The SODA AOSE Methodology

Multiagent Systems LS

Sistemi Multiagente LS

Andrea Omicini & Ambra Molesini
{andrea.omicini, ambra.molesini}@unibo.it

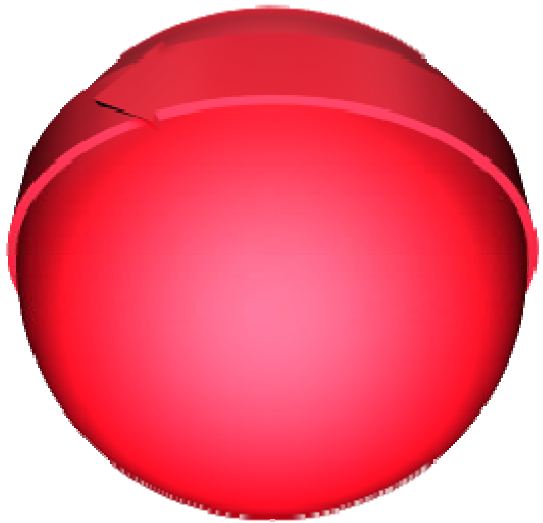
Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2009/2009

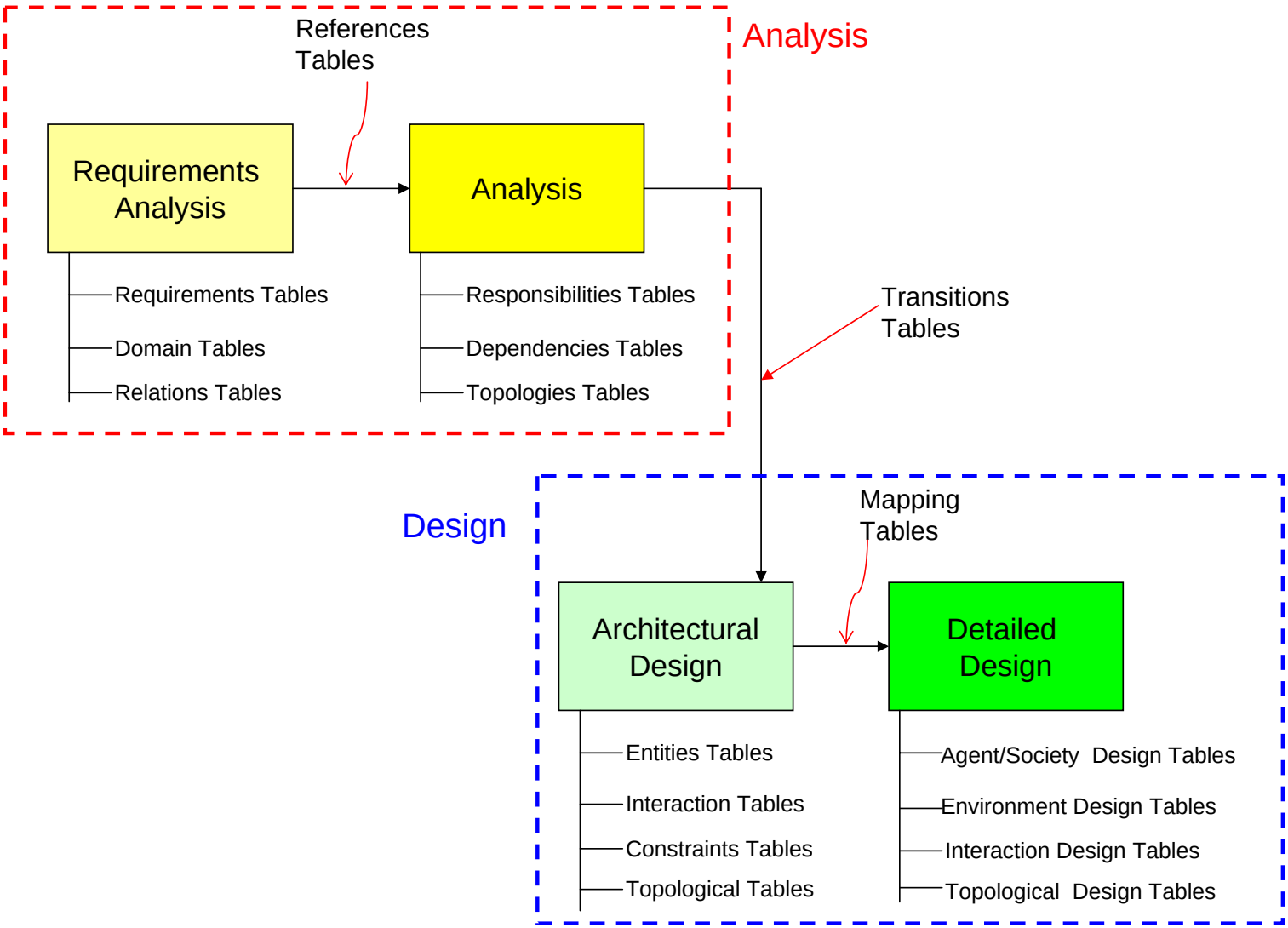
SODA: Societies in Open and Distributed Agent spaces

SODA ...

- ... is an agent-oriented methodology for the analysis and design of agent-based systems
- ... focuses on inter-agent issues, like the engineering of societies and environment for MAS [Omicini, 2001]
- ... adopts agents and artifacts – after the A&A meta-model – as the main building blocks for MAS development [Molesini et al., 2005]
- ... introduces a simple *layering* principle in order to cope with the complexity of system description [Molesini et al., 2006]
- ... adopts a tabular representation



SODA: Overview



Software Engineering Methodologies: Concerns

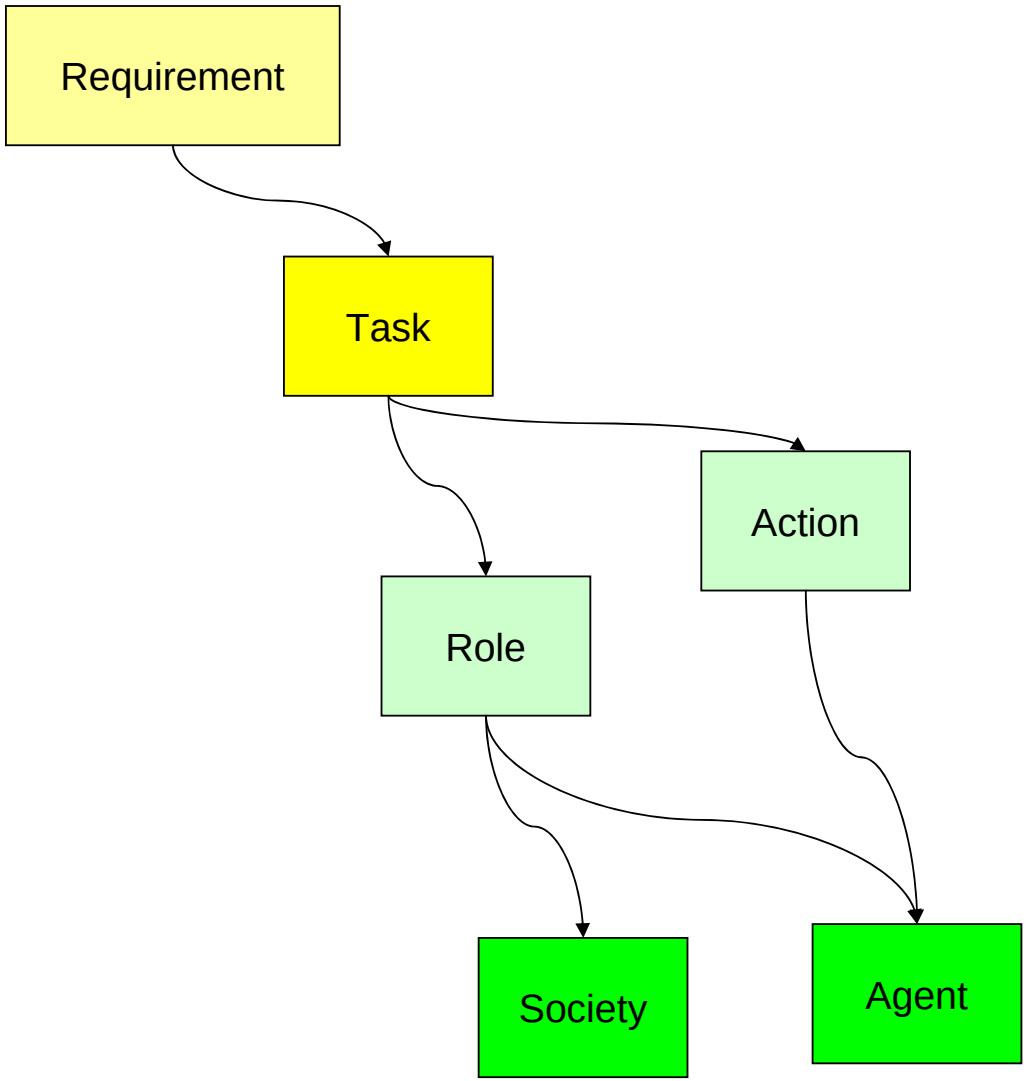
Methodology [Ghezzi et al., 2002]

