

# SOFTWARE ENGINEERING METHODOLOGIES: THE AGENT APPROACH



**Ambra Molesini**

ambra.molesini@unibo.it  
Università di Bologna  
Italy



**Andrea Omicini**

andrea.omicini@unibo.it  
Università di Bologna at Cesena  
Italy

**Enrico Denti**

enrico.denti@unibo.it  
Università di Bologna  
Italy

# Outline

- Agent Oriented Software Engineering
- AOSE Methodologies: State of the Art
- Conclusions

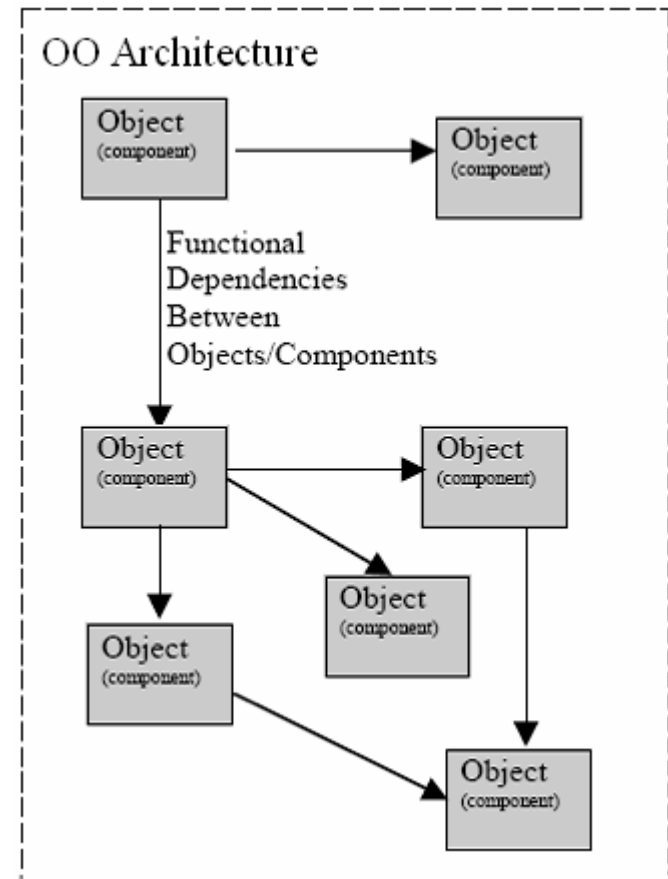
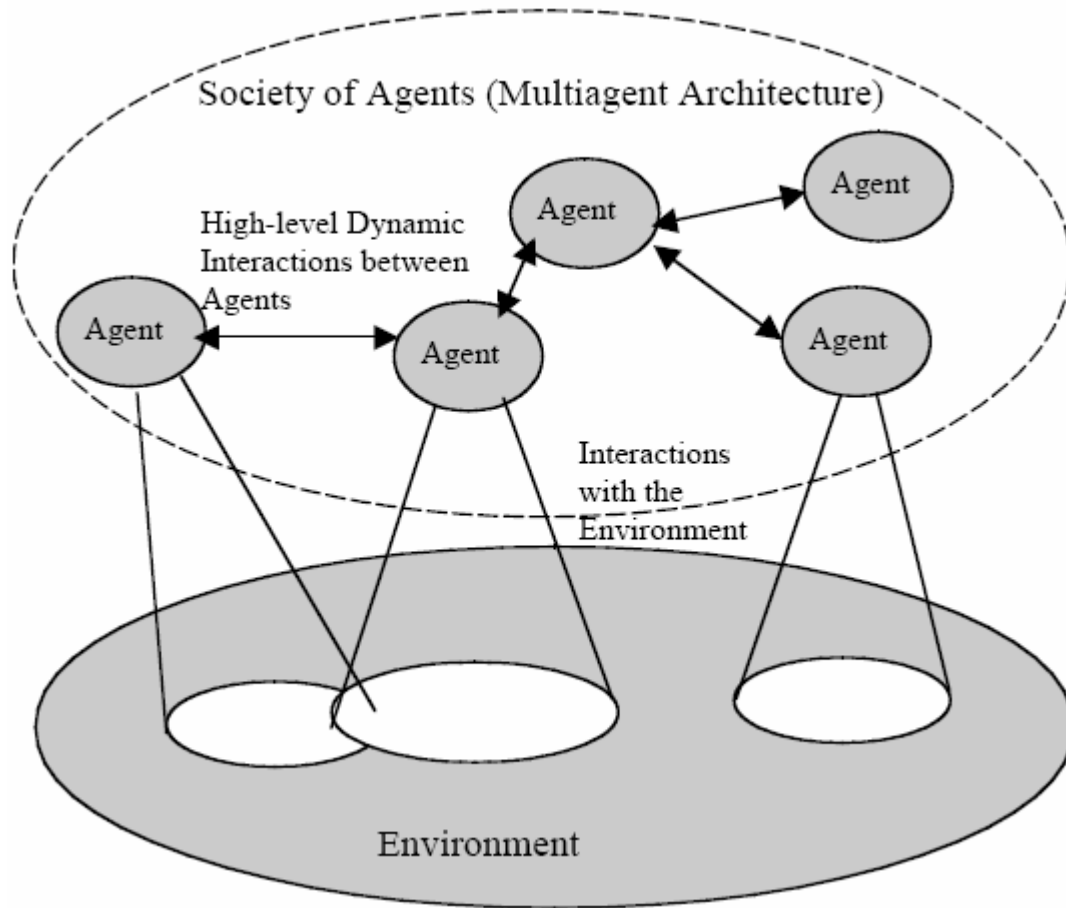
# Why Agent-Oriented Software Engineering?

