Agent-Oriented Software Engineering

Ambra Molesini
Cesena - 19 Aprile 2006

Email: ambra.molesini@unibo.it
amolesini@deis.unibo.it

Outline

- Part 1: What is Agent-Oriented Software Engineering (AOSE)
- Part 2: Survey on AOSE methodologies
- Part 3: The SODA Methodology

Part 1

Agent-Oriented Software Engineering

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.

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!

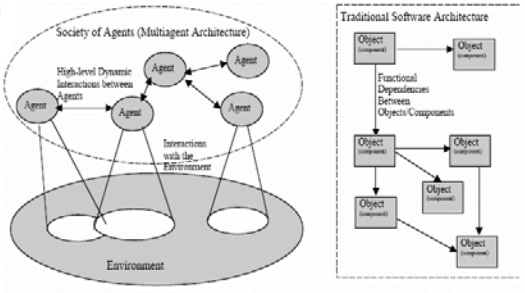
Weak Viewpoint

- Remember that….
- 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 agents-based one, can be indeed characterised in terms of weak multi-agent systems
- Let's better characterize the SE perspective on agents…

MAS vs. OOSE



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
  - Interaction protocols as an additional SE dimension
- Openness
  - Controlling self-interested agents, malicious behaviours, and badly programmed agents
  - Dynamic re-organization of software architecture
- Mobility and Locality
  - Additional dimension of autonomous behaviour
  - Improve locality in interactions

Agent-Oriented Abstractions

- The development of a multi-agent 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
- Necessity of mechanisms to *manage the complexity of system description*

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 have characteristics that are very similar to those of agents and multi-agent 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 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

Summarizing

- A software engineering paradigm defines:
  - The mindset, the set of abstractions to be used in software development and, consequently,
  - Methodologies
  - 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 rage of distributed computing applications….

Part 2

Survey on AOSE Methodology

Outline

- What is a Methodology?
- Methodology overview
  - Gaia
  - PASSI
  - Tropos

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

From: "Fundamental of Software Engineering". Prentice Hall International

What is an AO methodology?

- AOSE methodologies mainly try to suggest a clean and disciplined approach to analyze, 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 organize 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]

[1] Bernon at. all "A study of some multi-agent meta-models" Agent-Oriented Software Engineering V. Volume 3382 of LNCS, Springer (2004) 62–77

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 al.)
  - PASSI (Cossentino)
  - SODA (Omicini, Molesini)
  - Prometheus (Winokoff and Pagdam)
  - Etc.
- Exploiting abstractions that made them more suited to specific scenarios or to others..
- In this part we show Gaia, PASSI and Tropos
- In part 3 we focus on SODA