- 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: the abstractions
  - one that focuses on the steps that have to be done, the work products that have to be produced – and their documentation –. . . : the process

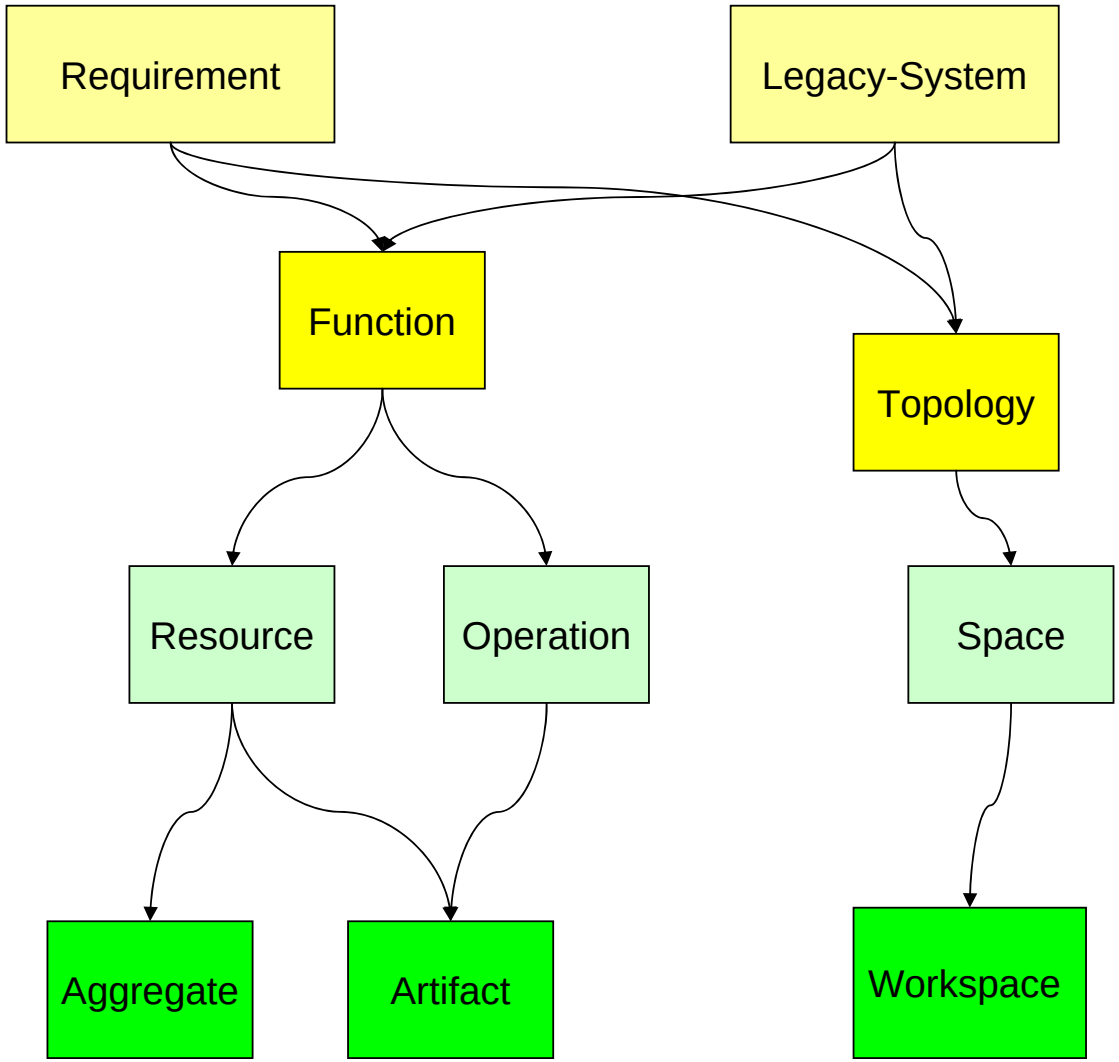
SODA Abstractions

- The abstractions supported by SODA are logically divided into three categories
  - the abstractions for modelling/designing the system active part (task, role, agent, etc.);
  - the abstractions for the reactive part (function, resource, artifact, etc.);
  - the abstractions for interaction and organisational rules (relation, dependency, interaction, rule, etc.).
- Each of the four SODA's steps models the system by exploiting a specific subset of the abstractions
  - each subset always includes at least one abstraction for each of the above categories: at least one abstraction for the system active part, one for the reactive part, and another for interaction and organisational rules.