- Software engineering is necessary to discipline software systems and software processes
- 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!

# Agent-Oriented Abstractions

- The development of a multi-agent system should fruitfully exploit ad-hoc **abstractions**
  - **agents**, autonomous entities, independent *loci* of control, situated in an environment, interacting with each other
  - **environment**, the world of resources agents perceive
  - **interaction protocols**, the acts of interactions between agents, and between agents and the environment
- In addition, there may be the need of abstracting the **local context** where an agent lives to handle mobility & openness
- Need of mechanisms to *manage the complexity* of *system description*

# MAS Architecture vs. OO Architecture



# AOSE

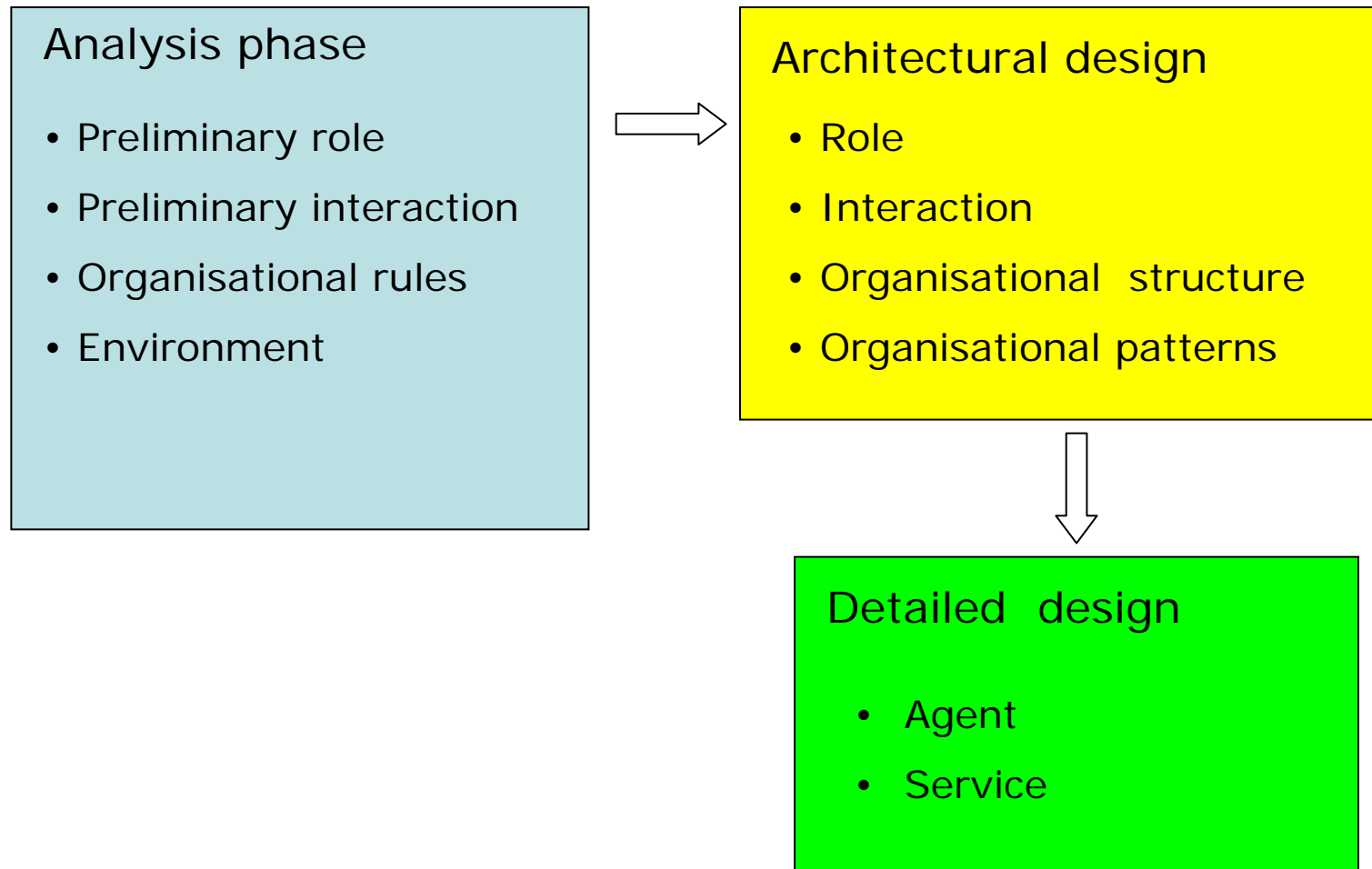
- AOSE is not only for “agent systems”
  - most of today's software systems features are very similar to those of agents and multi-agent systems
  - AOSE abstractions, methodologies, and tools are well suited for such software systems
- But of course...
  - AOSE may sometimes appear to be too “high-level”
  - there is a gap between the AOSE approach and the available technologies

# AOSE Methodologies: State of the Art

# Gaia

- The most known AOSE methodology
  - First proposed by Jennings and Wooldridge in 1999
  - 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