Ok, I am not an impartial judge…

Gaia

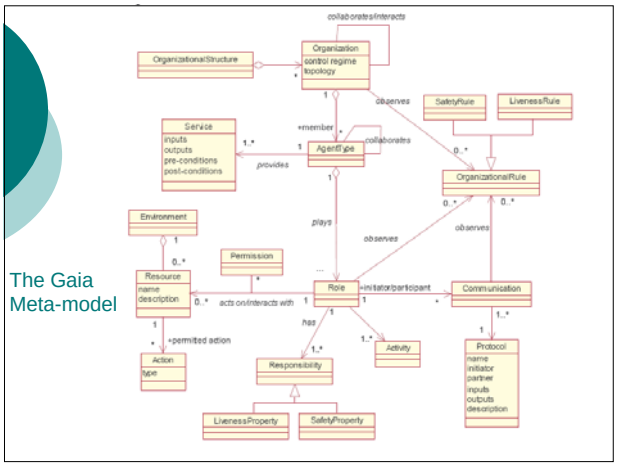
Zambonelli, Jennings and Wooldridge

The Gaia Methodology

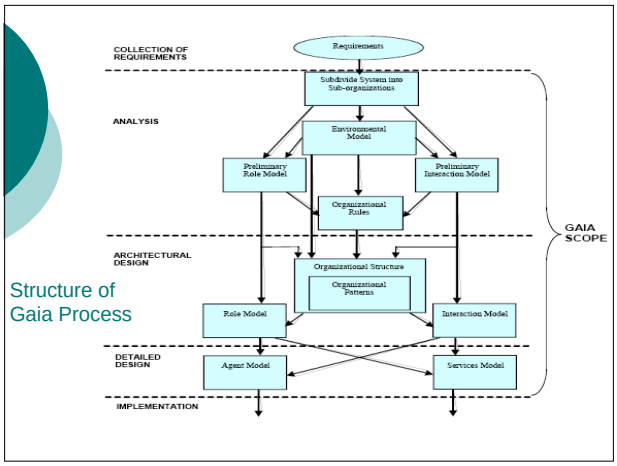
- It is the most known AOSE methodology
  - Firstly proposed by Jennings and Wooldridge in 1999
  - Extended and modified by Zambonelli in 2000
  - Final Stable Version in 2003 by Zambonelli, Jennings, Wooldridge
  - Many other researchers are working towards further extensions…
- Key Goals
  - Starting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

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



The Gaia Meta-model



Structure of Gaia Process

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

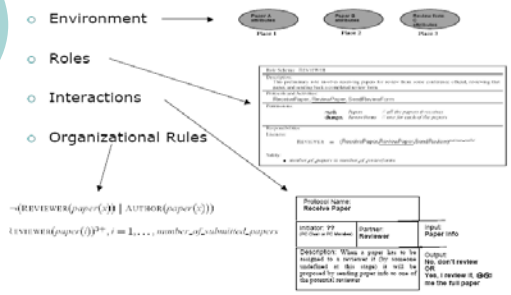
- Preliminary Role Model
  - See what “roles” must be played in the organisation
  - A role defines a “responsibility” centre in the organisation with a set of expected behaviours (permissions and responsibilities)
- Preliminary Interaction Model
  - See how roles must interact with each other so as 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



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 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 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?

PASSI

Cossentino

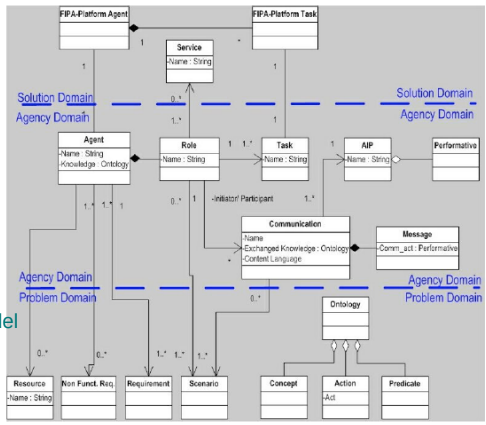
Characteristic of PASSI

- *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

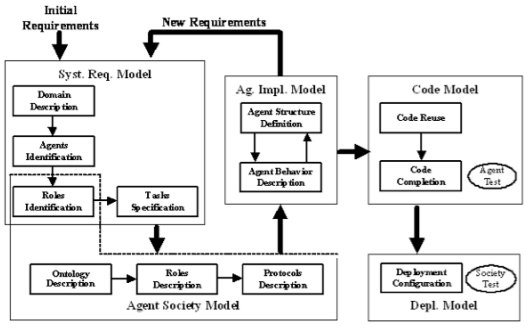
Characteristic of PASSI

- *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 Process Overview



The System Requirement Model

- It is composed of four phase
- *Domain Requirements Description*: a functional description on the system 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 performatives

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

- PASSI considers 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?

Tropos

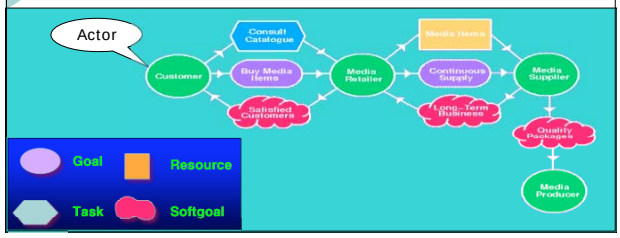
Giogini et all.

