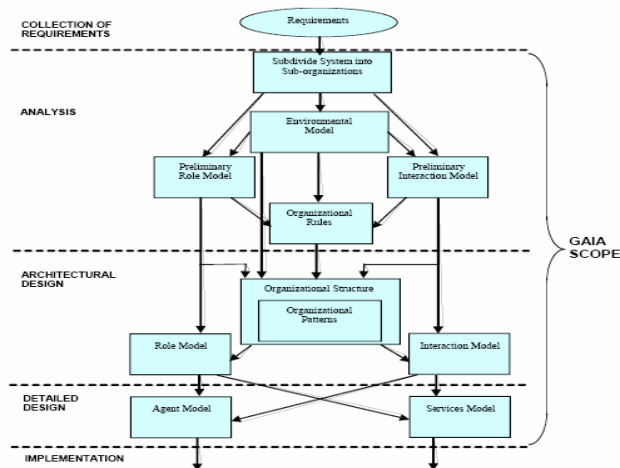
# Key Characteristics of Gaia

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

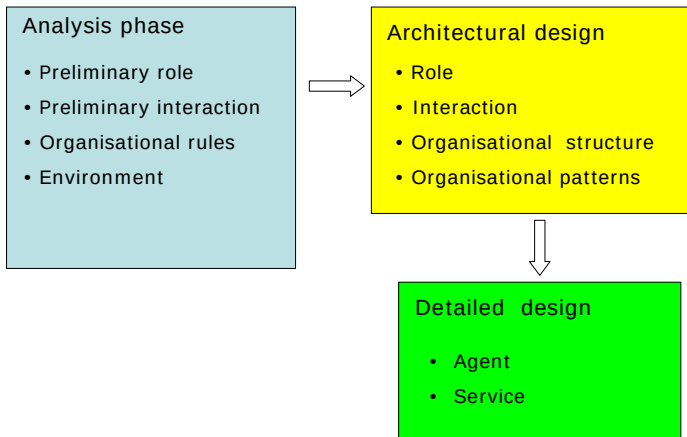




# Gaia: Models Relation



# Gaia: Abstractions



# Analysis Phase

- Sub-organisation
  - determining whether multiple organisations have to co-exist in the system
  - See if the system can easily conceived as a set of loosely interacting problems
- Environmental Model
  - Analyse the operational environment
  - See how it can be modelled in term of an agent environment
  - Resources to be access and how



# Analysis Phase

- Preliminary Role Model
  - See what *roles* must be played in the organisation
  - A role defines a *responsibility* centre in the organisation with a set of expected behaviours (permissions and responsibilities)
- Preliminary Interaction Model
  - See how roles must interact with each other so as to fulfil expectations
  - Definition of protocols for each type of inter-role interaction



# Analysis Phase

- Organisational Rules
  - Analyse what *global* rules exists in the system that should rule all the interactions and behaviour between roles
  - These defines sorts of *social rules* or *law* to be enacted in the organisation
  - Liveness rules define how the dynamics of the organisation should evolve over the time
  - Safety rules define time-independent global invariants for the organisation that must be respected



# Gaia Analysis: Graphical Representation of Models

## Environment



## Roles

## Interactions

## Organizational Rules

|                                                                                                                                                                           |                  |                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|-----------------------------------------------------------------------------------|
| Role Scheme: REVIEWER                                                                                                                                                     |                  |                                                                                   |
| Description:<br>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:<br>ReceivePaper, ReviewPaper, SendReviewForm                                                                                                    |                  |                                                                                   |
| Participations:                                                                                                                                                           |                  |                                                                                   |
|                                                                                                                                                                           | reads<br>changes | Papers // all the papers it receives<br>ReviewForms // one for each of the papers |
| Responsibilities                                                                                                                                                          |                  |                                                                                   |
| Licenses:<br>REVIEWER = (ReceivePaper, ReviewPaper, SendReviewForm) <i>no owner</i>                                                                                       |                  |                                                                                   |
| Satisfy:<br>• another of papers is number of reviewForms                                                                                                                  |                  |                                                                                   |

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

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

|                                                                                                                                                                               |                      |                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|---------------------------------------------------------------------------------|
| Protocol Name:<br>Receive Paper                                                                                                                                               |                      |                                                                                 |
| Initiator: ??<br>(PC Chair or PC Member)                                                                                                                                      | Partner:<br>Reviewer | Input:<br>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:<br>No, don't review<br>OR<br>Yes, I review it. @@@<br>me the full paper |



# From Analysis to Design

- Once all the analysis model are in place, we can start reasoning at how organising them into a concrete architecture
- An *agent architecture* in Gaia is
  - A full specification of the structure of the organisation
  - With full specifications on all the roles involved
  - With full specification on all interactions 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 organisation
    - Defining all patterns of interactions
    - Introducing further *organisational* roles
    - Arranging the structure so that the organisational rules are properly enacted



# Architectural Design Phase

- Aimed at determining the final architecture of the system
- The architecture, i.e., the organisational 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.