# Gaia Process



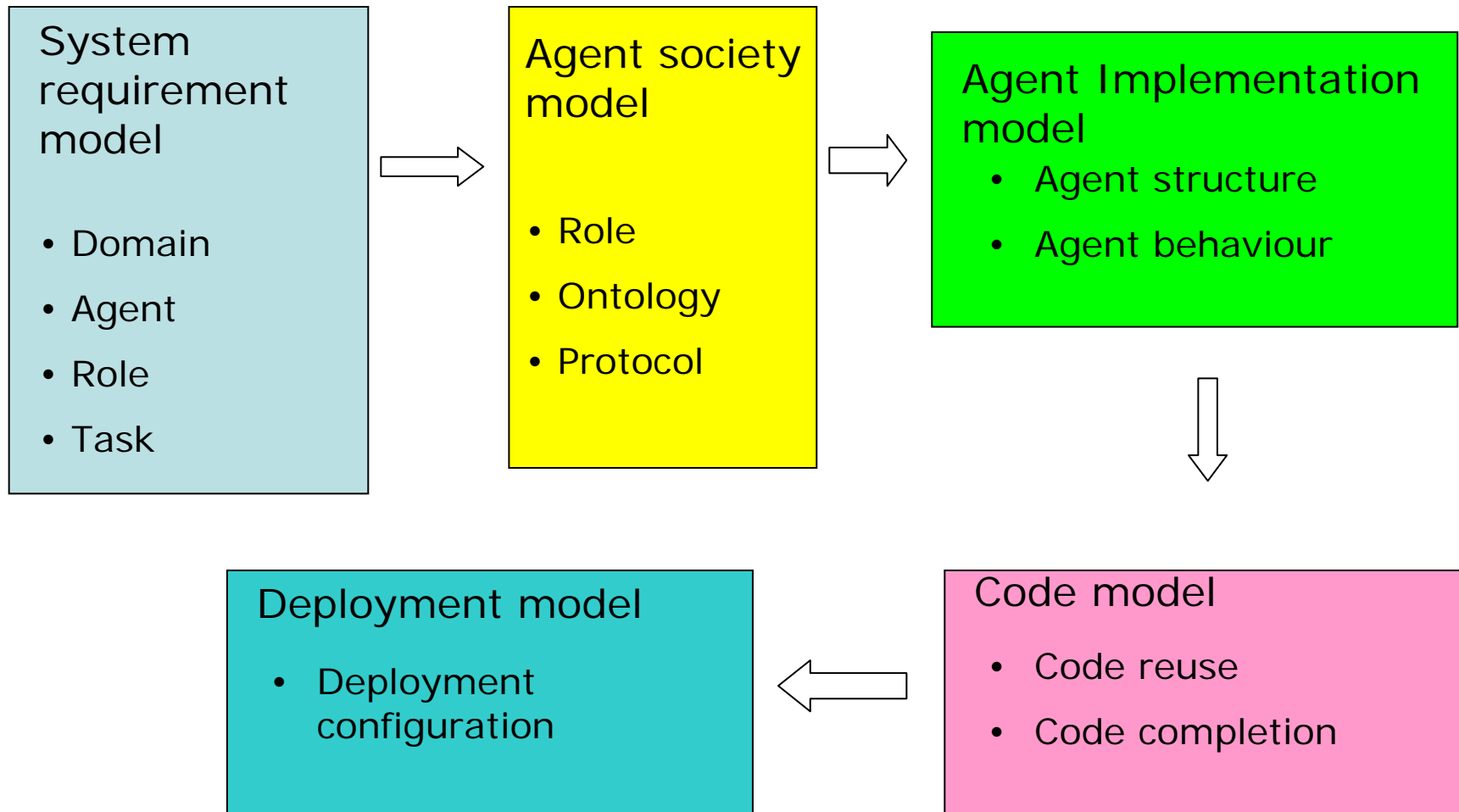
# Gaia Limitations

- 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
- ... environment?
- ... support to manage complexity?