Characteristic 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 analyzed with respect to its intended environment.
- Then the developers can *capture and analyze* 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

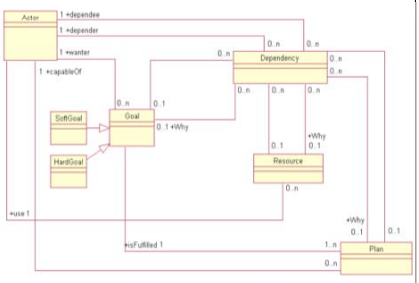
Characteristic of Tropos

- 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



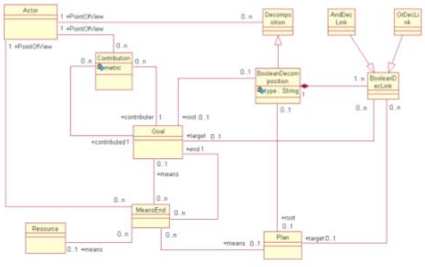
Tropos Meta-model 1/2

- **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

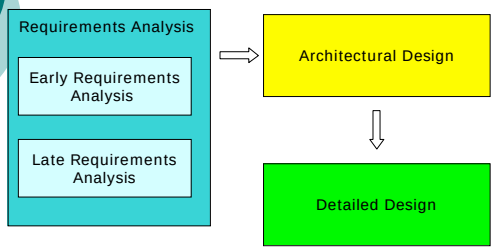


Tropos Meta-model 2/2

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



Tropos



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 (depender) 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 an 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

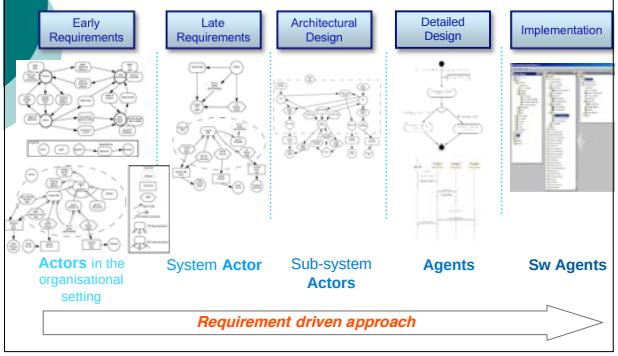
Architectural Design

- Tropos is interested in developing a suitable set of architectural styles for multi-agent software systems: studying the Organization 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 sophisticate software agents requiring advanced reasoning mechanism for planning
- … and the environment?
- … the support to manage complexity?

Part 3

SODA

Outline

- Introduction to SODA
- Agents & Artifacts
- Layering Principle
- SODA in detail

Introduction to SODA

SODA

- SODA (Societies in Open and Distributed Agent spaces) is an agent-oriented methodology for the analysis and design of agent-based systems
- SODA focuses on inter-agent issues, like the engineering of societies and environments for MASs
- SODA adopts *agents* and *artifacts* as building block for MAS development

SODA

- SODA introduces a simple layering principle in order to manage the complexity of the system description
- SODA adopts a tabular representations

Agents & Artifacts

Remember that…

- Artifacts take the form of objects or tools that agents *share* and *use* to
  - support their activities
  - achieve their objectives
- Artifacts are explicitly designed to provide some *functions* which guide their use.
- An artifact can have *responsibilities*

Example

- Coordination Artefacts
  - govern social activities
  - enable and mediate agent interaction
  - mediate the interaction between individual agents and their environment
  - capture, express and embody the parts of the environment that support agents' activities

Features & Classification

- An artifact exposes
  - usage interface
  - operating instructions
  - function description
- Other interesting artifact features are:
  - inspectability
  - malleability
  - linkability
- A possible classification
  - individual artifacts
  - social artifacts
  - resource artifacts