# Architectural Design Phase

- Choosing the Organisational Structure
  - Consideration about simplicity, real-world organisation complexity of the problem, need to enact organisational rule with small effort
  - Exploiting organisational Patterns
- Completion of role model with the *organisational roles* identified from the adoption of specific organisational structure
- Completion of interaction model with the *organisational protocols* derived from adopted organisational structure



# Detailed Design Phase

- 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



# Limitations

- Gaia does not deal directly with implementation issues
- Gaia does not deal with the activity of requirements capture and modelling and of early requirements engineering
- Gaia supports only the sequential approach to software development
- ... the Environment?
- ... The support to manage complexity?



# Outline

- 1 What is Agent-Oriented Software Engineering (AOSE)?
- 2 Survey on AOSE methodologies
  - Introduction to Methodologies
  - Method Engineering and Agent
  - Gaia
  - **PASSI**
  - Tropos
  - ADELFE



# Characteristics of PASSI

- PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology.
- The methodology integrates design models and concepts from both Object Oriented Software Engineering and MAS using UML notation
- PASSI refers to the most diffuse standards: UML, FIPA, JAVA, Rational Rose
- PASSI is conceived to be supported by PTK (PASSI Tool Kit) an agent-oriented CASE tool

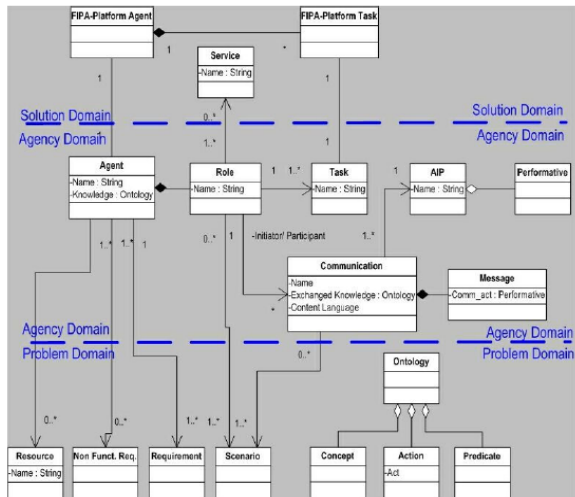


# Characteristics of PASSI

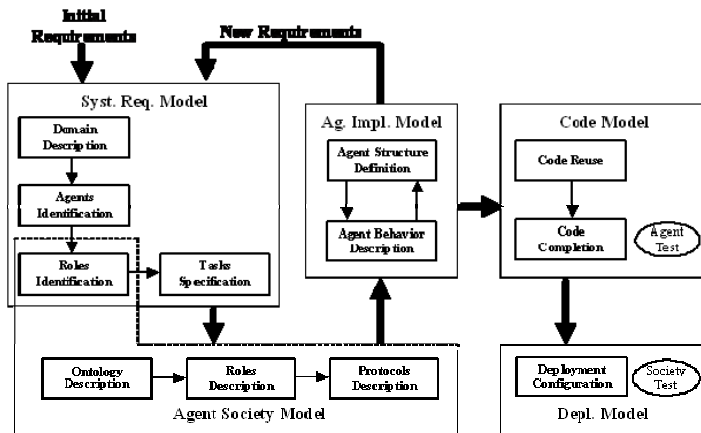
- PASSI process supports:
  - Modelling of requirements is based on use-cases
  - Ontology that as a central role in the social model
  - Multiple perspectives: agents are modelled from the social and internal point of view, both structurally and dynamically
  - Reuse of existing portions of design code; this is performed through a pattern-based approach
  - Design of real-time systems
  - The design process is incremental and iterative
- Extends UML with the MAS concepts