# 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 Process



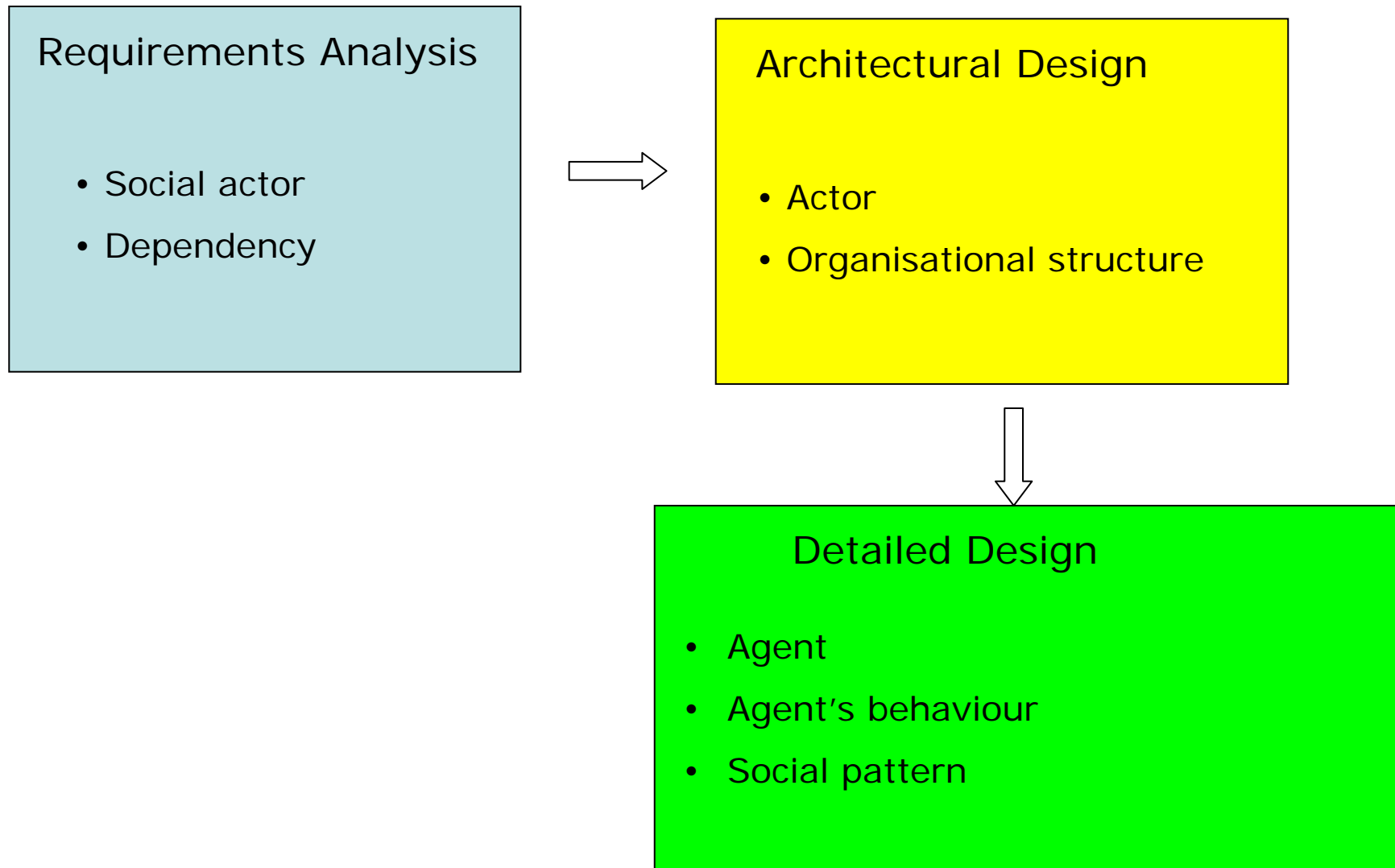
# PASSI Limitations

- *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

- *Tropos* is founded on two key features
  - (i) the notions of agent, goal, plan and various other knowledge level concepts
  - (ii) a crucial role is assigned to requirements analysis and specification when the system-to-be analysed with respect to its intended environment.
- 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