Agents & Artifacts

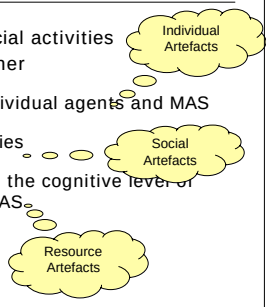
- Artifacts constitute the basic building block both for
  - MAS analysis/modelling
  - MAS development
- Agents and Artefacts can be assumed as two fundamental abstractions for modelling MAS structure
  - agents speaking with other agents
  - agents using artifacts to achieve their objectives

Meta-model Ingredients

- Agents & Artifacts lead to new *ontological meta-model* for MASs
- Artifacts allow to
  - model the environment as a first-class entity
  - engineer the space of interaction among agents (not only mere conversations between agents, but complex agent interaction patterns)
  - enrich MAS design with social/organisational structure, topological models, as well as (complex) security models

In particular in SODA…

- Agents model individual/social activities
- Artifacts **glue** agents together
  - they mediate between individual agents and MAS
  - they build up agent societies
  - they wrap up and bring to the cognitive level of agents the resources of MAS



Layering Principle

Complex systems and layering

- As advocate in the *theory of hierarchies* all complex systems are amenable to be represented as organised on different layers
- Each level is essential to the general understanding of the system's wholeness, and is autonomous with its own laws, patterns and behaviour
- At the same time, no level can be understood in isolation independently of all the other levels, and the system as a whole can be understood only through the understanding and representation of all its levels
- A complex system is a system requiring layer, independent but strongly correlated ones, in order to fully understand and reproduce its dynamics and behaviour. (e.g., biological systems)

Layering and MASs

- When applied to the engineering of MASs, the *hierarchy principle* suggests that agent-oriented processes and methods should support some forms of MAS layering
- Allowing engineers to design and develop MAS along different levels of abstractions
  - a number of independent, but strictly related, MAS layers

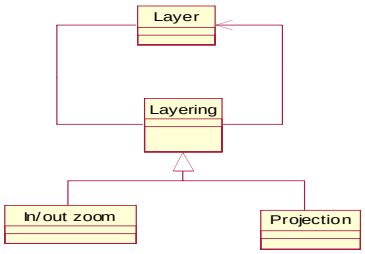
Layering in SODA

- We achieve the layering principle by means of the zooming and projection mechanisms.
- In the zooming mechanism we have two kinds of zoom
  - in-zoom: when passing from abstract layer to another more detailed
  - out-zoom: when passing from detailed layer to another more abstract.
- The *projection mechanism* projects the no zoomed entities form one layer to another to achieve the internal consistency of one layer

System's view

- It is possible to have two type of *system's view*
  - Horizontal views allow to analyse the system in one level of detail.
  - Vertical view allows to analyse of one kind of abstract entity in its whole layer from layer.

Meta-Model of Layering



Layering in SODA

- In general, when working with SODA, we start from a certain layer, we could call *core layer*, and it is labelled with "c"
- In the other layer we find only the in/out zoomed entities and the projection entities.
- The in-zoomed layers are labelled with "c+1", "c+2" and the out-zoomed layers are labelled "c-1", "c-2". .
- The projection entities will be labelled with "+" if the projection is form abstract layer to detailed layer, "-" otherwise

Important

- The only relations between layers are the "zooming relation" express by means of *zooming table (in the following)*
- if we have relation between entities belonging different layers we have to project these entities in the same layer

SODA

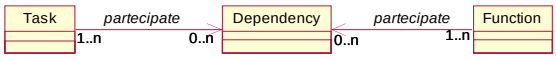
Omicini, Molesini

SODA

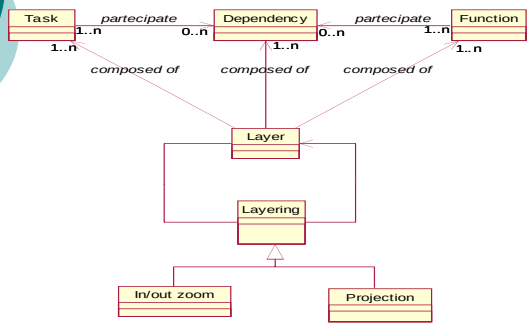
- SODA is organised in three phase:
  - *Requirements analysis phase*: the system's requirements are analysed and modelled in terms of tasks, functions and dependencies
  - *Analysis phase*: in this phase we analyse the solution domain, the system is modelled in terms of roles, resources, interactions and constraints
  - *Design phase*: in this phase we design the system in terms of agents, societies and artifacts