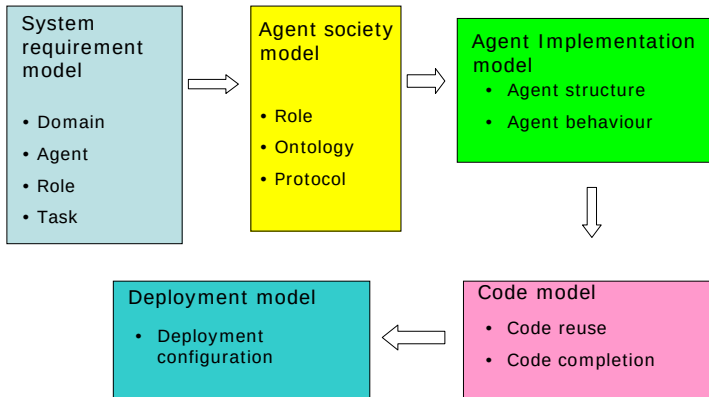
# PASSI: Meta-model



# PASSI: Models Relation



# PASSI: Abstractions



# The System Requirement Model

- It is composed of four phase:
  - *Domain Requirements Description*: a functional description on the system made by using conventional use case diagrams
  - *Agent Identification*: the phase of attribution of responsibilities to agent, represented as a stereotyped UML packages
  - *Role Identification*: a series of sequence diagrams exploring the responsibilities of each agent through role-specific scenarios
  - *Task Specification*: specification of the capabilities of each agent with activity diagrams



# Agent Societies Model

- A model of the social interactions and dependencies among the agents involved in the solution. Developing this model involves three step:
  - *Ontology Description*: use of class diagrams and OCL constraints to describe the knowledge ascribe to individual agents and their communications
  - *Role Description*: class diagrams are used to show the roles played by agent, the task involved, communication capabilities and inter-agent dependency
  - *Protocol Description*: use of sequence diagrams to specify the grammar of each pragmatic communication protocol in terms of speech-act performative



# Agent Implementation Model

- A classical model of the solution architecture in terms of classes and methods; the most important differences with common object-oriented approach is that we have two different levels of abstraction, the social (multi-agent) level and the single level. This model is composed by:
  - *Agent Structure Definition*: conventional class diagrams describe the structure of solution agent classes
  - *Agent Behaviour Description*: activity diagrams or state charts describe the behaviour of individual agent



# Code Model

- A model of the solution at the code level requiring the following steps to produce it:
  - Generation of code from the model using one of the functionalities of the PASSI add-in
  - It is possible to generate not only the skeletons but also largely reusable parts of the method's implementation based on a library of reused patterns and associated design description
  - Manual completion of the source code



# PASSI Patterns

- We consider a pattern of agent as composed of its design level description and the corresponding JAVA code
- More in detail each patter is composed of:
  - A structure
    - Usually a base agent class and a set of task/behaviour classes
    - Described using UML class diagrams
  - A behaviour
    - Expressed by the agent using its structural elements
    - Detailed in UML dynamic diagrams (activity / state chart)
  - A portion of code
    - Some lines of code implementing the structure and the behaviour described in the previous diagram



# Deployment Model

- A model of the distribution of the parts of the system across hardware processing units and their migration between processing units. It involves one step
  - *Development configuration*: deployment diagrams describe the allocation of agents to the available processing units and any constraints on migration and mobility



# Test

- The testing activity has been divided in two different steps
  - *The Single Agent Test*: is devoted to verifying the behaviour of each agent regarding the original requirements for the system solved by specific agent
  - During the *Society Test*, integration verification is carried out together with the validation of the overall results of this iteration
  - The *Single Agent Test* is performed on the single agent before the deployment phase, while the *Society Test* is carried out on the complete system after its deployment.



# Limitations

- *Multiplicity problem* (from UML): the need to concurrently refer to different models in order to understand a system and the way it operates and changes over time is a critical issue
- (From UML) Each model introduces its own set of symbols and concepts, thus leading to an unnatural complexity in terms of vocabulary.
- The environment is not considered
- ...the support to manage complexity?



# Outline

- 1 What is Agent-Oriented Software Engineering (AOSE)?
- 2 Survey on AOSE methodologies
  - Introduction to Methodologies
  - Method Engineering and Agent
  - Gaia
  - PASSI
  - Tropos
  - ADELFE



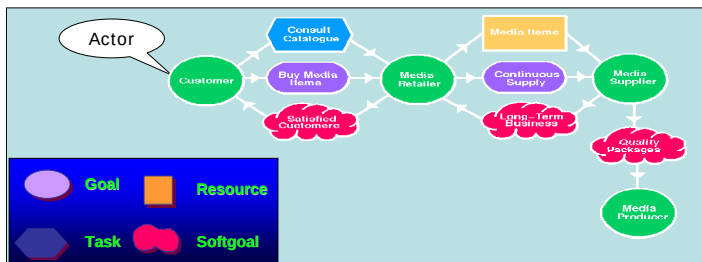
# Characteristics of Tropos

- Tropos is an agent-oriented software development methodology founded on two key features
  - (i) the notions of agent, goal, plan and various other knowledge level concepts are fundamental primitives used uniformly throughout the software development process
  - (ii) a crucial role is assigned to requirements analysis and specification when the system-to-be analysed with respect to its intended environment.
- Then the developers can capture and analyse the goals of stakeholders
- These goals play a crucial role in defining the requirements for the new system: prescriptive requirements capture the what and the how for the system-to-be