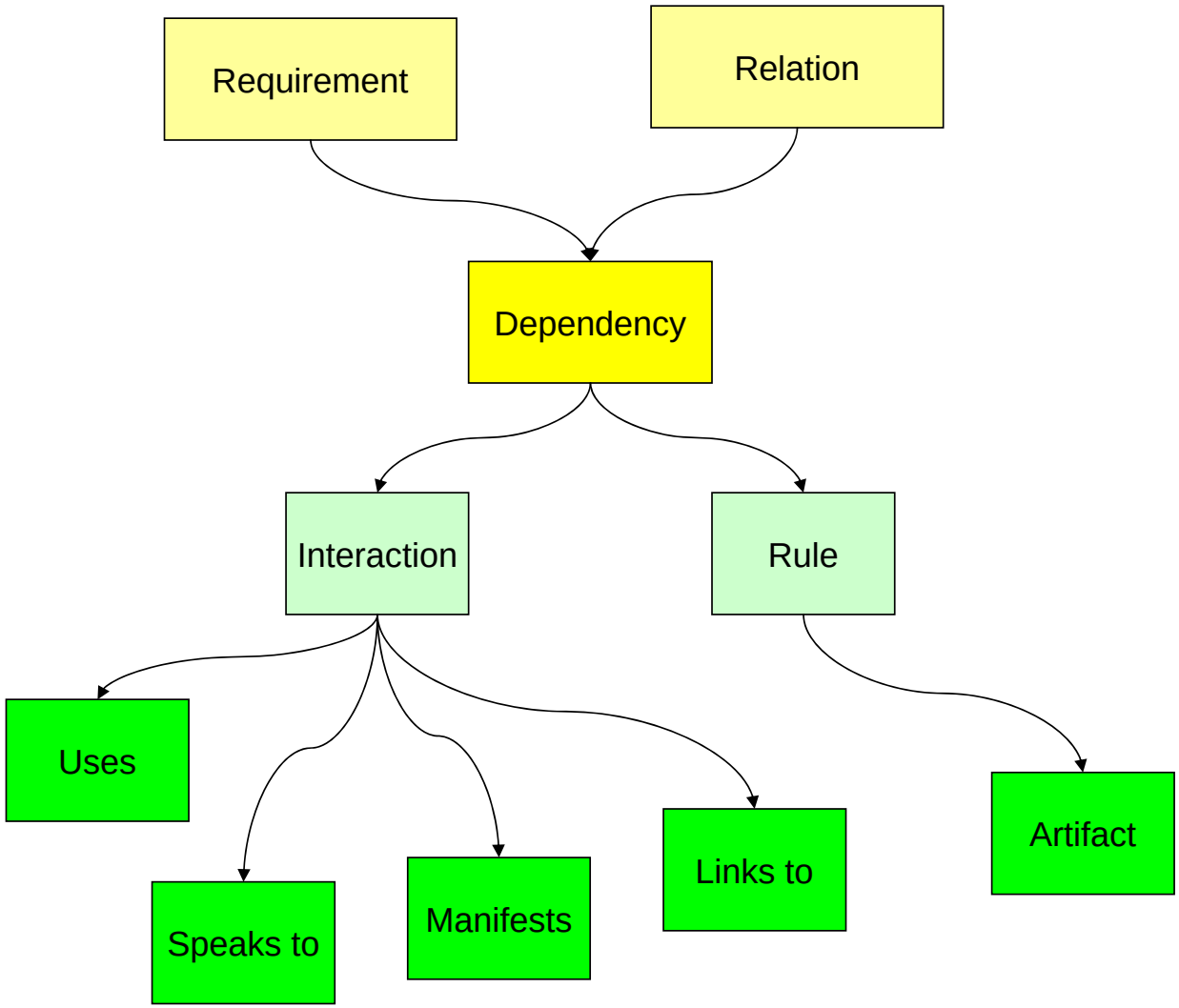
SODA Active Abstractions



SODA Reactive Abstractions



SODA Interaction & Normative Abstractions



Artifacts

- 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 by agents

Example: coordination artifacts

- 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

Classification

A possible classification for artifacts

Individual artifacts — exploited by one agent only in order to mediate its interaction with the environment. In general, individual artifacts are not directly affected by the activity of other agents, but can, through linkability, interact with other artifacts in the MAS

Social artifacts — exploited by more than one agent, mediate between two or more agents in a MAS. In general, social artifacts typically provide MAS with a service which is in the first place meant to achieve a social goal of the MAS, rather than an individual agent goal

Environmental artifacts — mediate between a MAS and an external resource. In principle, environmental artifacts can be conceived as a means to raise external MAS resources up to the agent cognitive level

Agents & Artifacts (A&A)

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

A&A Ingredients in SODA

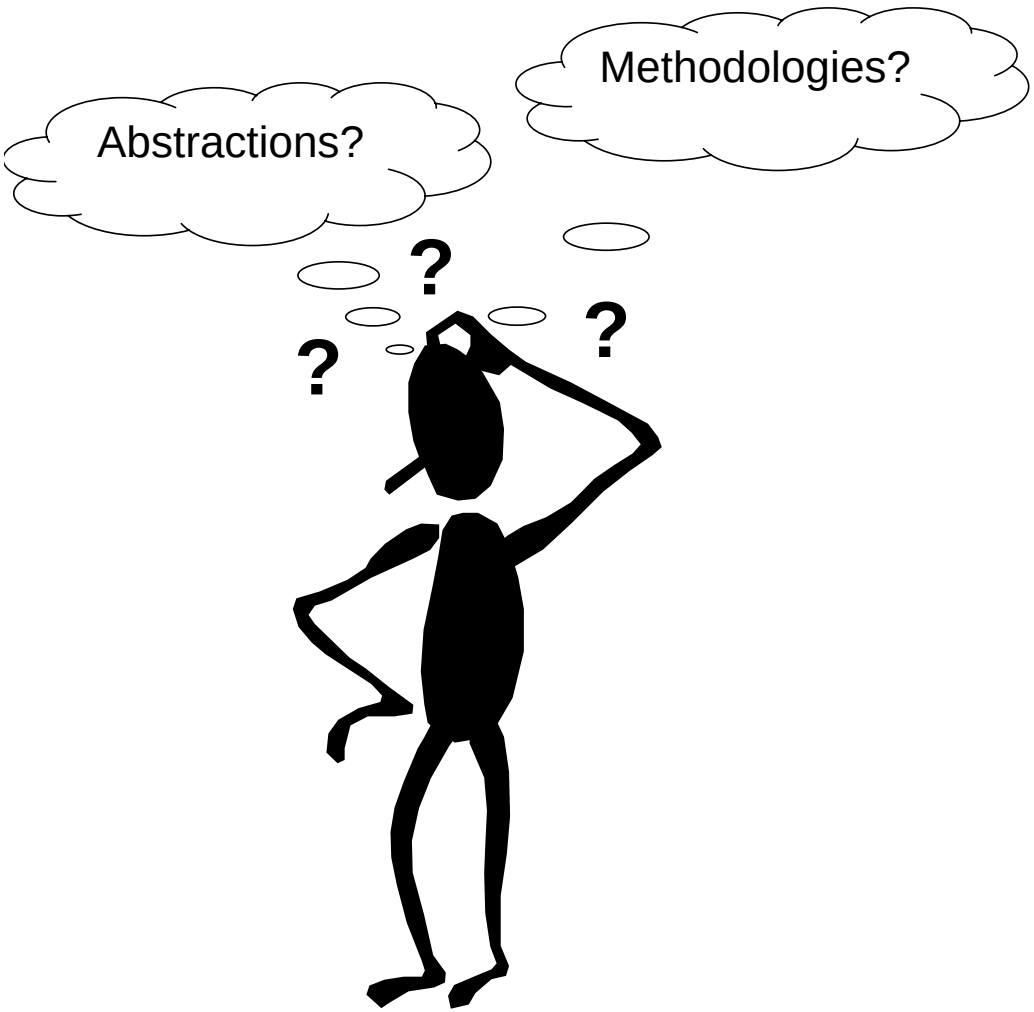
Agents model individual and social activities

Artifacts *glue* agents together, as well as MAS and the environment

- artifacts mediate between individual agents and MAS
- artifacts build up agent societies
- artifacts wrap up the resources of MAS and bring them to the cognitive level of agents

Workspaces structure agents and artifacts organisation & interaction

SODA Abstractions: A Formal Representation?