# TROPOS Process



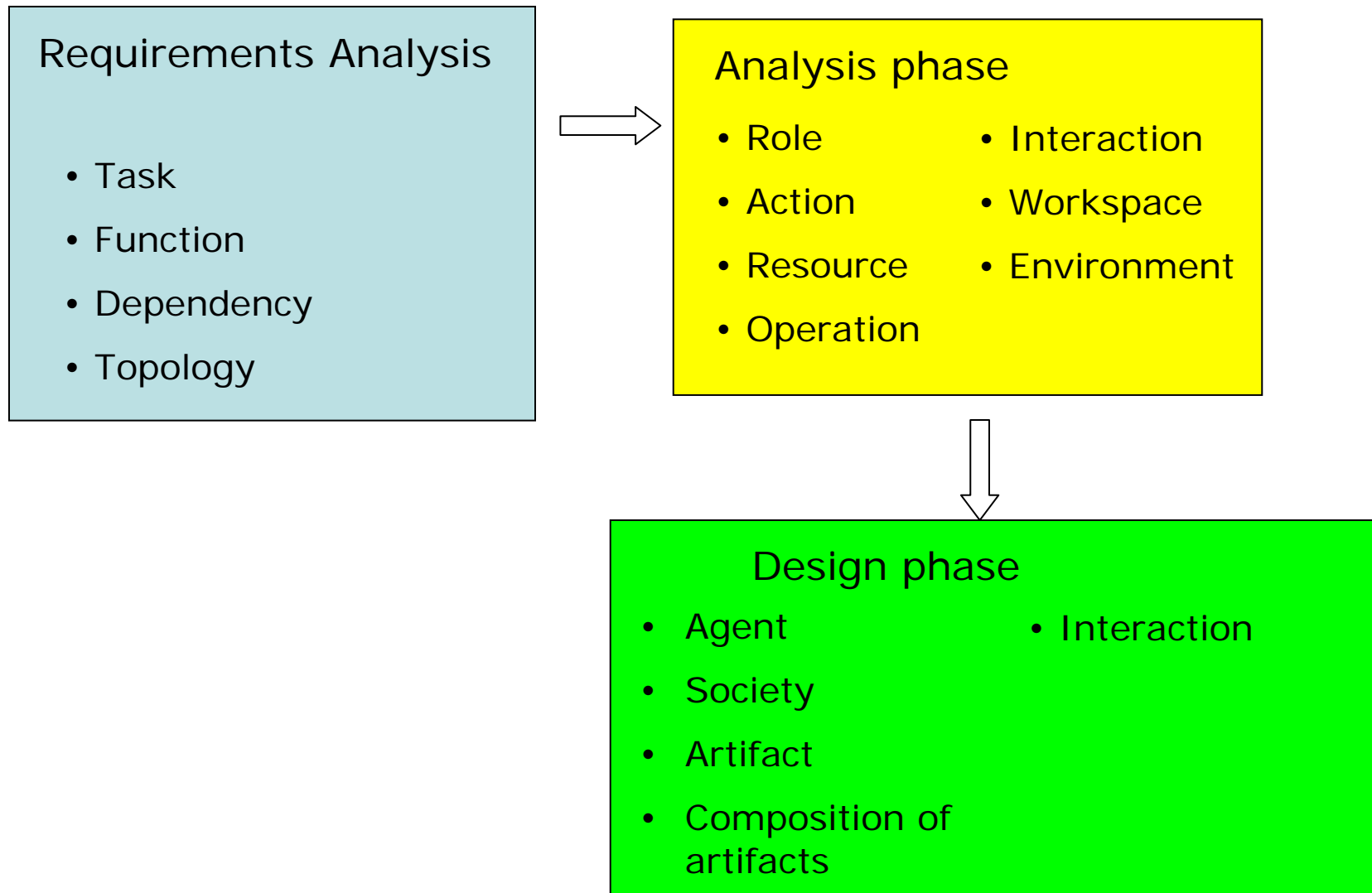
# TROPOS Limitations

- Tropos is not intended for any type of software: only for systems with identifiable stakeholders
- Tropos, in its current form, is not suitable for sophisticated software agents requiring advanced reasoning mechanism for planning
- ... environment?
- ... support to manage complexity?

# 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* (A&A) as building block for MAS development
- SODA introduces a simple layering principle in order to manage the complexity of the system description

# SODA Process



# SODA Limitations

- SODA does not deal with the activity of requirements capture
- Only Inter-agents aspects
  - need of other methodology for design intra-agents aspects

# In Summary

|               | Requirement analysis | Social structure management | Environment management | Complexity management |
|---------------|----------------------|-----------------------------|------------------------|-----------------------|
| <b>Gaia</b>   | Good                 | Good                        | Absent                 | Absent                |
| <b>Tropos</b> | Good                 | To improve                  | Absent                 | Absent                |
| <b>PASSI</b>  | Good                 | Good                        | Absent                 | Absent                |
| <b>SODA</b>   | Good                 | Good                        | Good                   | Good                  |

# Conclusions

- The models of
  - environment
  - social structureshould be key points in any agent-oriented methodologies
- Currently SODA appears the only methodology that supports both aspects
- Need of standards and languages that support the agent approach natively