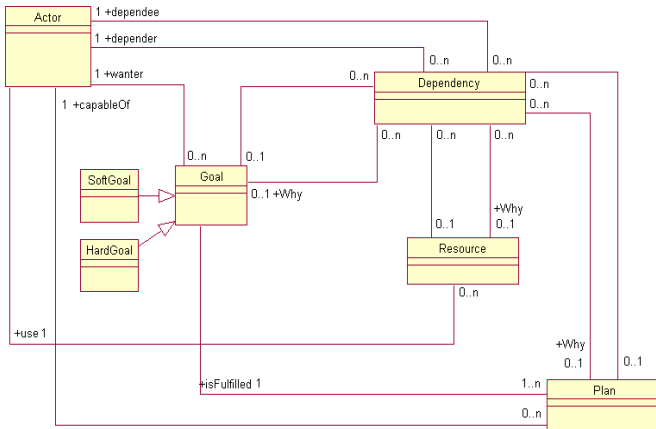


# Characteristics of Tropos

- Tropos adopts Eric Yu's i\* model which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modelling an application during early requirements analysis



# Tropos Meta-model 1/2

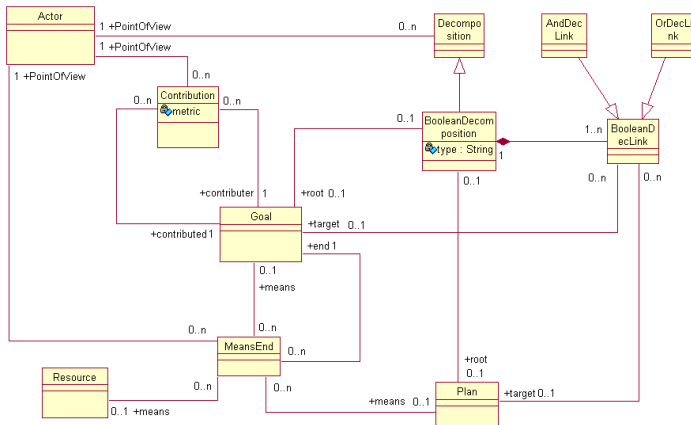


# Tropos Meta-model 1/2

- Actor: an entity that has strategic goals and intentionality
- Goal: actors' strategic interests
- Resource: a physical or an informational entity
- Plan: a way of doing something
- Dependency: depender – dependum – dependee



# Tropos Meta-model 2/2

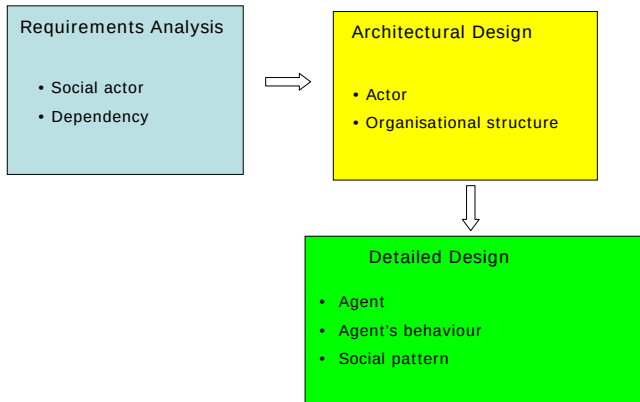


# Tropos Meta-model 2/2

- AND/OR decomposition: root(Goal)– sub(Goals)
- Contribution: towards the fulfillment of a goal
- Means-end analysis: a means to satisfy the goal



# Tropos: Abstractions



# Early Requirements Analysis

- Focuses on the intentions of stakeholders. Intentions are modelled as goals
- Through some form of goal-oriented analysis, these initial goals lead to the functional and non-functional requirements of the system
- Stakeholders are represented as (social) actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished
- Includes the Actor diagram and Rationale diagram



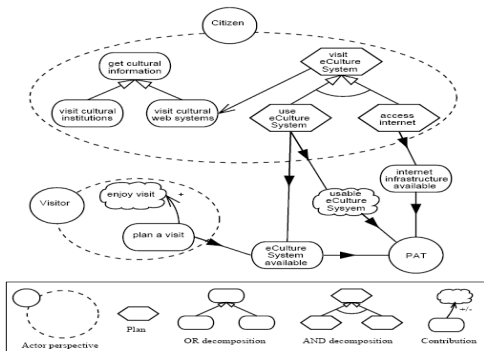
# Early Requirements Analysis

- An Actor diagram is a graph involving actors who have strategic dependencies among each other. A dependency describes an *agreement* between a depending actor (dependor) and an actor who is depended upon (dependee)
- Actor Diagrams are extended during this phase by incrementally adding more specific actor dependencies, discovered by means-end analysis of each goal. This analysis is specified using a rationale diagrams.
- Means-end analysis aims at identifying plans, resources and softgoals that provide means for achieving a goal.