- How can we represent the SODA abstractions? ...
- ... and their relationships? ...
- ... in a standard way in order to support methodologies comparisons? ...
- The meta-modelling technique from Software Engineering can help us

Meta-models for Software Engineering Methodologies

Definition

Meta-modelling is the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for the modelling in a predefined class of problems

- A 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 consists of instantiating the system meta-model that the designers have in their mind in order to fulfill the specific problem requirements [Bernon et al., 2004]

Using Meta-models

- Meta-models are useful for specifying the concepts, rules and relationships used to define a family of related methodologies
- Although it is possible to describe a methodology without an explicit meta-model, formalising the underpinning ideas of the methodology in question is valuable when checking its consistency or when planning extensions or modifications
- Meta-models are often used by methodologists to construct or modify methodologies

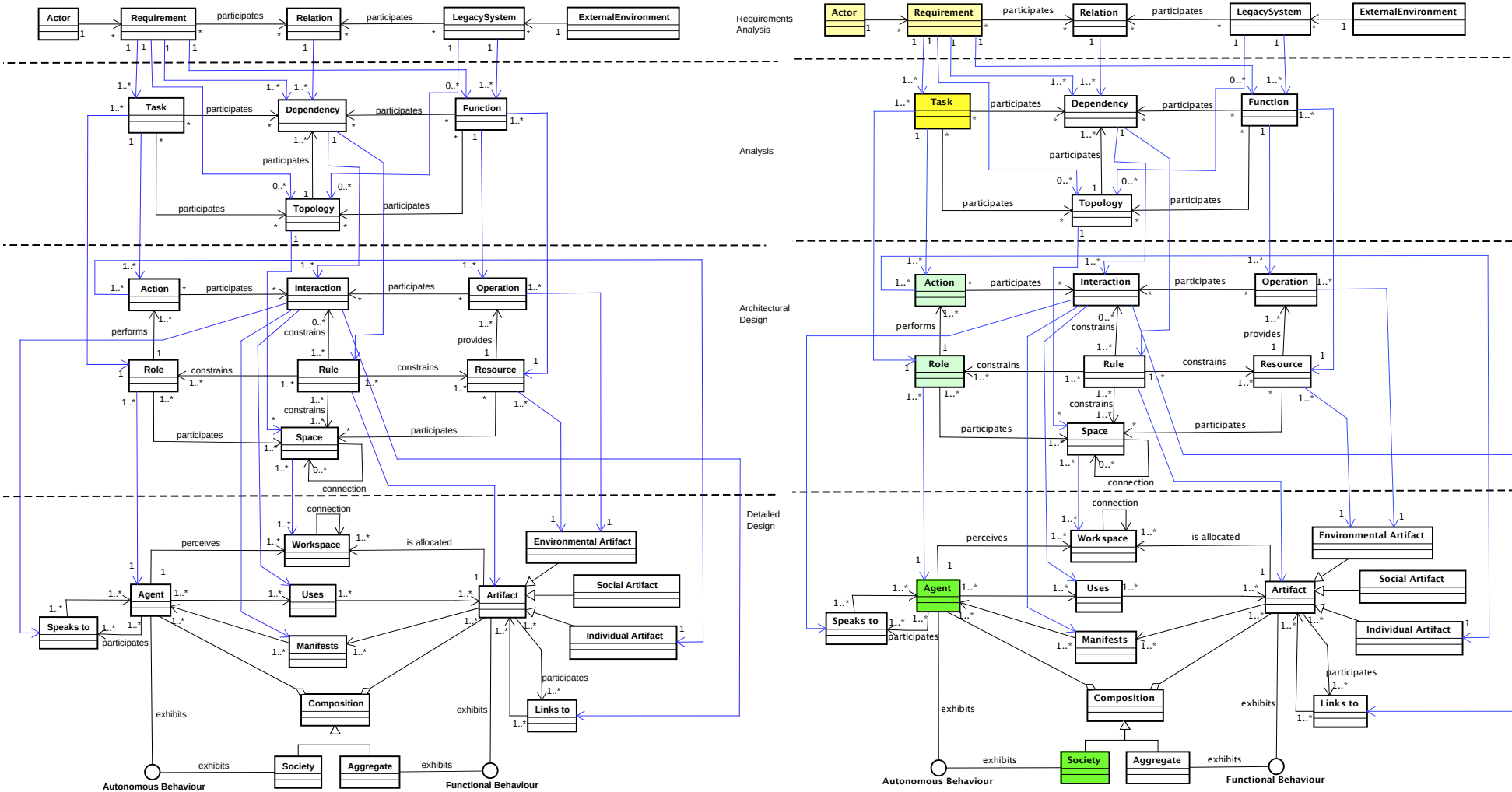
Meta-models & Methodologies

- Methodologies are used by software development teams to construct software products in the context of software projects
- Meta-model, methodology and project constitute, in this approach, three different areas of expertise that, at the same time, correspond to three different levels of abstraction and three different sets of fundamental concepts
- As the work performed by the development team at the project level is constrained and directed by the methodology in use, the work performed by the methodologist at the methodology level is constrained and directed by the chosen meta-model
- Traditionally, these relationships between *modelling layers* are seen as instance-of relationships, in which elements in one layer are instances of some element in the layer above

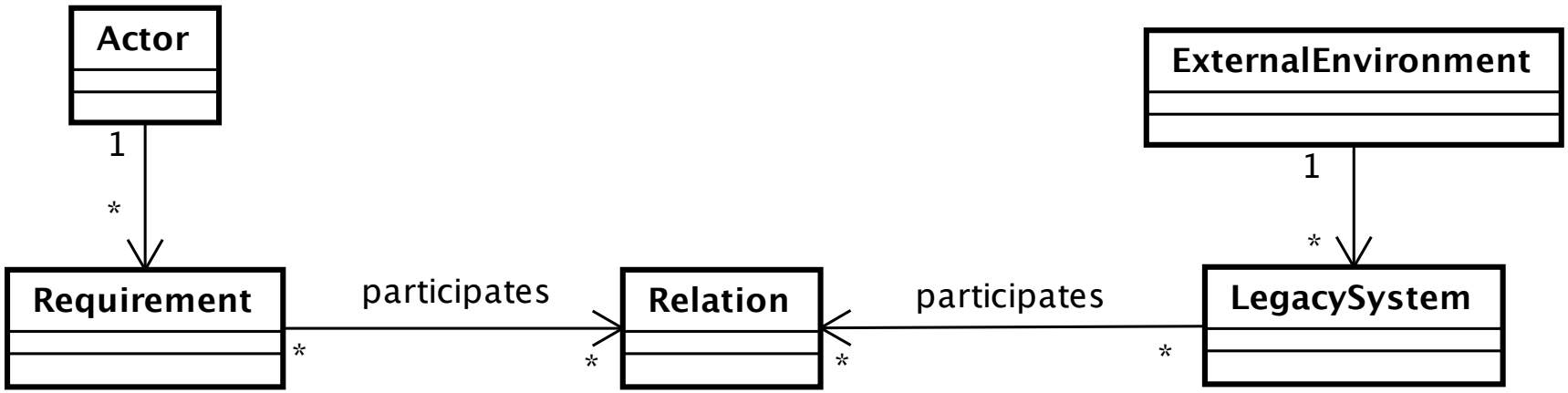
SODA Abstractions Adopted at Each Step

- *Analysis phase*
  - *Requirements Analysis* — the system's requirements and the external environment are analysed and modelled
  - *Analysis* — the system's requirements are modelled in terms of tasks, functions, topologies and dependencies
- *Design phase*
  - *Architectural Design* — in this phase we analyse the solution domain, the system is modelled in terms of roles, resources, actions, operations, interactions, rules and spaces
  - *Detailed Design* — in this phase we design the system in terms of agents, societies, artifacts, compositions, aggregates, workspaces, uses, links to, manifests and speaks to

The SODA Meta-model



Requirements Analysis Meta-model



Requirements Analysis

Actor — is a user of the systems that needs several functionalities from the systems. We use the system as an actor in order to express several non-functional requirements as security, standards and so on. The actors are used in order to facilitated the trace of the sources of requirements.