First Meta-model of Requirements Analysis Phase

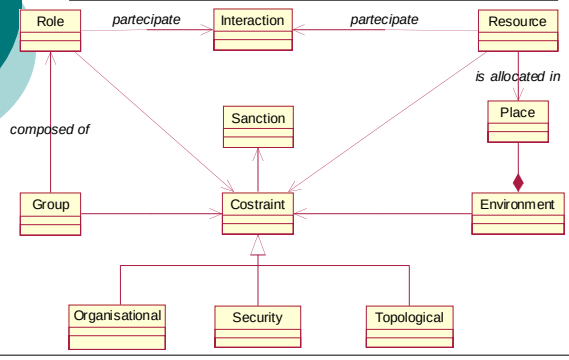
- *Task* is an activity that requires one or more competences and the use of functions
- *Function* is an reactive activity that aimed at supporting tasks
- *Dependency* is any relationship (interactions, constraints. . .) among other (tasks and/or functions) abstract entities



Meta-Model of Requirements Analysis Phase



First Meta-Model of Analysis Phase

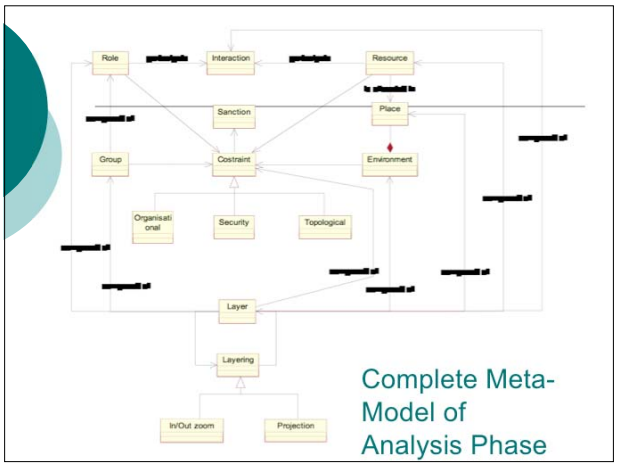


Entities in the Meta-model of Analysis Phase

- *Role* is defined as the abstraction responsible for the achievement of one or more tasks.
- *Group* is defined as the abstraction responsible for a collection of roles. It derives from the zoom of a role. To preserve the consistency in the group it is necessary to introduce social rules.
- *Resource* is defined as the abstraction that provides some functions.
- *Interaction* is defines as a relation that aimed to exchange some information (of any type) among abstract entities. It is represented by means of interaction protocols

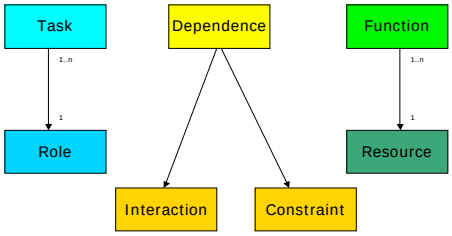
Entities in the Meta-model of Analysis Phase

- *Constraint* is defined as a relation among abstract entities that aimed to bound the abstract entities. For example constraint can be organisational constraints, topological constraints, security constraints.
- *Sanction* is a punishment of constraint violation
- *Environment* is the environment of the system.
- *Place* is a conceptual locus in the environment.