# Early Requirements Analysis

- A Rationale diagram describes and supports the reasoning that each actor goes through concerning its relationships with other actors



# Late Requirements Analysis

- The conceptual model developed during early requirements is extended to include system as new actor, along with dependencies between this actor and others in its environment
- These dependencies define functional and non-functional requirements for the system-to-be.
- In Tropos, the system is represented as one or more actors which participate in a Actor diagram, along with other actors from the system's operational environment. In other words, the system comes into the picture as one or more actors who contribute to the fulfilment of stakeholder goals
- Actor and Rationale diagrams are also used in this phase



# Architectural Design

- Tropos is interested in developing a suitable set of architectural styles for multi-agent software systems: studying the Organisation Theory and Strategic Alliances leads to propose models such as the structure-in-5, the pyramid style, the chain of values, the matrix, the bidding style to try to find and formalise recurring organisational structures and behaviours
- The analysis for selecting an organisational setting that meets the requirements of the systems is based on specific propagation algorithms

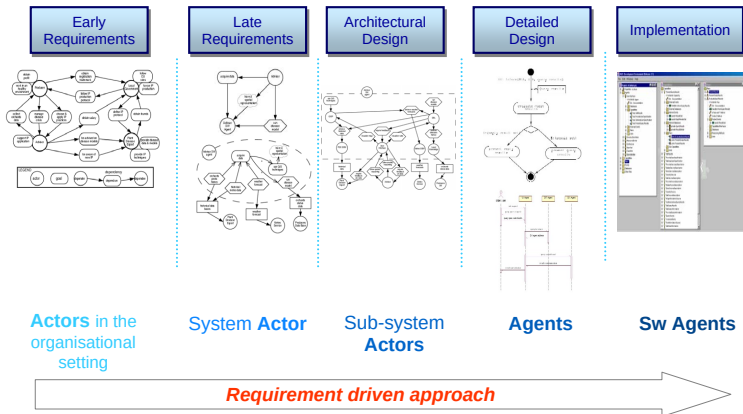


# Detailed Design

- This phase introduces additional detail for each architectural component of a system
- In particular, this phase determines how the goals assigned to each actor are fulfilled by agents in terms of design patterns
- Social Pattern in Tropos are designed patterns focusing on social and intentional aspects that are recurrent in MAS. They are classified in *Pair* and *Mediation*
  - Pair: describes direct interaction between negotiating agent (es: Bidding pattern)
  - Mediator: describes intermediary agents that help other agents to reach an agreement on an exchange of service (es: Broker pattern)



# AO Visual Modelling with Tropos



# Limitations

- Tropos is not intended for any type of software: no system with no identifiable stakeholders
- Tropos, in its current form, is not suitable for sophisticated software agents requiring advanced reasoning mechanism for planning
- ...and the environment?
- ...the support to manage complexity?



# Outline

- 1 What is Agent-Oriented Software Engineering (AOSE)?
- 2 Survey on AOSE methodologies
  - Introduction to Methodologies
  - Method Engineering and Agent
  - Gaia
  - PASSI
  - Tropos
  - ADELFE



# Characteristics of ADELFE

- ADELFE is dedicated to the design of systems that are complex, open and not well-specified (Adaptive Multi-Agent Systems)
- The primary objective of ADELFE method is to cover all the phases of a classical software design
- RUP has been tailored to take into account specificities coming from the design of AMAS
- ADELFE follows the vocabulary of SPEM
- Only the requirement, analysis and design *work definitions* require modifications in order to be adapted to AMAS, other appearing in the RUP remaining the same.
- ADELFE is supported by several Tools



# Adaptive Multi-Agent Systems Theory

- The openness and dynamics are source of *unexpected* events and an open systems plugged into a dynamic environments has to be able to adapt to these changes, to *self-organise*
- Self-organisation is a means to make the system adapt but also to overcome complexity
- If a system is complex and its algorithm unknown, it is impossible to code its *global function*
- This function has then to emerge at the macro level (system level) from the interaction at the micro level (component level)
- It is sufficient to build a system whose components have a cooperative attitude to make it realise an expected function
- Cooperation is the local criterion that enables component to find the right place within the organisation



# Adaptive Multi-Agent Systems

- Adaptive Multi-Agent Systems are composed of agents that permanently try to maintain cooperative interactions with other.
- Any *cooperative agent* in AMAS follow a specific lifecycle that consists in:
  - The agent gets perceptions from its environment
  - It autonomously uses them to decide what to do in order to reach its own goal
  - It acts to realise the action on which it has previously decided