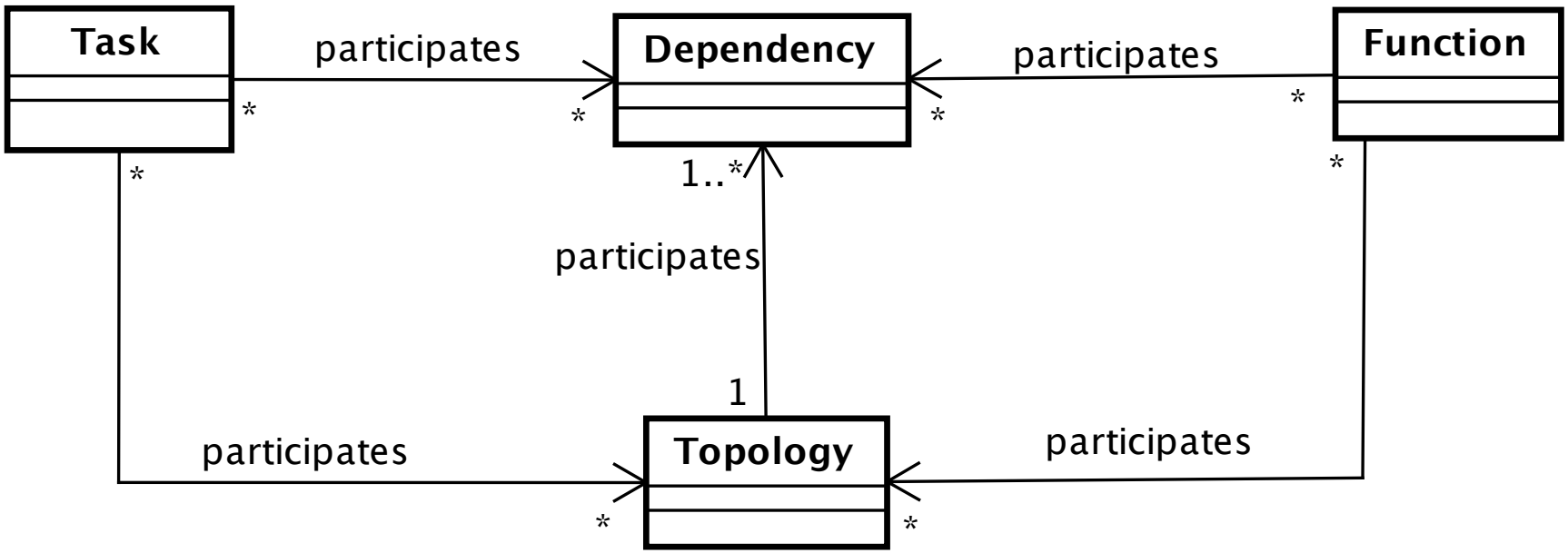
Requirement — is a functional, non-functional or domain description of the system service and constraint of the system.

External-Environment — is the external world of the system made by legacy systems that will interact with the system.

Legacy-System — is a single legacy system.

Relation — is a relationship among requirements and contexts.

Analysis Meta-model



Analysis

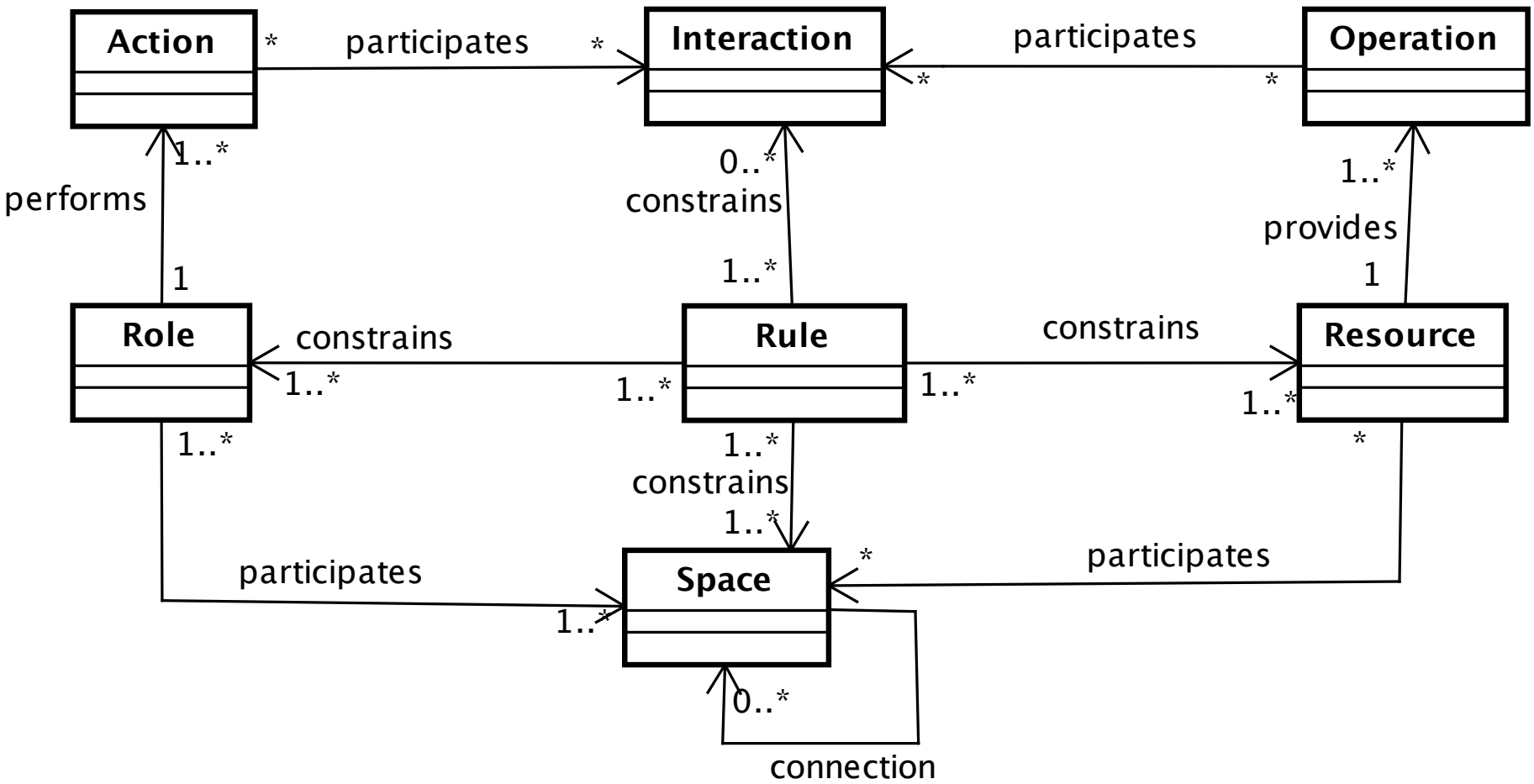
Task — is an activity that requires one or more competences and the use of functions

Function — is an reactive activity that aimed at supporting tasks

Dependency — is any relationship (interactions, constraints...) among other (tasks and/or functions) abstract entities

Topology — is any topological necessity of the environment's structure, often could be derived from functions. It is important to note that topology could influence the tasks because topology could constrains the achievement of tasks

Architectural Design Meta-model



Architectural Design

Role — is defined as the abstraction responsible for the achievement of one or more tasks