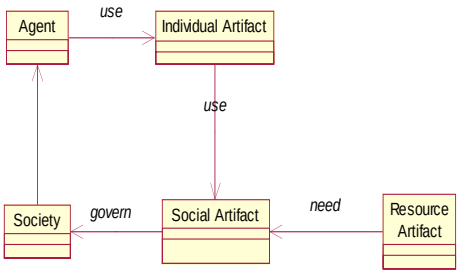


Complete Meta-Model of Analysis Phase

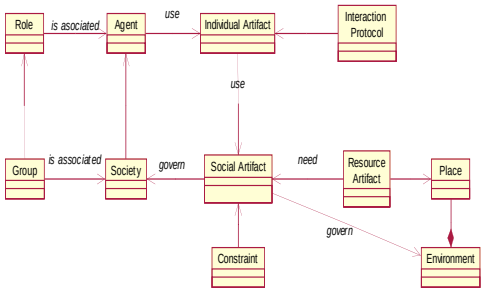
From Requirements Analysis to Analysis

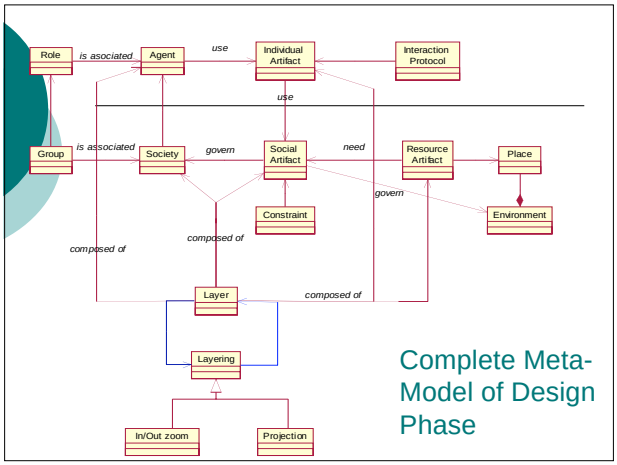


Entities of Design Phase



First Meta-Model of Design Phase





Complete Meta-Model of Design Phase

Tables of Requirements Analysis

- Responsibility Tables

Task	Description
task name	*task description*

$(L)T_t$

Function	Description
function name	*function description*

$(L)F_t$

Tables of Requirements Analysis

- Dependency Table

Dependency	Description
dependency name	*dependency description*

$(L)D_t$

Tables of Requirements Analysis

- Link Tables

Task	Dependency
task name	*dependency names*

$(L)TD_t$

Function	Dependency
function name	*dependency names*

$(L)FD_t$

Zooming Table

Layer L	Layer L+1
out-zoom entity	*in-zoom entities*

$(L)Z_t$

Tables of Analysis and Design phases

Sorry, currently…

SODA: Pros and Cons

- Pros
  - Design societies
  - Design environments
  - Layering principle
- Cons
  - Only Inter-agents aspects → need to other methodology for design intra-agents aspects

Available Thesis

- Tecnologie ad agenti per una casa intelligente
- Realizzazione di un tool a supporto della metodologia SODA
- …

From: http://www.alice.unibo.it

Addendum

Select References

Selected References

- **Introductory to Agents and Multiagent Systems**

– A. Newell, "The Knowledge Level", Artificial Intelligence, 18(1):87-127, 1982.
– P. Wegner, "Why Interaction is More Powerful than Algorithms", Communications of the ACM, 40(5):80–91, 1997.
– M. Wooldridge, "Reasoning About Rational Agents", MIT Press, 2000.
– M. Wooldridge, N. Jennings, "Intelligent Agents: Theory and Practice", The Knowledge Engineering Review, Vol. 10, No. 2, 1999.
– D. Chess, C. Harrison, A. Kershenbaum , "Mobile Agents: are They a Good Idea?", Mobile Object Systems, Lecture Notes in Computer Science, No. 122 2 , Springer-Verlag (D), pp. 25-45, February 1997.
– V. Parunak, "Go to the Ant: Engineering Principles from Natural Agent Systems ", Annals of Operations Research, 75:69-101, 1997.
– N. R. Jennings, "An Agent-Based Approach for Building Complex Software System", *Communications of the ACM*, 44(4):35:41, 2001.