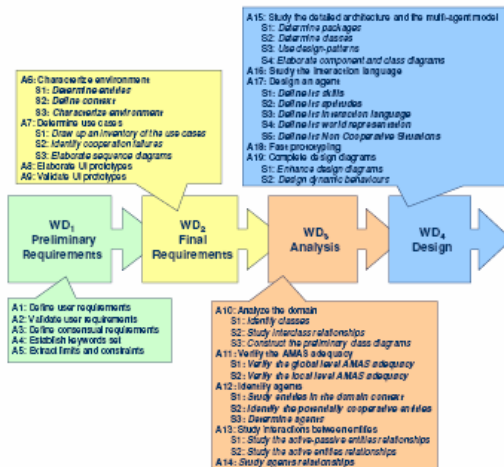
# Cooperative Agents

- The cooperative agents are equipped with *five modules* to represent “physical”, “cognitive”, or “social” capabilities:
  - *Skill Module*: represents knowledge on specific fields that enables agents to realise their partial function
  - *Representation Module*: enables an agent to create its own representation about itself, other agents, or the environment it perceive
  - *Interaction Module*: is composed of perceptions and actions.
  - *Aptitude Module*: provides capabilities to reason on perceptions, skills, and representations – for example, to interpret messages
  - *Cooperation Module*: embeds local rules to be locally *cooperative*. Cooperative means that agents are able to recognise “cooperation failures”





# ADELFE Models Relation



# Work Definition 1: Preliminary Requirements

- A1: Define user requirements
- A2: Validate user requirements
- A3: Define consensual requirements
- A4: Establish Keywords set
- A5: Extract limits and constraints



# Work Definition 2: Final Requirements

- **A6: Characterise environment**
  - S1: Determine entities
  - S2: Define context
  - S3: Characterise environment
- A7: Determine use cases
  - S1: Draw an inventory of use cases
  - **S2: Identify cooperation failure**
  - S3: Elaborate sequence diagrams
- A8: Elaborate UI Prototype
- A9: Validate UI Prototype



# Work Definition 3: Analysis

- A10: Analyse the domain
  - S1: Identify classes
  - S2: Study interclass relationship
  - S3: Construct preliminary class diagrams
- **A11: Verify the AMAS adequacy**
  - S1: Verify it at the global level
  - S2: Verify it at the local level
- **A12: Identify agents**
  - S1: Study entities in the domain context
  - S2: Identify potentially cooperative entities
  - S3: Determine agents
- A13: Study interaction between entities
  - S1: Study active/passive entities relationships
  - S2: Study active entities relationships
  - **S3: Study agent relationships**



# Work Definition 4: Design

- A14: Study detailed architecture and multi-agent model
  - S1: Determine packages
  - S2: Determine classes
  - S3: Use design-patterns
  - S4: Elaborate component and class diagrams
- **A15: Study interaction languages**



# Work Definition 4: Design

- **A16: Design agents**
  - S1: Define skills
  - S2: Define aptitude
  - S3: Define interaction languages
  - S4: Define world representations
  - S5: Define Non Cooperative Situations
- **A17: Fast prototyping**
- **A18: Complete design diagrams**
  - S1: Enhance design diagrams
  - S2: Design dynamic behaviours



## A6: Characterise environment

- S1: The characterisation begins by identifying the entities that interact with systems and constraints on these interactions
  - *Active entity* may behave autonomously and is able to act dynamical way with the systems
  - *Passive entity* can be considered as a resource by the systems; it may be used or modified by active ones but cannot change in autonomous way.
- S2: The context is studied through the interactions between entities in the system.
- S3: The designer must describe the environment with terms inspired by Russel and Norving
  - The environment could be accessible/inaccessible, continuos/discret, determinist/non deterministic, dynamic/static



## A7-S2: Identify Cooperation Failure

- The designer must begin to think about the events that can be “unexpected” or “harmfull” for the system
- These situation can lead to *Non Cooperative Situations* at the agent level
- These *cooperation failures* can be viewed as a kind of exception
- To take into account this aspects, the determination of use cases has been modified adding a step in which cooperation failures must be highlighted within the previously identified use case.



# A11: Verify the AMAS Adequacy

- Not every designer needs AMAS theory to build a system.
- The adequacy is studied by means of *AMAS Adequacy tool* at two levels, through a certain number of criteria
  - At the global level, to answer the question “is a system implementation using AMAS needed?”
  - At the local level, to try to answer the question “do some components need to be implemented as AMAS?” That is, is some decomposition or recursion useful during design?
- Is a positive answer is given by the tool in the former case, the designer can continue applying the process.



# A12: Identify Agents

- In ADELFE agents are not considered as being known in advance
- S1: the designer must identify them in a new activity in which the previously identified entities will be studied and evaluated
  - If an entity shows some specific properties, it may be a potential cooperative entity
  - Indeed , this does not concern all active entities, some of them will remain simple object without becoming agents.