Resource — is defined as the abstraction that provides some functions

Action — represents an action that the role potentially could be able to do

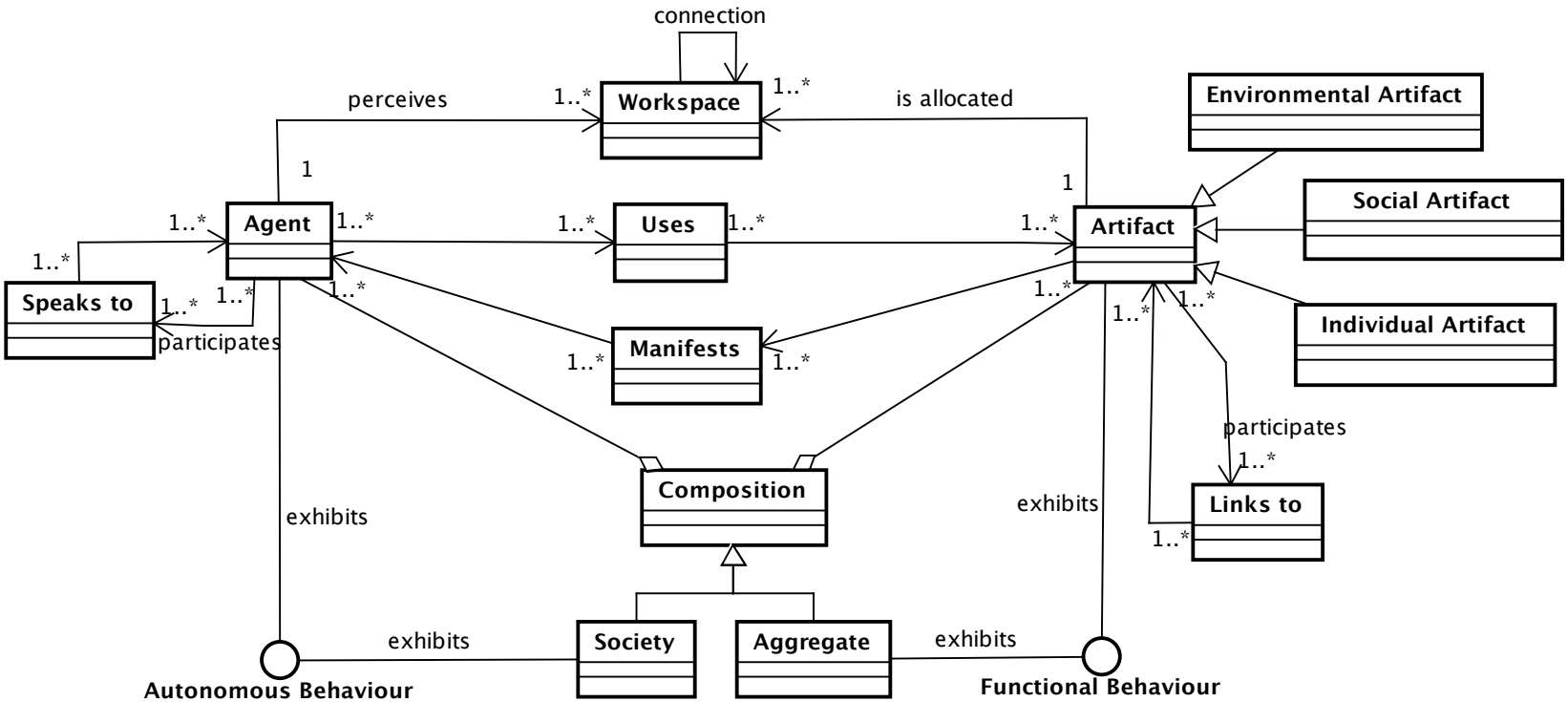
Operation — represents the operation that the resource is potentially able to provide

Interaction — represents the acts of the interaction among roles, among resources and between roles and resources

Rule — enables and bounds the entities' behaviour

Space — is a conceptual locus in the environment

Detailed Design Meta-model



Detailed Design

Agent — is an autonomous entity able to play several roles

Society — is a group of interacting agents and artifacts when its overall behaviour is essentially an *autonomous*, proactive one.

Artifact — is a group of interacting agents and artifacts when its overall behaviour is essentially a *functional*, reactive one.

Aggregate — is defined as the abstraction responsible for a collection of artifacts

Workspace — is a conceptual locus in the environment

Uses — the act of interaction between agent and artifact: agent uses artifact

Speaks to — the act of interaction among agents: agent speaks with another agent

Manifests — the act of interaction between artifact and agent: artifact manifests itself to agent

Links to — the act of "interaction" among artifact: artifact is linked to another artifact

Development Process

Development Process [Cernuzzi et al., 2005]

- The development process is an ordered set of steps that involve all the activities, constraints and resources required to produce a specific desired output satisfying a set of input requirements
- Typically, a process is composed by different stages/phases put in relation with each other
- Each stage/phase of a process identify a portion of work definition to be done in the context of the process, the resources to be exploited to that purpose and the constraints to be obeyed in the execution of the phase
- Case by case, the work in a phase can be very small or more demanding
- Phases are usually composed by a set of activities that may, in turn, be conceived in terms of smaller atomic units of work (steps)

Software Process

Software Process [Fuggetta, 2000]

The software development process is the coherent set of policies, organisational structures, technologies, procedures and deliverables that are needed to conceive, develop, deploy and maintain a *software* product

The software process exploits a number of contributions and concepts [Fuggetta, 2000]

Software development technology — Technological support used in the process. Certainly, to accomplish software development activities we need tools, infrastructures, and environments

Software development methods and techniques — Guidelines on how to use technology and accomplish software development activities. The methodological support is essential to exploit technology effectively

Organisational behavior — The science of organisations and people.

Marketing and economy — Software development is not a self-contained endeavor. As any other product, software must address real customers' needs in specific market settings.

SPEM

- SPEM (Software Process Engineering Meta-model) [Object Management Group, 2008] is an OMG standard object-oriented meta-model defined as an UML profile and used to describe a concrete software development process or a family of related software development processes
- SPEM is based on the idea that a software development process is a collaboration between active abstract entities called *roles* which perform operations called *activities* on concrete and real entities called *work products*
- Each role interacts or collaborates by exchanging work products and triggering the execution of activities
- The overall goal of a process is to bring a set of work products to a well-defined state

SPEM

Goals

The goals of SPEM are to

- support the representation of one specific development process
- support the maintenance of several unrelated processes
- provide process engineers with mechanisms to consistently and effectively manage whole families of related processes promoting process reusability

SPEM


Standardize representation and manage libraries of reusable Method Content
Content on agile development
Content on managing iterative development
Guidance on serialized java beans
JUnit user guidance
Content on J2EE
Configuration mgmt guidelines
Develop and manage Processes for performing projects
Process for Custom Application Development with J2EE
Process for Embedded System Development
Process for SOA Governance
Process assets patterns
Standard or reference processes
Enactable project plan templates
Corporate guidelines on compliance
Configure a cohesive process framework customized for my project needs
Create project plan templates for Enactment of process in the context of my project