Selected References

- **Agent Abstractions**

– F. Zambonelli, V. Parunak, "From Design to Intention: Signs of a Revolution" , 1st Joint Conference on Autonomous Agents and Multi-agent Systems, Bologna (I), July 2002.
– A. Howard, M. J. Mataric, "Cover Me! A Self-Deployment Algorithm for Mobile Sensor Networks", International Conference on Robotics and Automation, 2002, to appear.
– B. A. Huberman, T. Hogg, "The Emergence of Computational Ecologies", in *Lectures in Complex Systems* , Addison-Wesley, 1993.
– D. Estrin, D. Culler , K. Pister, G. Sukhatme , "Connecting the Physical World with Pervasive Networks", IEEE Pervasive Computing, vol. 1(1):59-69,Jan 2002.
– M. Ripeani, A. Iamnitchi , I. Foster, " Mapping the Gnutella Network", *IEEE Internet Computing*, 6(1):50-57, Jan.-Feb. 2002.
– M. Sipper. "The Emergence of Cellular Computing" , *IEEE Computer*, 37(7):18-26, July 1999.
– D.Tennenhouse, "Proactive Computing", *Communications of the ACM*, 43(5):43-50, May 2000.
– F. Zambonelli, A. Roli, S. Gatti , "What Can Cellular Automata Tell Us About the Behaviour of Large Multiagent Systems?, 1st International Workshop on Large Multi- Agent Systems, Orlando (FL), May 2002.

Selected References

- **Introductory to AOSE**

– N.R. Jennings, "On Agent-Based Software Engineering", Artificial Intelligence, 117:227-296, 2000.
– N. R. Jennings, P. Faratin , T. J. Norman, P. O'Brien , B. Odgers, "Autonomous Agents for Business Process Management" , Int. Journal of Applied AI, Vol. 14 (2), pp. 145-189, 2000.
– M. J.Wooldridge and N. R. Jennings, "Software Engineering with Agents: Pitfalls and Pratfalls", IEEE Internet Computing, Vol.3, No. 3, May-June 1999.
– Y. Shoham, "An Overview of Agent-Oriented Programming", in J. M. Bradshaw, editor, Software Agents, pages 271–290. AAAI Press / The MIT Press, 1997.
– K. Siau and M. Rossi, "Evaluation of Information Modelling Methods – A Review", Proceeding 31st Annual Hawaii International Conference on System Sciences,pp. 314-322, 1998.
– F. Zambonelli, N. Jennings, M. Wooldridge, "Organizational Abstractions for the Analysis and Design" , 1st International Workshop on Agent-oriented Software Engineering, LNAI No. 1957, Springer , 2001.
– F. Zambonelli, N, Jennings, A. Omicini, M. Wooldridge, "Agent-Oriented software Engineering for Internet Applications", Coordination of Internet Agents: Models, Technologies, and Applications, Chapter 13. Springer-Verlag, March 2001.

Selected References

- **Surveys on Methodologies**

– C. Iglesias, M. Garijo , J. C. Gonzales, "A Survey of Agent-oriented Methodologies", Intelligent Agents V, LNAI No. 1555, 1999.
– M. Wooldridge , P. Ciancarini, "Agent-Oriented Software Engineering", in Agent-Oriented Software Engineering, LNCS No. 1957, 2001.
– O. Shehory and A. Sturm, "Evaluation of Modeling Techniques for Agent- Based Systems ", Proceedings of The Fifth International Conference on Autonomous Agents, pp. 624-631, 2001.

- **Book on Methodologies**

– B. Henderson-Sellers, P. Giorgini, "Agent-Oriented Methodologies", Idea Group Inc, Hersey, PA, USA, 2005.
– F. Bergenti, M-P. Gleizes, F.Zambonelli, "Methodologies and Software Engineering for Agent Systems", Kluwer Accademic Publisher, 2004.
– M. Luck, R. Ashri, M. D'Inverno, "Agent-Based Software Development ", Artech House Publisher, 2004.