# A12: Identify Agents

- S2: The designer, studying the interaction with environment and with other entities, has then to determine if entity may encounter cooperation failure that will be considered “Non Cooperative Situations”
- S3: The entities verifying all these criteria will be identified as agents and the classes related to them marked with specific stereotype



## A13-S3: Study Agent Relationships

- Studying relationship between passive and active entities or between solely active ones is done by using UML collaboration or sequence diagrams in a standard way
- In this step AUML protocol diagrams are used to express relations between all existing agents.



# A15: Study Interaction Languages

- The designer has then to study interaction languages to define the way in which agents interact
- If agents interact in order to communicate, information exchanged between agents must be described using AUML protocol diagrams
- Languages that enable the interactions between agents may be implemented by a set of classes by a design pattern, including an implementation of FIPA-ACL.



# A16: Design Agents

- The designer can refine the “cooperative agents” classes he has previously defined.
- The different modules composing an agent must be given in this activity, as well as their physical properties



# A17: Fast Prototyping

- Once the behaviour of the agents involved in the concerned AMAS is defined, the simulation functionality of OpenTool enables the designer to test them.
- The functionality of OpenTool requires a dynamic model (state-chart) for each simulated entity.
- The customised version of OpenTool is able to automatically transform a protocol diagram into state-chart.



# Limitations

- ADELFE is specialised and therefore cannot be used to design all the existing applications or to model all type of agents
- If the Verify Adequacy fails the design process is stopped. This limit could be also removed by coupling another methodology with ADELFE.
- No operational tool such as platform or a set of software tools is coupled to ADELFE to guide implementation, testing or deployment
- ... the concept of role?
- ... the support to manage complexity?



# Introduction to Agents and Multi-agent Systems I



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 and N. Jennings

Intelligent Agents: Theory and Practice

The Knowledge Engineering Review, Vol. 10, No. 2, 1999.



# Introduction to Agents and Multi-agent Systems II



D. Chess and C. Harrison and A. Kershenbaum

Mobile Agents: are They a Good Idea?

Mobile Object Systems, LNCS, No. 122 2 , Springer, 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. Jennings

An Agent-Based Approach for Building Complex Software System

Communications of the ACM, 44(4):35:41, 2001.



# Surveys on Methodologies and Meta-Model I



C. Bernon and M. Cossentino and M.P. Gleizes and P. Turci  
and F. Zambonelli

A Study of Some Multi-agent Meta-models

Agent-Oriented Software Engineering V. LNCS 3382

Springer 2004



C. Iglesias and M. Garijo and J.C. Gonzales

A Survey of Agent-oriented Methodologies

Intelligent Agents V, LNAI No. 1555, 1999.



O. Shehory and A. Sturm

Evaluation of Modeling Techniques for Agent- Based Systems

Proceedings of The V International Conference on

Autonomous Agents, pp. 624-631, 2001.



# Surveys on Methodologies and Meta-Model II



M. Wooldridge and P. Ciancarini

Agent-Oriented Software Engineering

Agent-Oriented Software Engineering, LNCS No. 1957, 2001.



B. Henderson-Sellers and P. Giorgini

Agent-Oriented Methodologies

Idea Group Inc, Hersey, PA, USA, 2005.



F. Bergenti and M.P. Gleizes and F. Zambonelli

Methodologies and Software Engineering for Agent Systems

Kluwer Accademic Publisher, 2004.



M. Luck and R. Ashiri and M. D'Inverno

Agent-Based Software Development

Artech House Publisher, 2004.



# The Gaia Methodology I



F. Zambonelli and N. Jennings and M. Wooldridge  
Multi-Agent Systems as Computational Organisations: The  
Gaia Methodology

in Agent-Oriented Methodologies

B. Henderson-Sellers and P.Giorgini, Idea Group Inc, Hersey,  
PA, USA, 2005.



F. Zambonelli and N. Jennings and M. Wooldridge  
Developing Multi-Agent Systems : The Gaia Methodology  
ACM Transaction on Software Engineering and Methodology,  
12(3), pp. 417–470.