SPEM

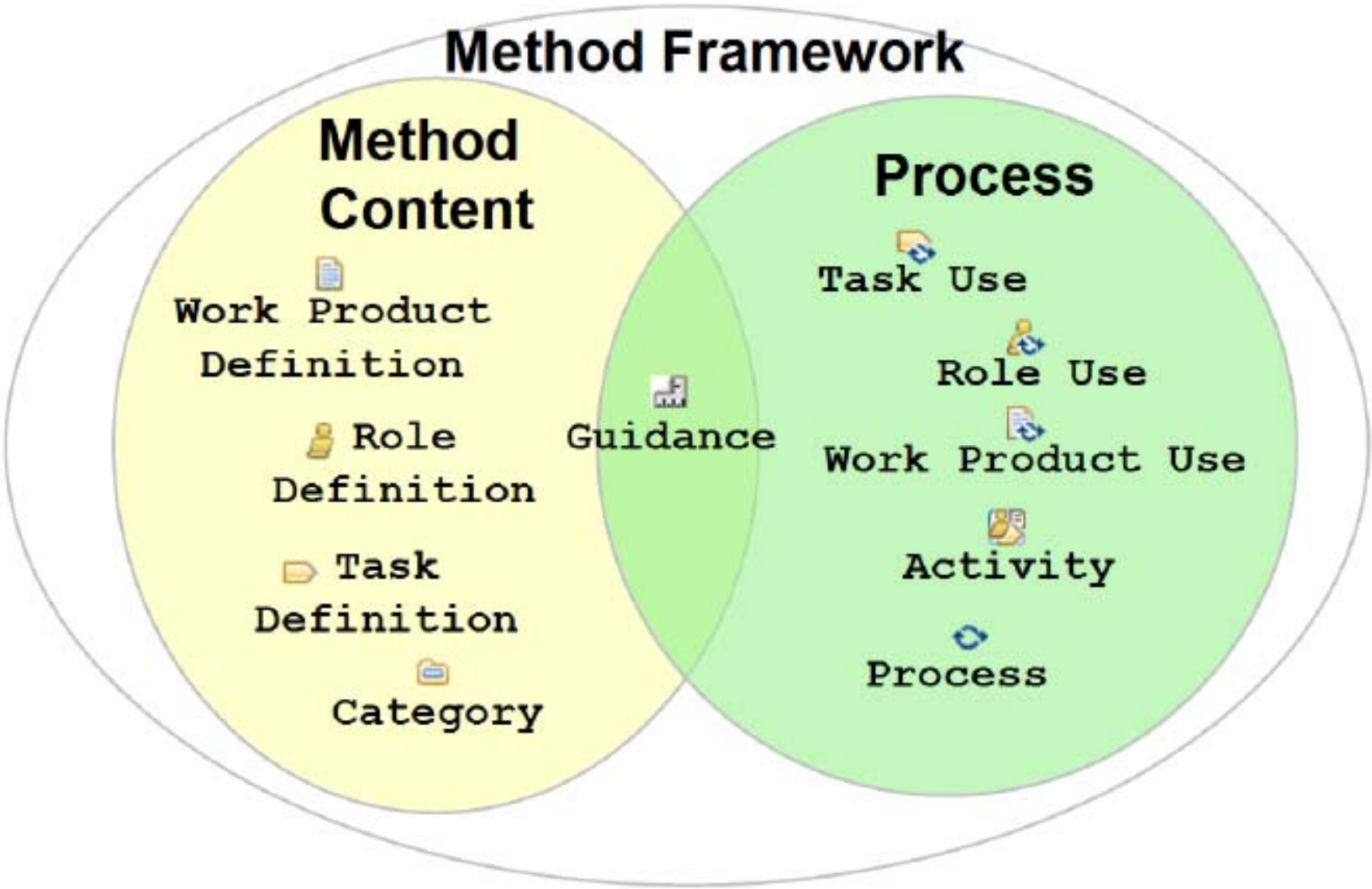
- Clear separation between

  Method Contents — introduce the concepts to document and manage development processes through natural language description

  Processes — defines a process model as a breakdown or decomposition of nested *Activities*, with the related *Roles* and input / output *Work Products*

  Capability patterns — reusable best practices for quickly creating new development processes

SPEM: Method Content and Process



SPEM Notation

Stereotype	Symbol
Activity	
Category	
Composite role and Team	 
Guidance	
Milestone	
Process	
Process Component	
Process Pattern	
Role Definition and Use	 
Task Definition and Use	 
Tool Definition	
WorkProduct Definition and Use	 

Layering: The Intuition

A complex system

is any system featuring a large number of interacting components, whose aggregate activity is nonlinear and typically self-organisation [Simon, 1996].

The human mind

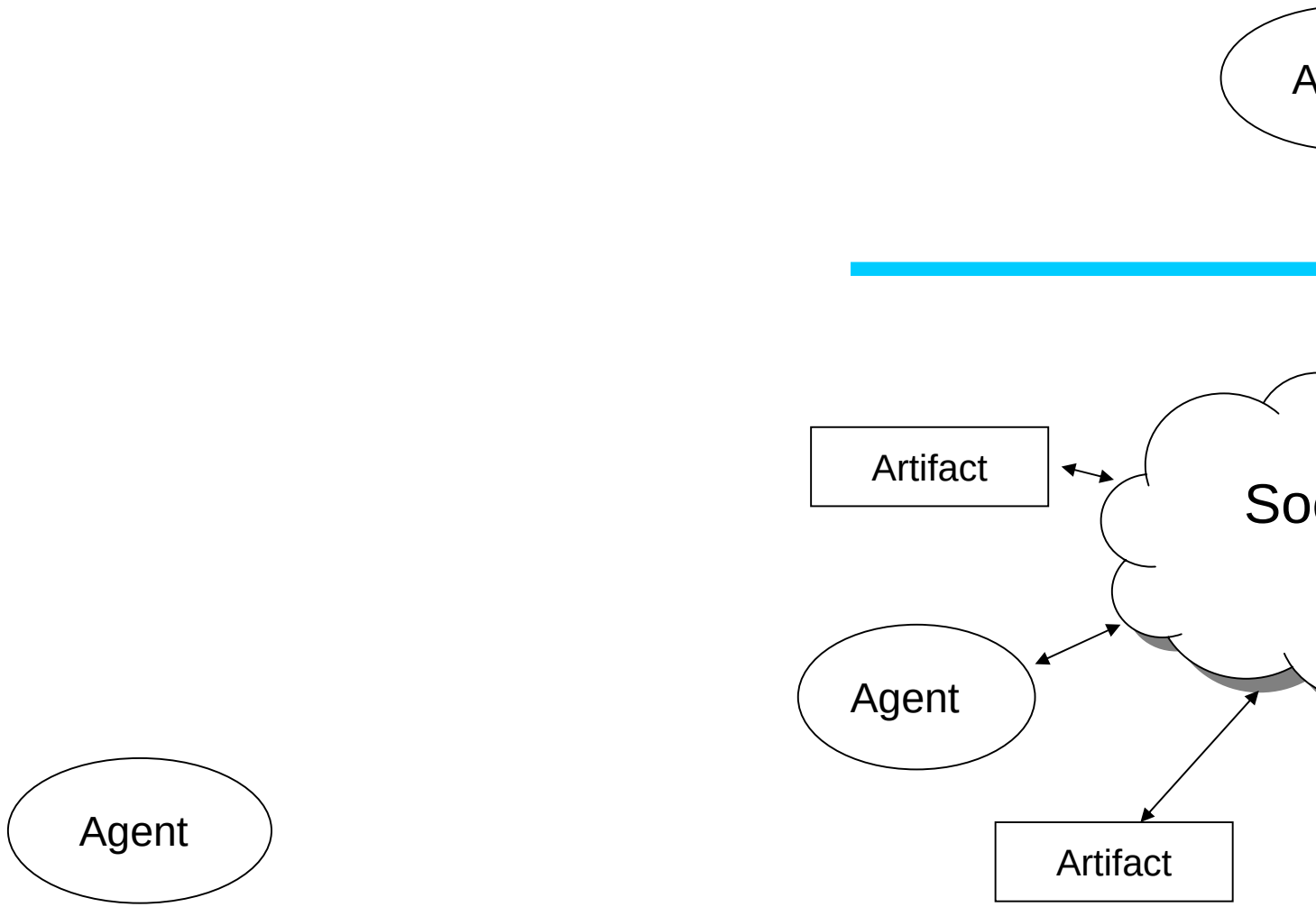
is not able to reason about

- a great number of different components / parts
- too many details...