Selected References

- **The GAIA Methodology**

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

- **The PASSI Methodology**

– 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, M.Cossentino, L.Sabatucci, V. Seidita, "From PASSI to Agile PASSI: Tailoring a Design Process to Meet new Needs", in 2004 IEEE/WIC/ACM international Joint Conference on Intelligent Agent Technology, Beijing, China

Selected References

- **The TROPOS Methodology**
  - P. Giorgini, M. Kolp, J. Mylopoulos, 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, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini , "Tropos: An Agent-Oriented Software Development Methodology", Autonomous Agent and Multi-Agent Systems. Vol 8 pp. 203—236 issn: 1387-2532
- **The SODA Methodology**
  - A. Omicini, "SODA: Societies and Infrastructures in the analysis and deisgn of agent-based systems ", Agent-Oriented Software Engineering, LNCS 1957, Springer-Verlag, 2001.
  - A. Molesini, A. Omicini, A. Ricci, E. Denti, "Zooming Multi-Agent Systems", Agent-Oriented Software Engineering VI, LNCS 3950, Springer-Verlag, 2006.
  - A. Molesini, A. Omicini, E. Denti, A. Ricci, "SODA: a Roadmap to Artifacts", Engineering Societies in the Agents World VI, LNCS 3963, Springer-Verlag, 2006.
  - A. Molesini, E. Denti, A. Omicini, "MAS Meta-model on test: UML vs. OPM in the SODA case study", Multi-Agent Systems and Applications IV, LNAI 3690, Springer-Verlag, 2005.

Selected References

- **Other Relevant Methodologies:**
  - ***MASE***
    - S. A. DeLoach, M. F. Wood, Cl. H . Sparkman , "Multiagent Systems Engineering", The International Journal of Software Engineering and Knowledge Engineering, Vol. 11 (3) pp. 231-258, 2001.
  - ***MESSAGE***
    - G. Caire , F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, J. Pavon , P. Kearney, J. Stark, P. Massonet, Agent Oriented Analysis using MESSAGE/UML", 2 nd International Workshop on Agent-Oriented Software Engineering, LNCSN o .2222, Springer-Verlag , pp. 119-135, 2001.
  - **CommonKADS**
    - C. A. Iglesias , M. Garrijo , 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
  - ***AUML***
    - B. Bauer, J.P. Muller, J. Odell , " Agent UML: A Formalism for Specifying Multiagent Software Systems ", The International Journal of Software Engineering and Knowledge Engineering, Vol. 11 (3), pp. 207-230, 2001.
  - ***DESIRE***
    - F. M. T. Brazier, B. Dunin- Keplicz, N. R. Jennings and J. Treur, "DESIRE: Modelling Multi- Agent Systems in a Compositional Formal Framework". Intl. Journal of Cooperative Information Systems, Vol. 6, pp. 67-94, 1997.

Selected References

- **Open Research Directions & Visions**
  - F. Zambonelli, 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, S. Bruekner, " Entropy and Self-Organization in Agent Systems ", *5th International Conference on Autonomous Agents* , ACM Press, May 2001.
  - R. Albert, H. Jeong, A. Barabasi, "Error and Attack Tolerance of Complex N etworks", *Nature*, 406:378-382, 27 July 2000.
  - G. D. Abowd, 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, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman and R. Weiss, "Amorphous Computing", *Communications of the ACM*, 43(5), May 2000.
  - M. Mamei, F. Zambonelli, L. Leonardi, "A Physically Grounded Approach to Coordinate Movements in a Team", 1st International Workshop on Mobile Teamwork, Vienna (A), IEEE CS Press, July 2002.
  - A. Roli, F. Zambonelli, "What Can Cellular Automata Tell Us About the Behavior of Large Multiagent Systems?", 1 st International Workshop on Engineering of Large Multiagent Systems ", Orlando (FL), May 2002, to appear in LNCS.
  - F. Zambonelli, A. Omicini, "Challenges and Research Directions in Agent-Oriented Software Engineering ", Autonomous Agents and Multi-Agent Systems 9(3). Kluwer Academic Publishers, November 2004.