# The PASSI Methodology I



M. Cossentino

From Requirements to Code with the PASSI Methodology  
in Agent-Oriented Methodologies

B. Henderson-Sellers and P.Giorgini, Idea Group Inc, Hersey,  
PA, USA, 2005.



A. Chella and M. Cossentino and L. Sabatucci and V. Seidita

From PASSI to Agile PASSI: Tailoring a Design Process to  
Meet new Needs

IEEE/WIC/ACM international Joint Conference on Intelligent  
Agent Technology, Beijing, China, 2004



# The Tropos Methodology I



P. Giorgini and M. Kolp and J. Mylopoulos and J. Castro  
Tropos: a Requirements-Driven Methodology for  
Agent-Oriented Software

in Agent-Oriented Methodologies

B. Henderson-Sellers and P.Giorgini, Idea Group Inc, Hersey,  
PA, USA, 2005.



P. Bresciani and P. Giorgini and F. Giunchiglia and  
J. Mylopoulos and A. Perini

Tropos: An Agent-Oriented Software Development  
Methodology

Autonomous Agent and Multi-Agent Systems. Vol 8 pp.  
203–236 issn: 1387-2532



# The ADELFE Methodology I



C. Bernon and V. Camps and M.P. Gleizes and G. Picard  
Engineering Adaptive Multi-Agent Systems: The ADELFE  
Methodology

in Agent-Oriented Methodologies

B. Henderson-Sellers and P.Giorgini, Idea Group Inc, Hersey,  
PA, USA, 2005.



C. Bernon and M.P. Gleizes and S. Peyruqueou and G. Picard  
ADELFE, a Methodology for Adaptive Multi-Agent Systems  
Engineering

P. Petta, R. Tolksdorf, and F. Zambonelli, editors,

Third International Workshop on Engineering Societies in the  
Agents World (ESAW-2002).



# Other Relevant Methodologies I



S.A. DeLoach and M.F. Wood and Cl.H. Sparkman  
Multiagent Systems Engineering

The International Journal of Software Engineering and  
Knowledge Engineering, Vol. 11 (3) pp. 231-258, 2001.



G. Caire and F. Leal and P. Chainho and R. Evans and  
F. Garijo and J. Gomez and J. Pavon and P. Kearney and  
J. Stark and P. Massonet

Agent Oriented Analysis using MESSAGE/UML

II International Workshop on AOSE, LNCS 2222,  
Springer-Verlag , pp. 119-135, 2001.



## Other Relevant Methodologies II



C.A. Iglesias and M. Garrijo and J. Gonzalez and J.R. Velasco  
Analysis and Design of Multiagent Systems using  
MAS-CommonKADS

Intelligent Agents IV: Agent Theories, Architectures and  
Languages, M. P. Singh, Anand Rao and M. J. Wooldridge,  
eds., LNCS 1365, Springer-Verlag , pp. 313-328, 1997



B. Bauer and J.P. Muller and 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.



## Other Relevant Methodologies III



F.M.T. Brazier and B. Dunin-Keplicz and N. Jennings and J. Treur

DESIRE: Modelling Multi- Agent Systems in a Compositional Formal Framework

International Journal of Cooperative Information Systems, Vol. 6, pp. 67-94, 1997.



# Open Research Directions & Visions I



F. Zambonelli and V. Parunak

From Design to Intention: Signs of a Revolution

1st Joint Conference on Autonomous Agents and Multi-agent Systems, Bologna (I), July 2002



V. Parunak and S. Bruekner

Entropy and Self-Organization in Agent Systems

5th International Conference on Autonomous Agents , ACM Press, May 2001.



R. Albert and H. Jeong and A. Barabasi

Error and Attack Tolerance of Complex Networks

Nature, 406:378-382, 27 July 2000.



# Open Research Directions & Visions II



G.D. Abowd and E.D. Mynatt

Charting Past, Present and Future Research in Ubiquitous Computing

ACM Transactions on Computer-Human Interaction,  
7(1):29-58, March 2000.



H. Abelson and D. Allen and D. Coore and C. Hanson and  
G. Homsy and T. Knight and R. Nagpal and E. Rauch and  
G. Sussman and R. Weiss

Amorphous Computing

Communications of the ACM, 43(5), May 2000.



## Open Research Directions & Visions III



M. Mamei and F. Zambonelli L. Leonardi

A Physically Grounded Approach to Coordinate Movements in a Team

1st International Workshop on Mobile Teamwork, Vienna, IEEE CS Press, July 2002.



A. Roli and F. Zambonelli

What Can Cellular Automata Tell Us About the Behavior of Large Multiagent Systems?

1st International Workshop on Engineering of Large Multiagent Systems, Orlando (FL), May 2002,



# Open Research Directions & Visions IV



F. Zambonelli and A. Omicini

Challenges and Research Directions in Agent-Oriented  
Software Engineering

Autonomous Agents and Multi-Agent Systems 9(3). Kluwer  
Academic Publishers, November 2004.



# Agent-Oriented Software Engineering

Multiagent Systems LS

Sistemi Multiagente LS

Ambra Molesini

`ambra.molesini@unibo.it`

Ingegneria Due

ALMA MATER STUDIORUM—Università di Bologna a Cesena

Academic Year 2006/2007