We need

- models...
- ... capturing the systems'details at different levels of abstraction

Layering: The Intuition I



Layering: The Intuition II

Managing Complexity

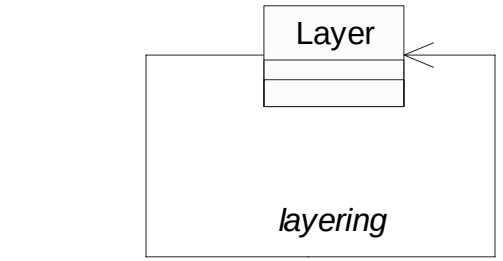
- Complexity is inherent in real-life systems
- Complexity management: balancing between *completeness* and *clarity*
- Analysis and design of real-life systems proceed *middle-out*
- Complex systems call for *layered*, hierarchical explanations
- Each layer is in some sense autonomous
  - at the same time, layers are organised in a hierarchy
  - each layer is strictly connected with the upper /lower layers

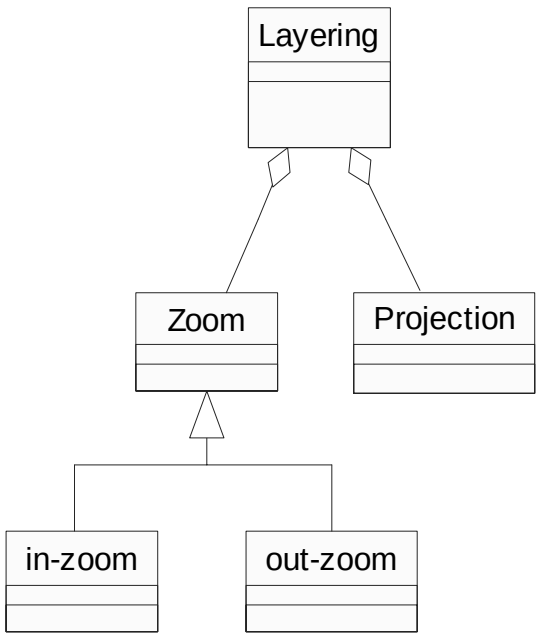
Layering, Systems & MAS

- In many branches of sciences, systems are represented as organised on different layers
- Each level is essential to the general understanding of the system's wholeness, but at the same time, no level can be understood in isolation
- When applied to the engineering of MAS, this principle suggests
  - that MAS models, abstractions, patterns and technologies can be suitably categorised and compared using a layered description
  - that agent-oriented processes and methods should support some forms of MAS layering

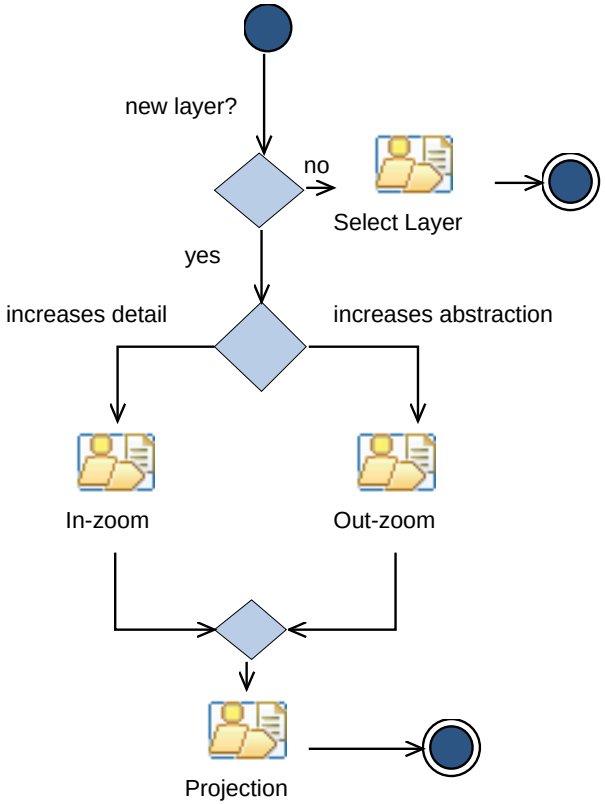
Layering in SODA: The Meta-model

- The layering principle is achieved by means of the zoom and projection mechanisms [Molesini et al., 2006, Molesini et al., 2008a]
- Two kinds of zoom
  - in-zoom — from an abstract to a more detailed layer
  - out-zoom — from a detailed to a more abstract layer
- The *projection mechanism* projects entities from one to another layer





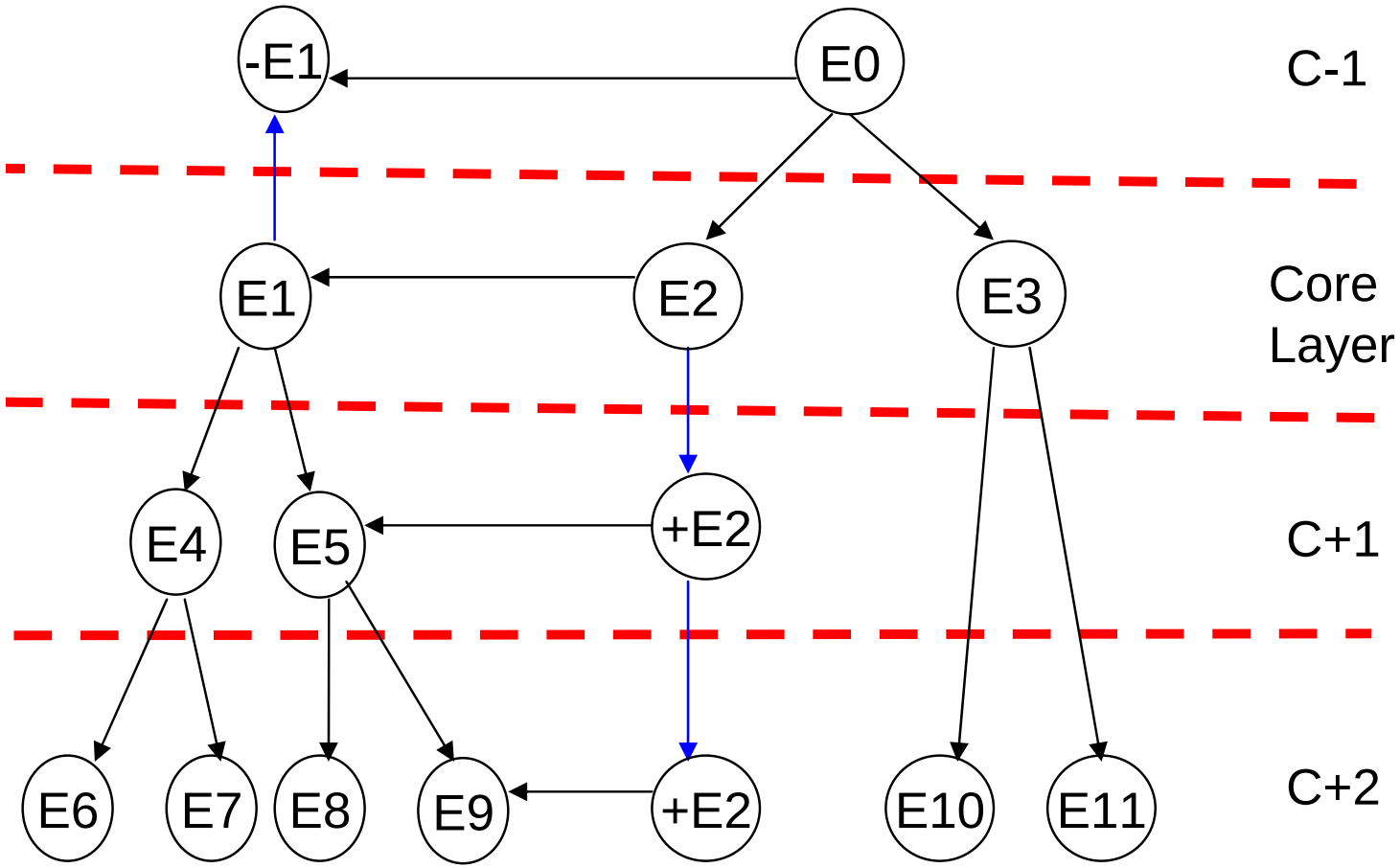
Layering in SODA: Process



Layering Principle

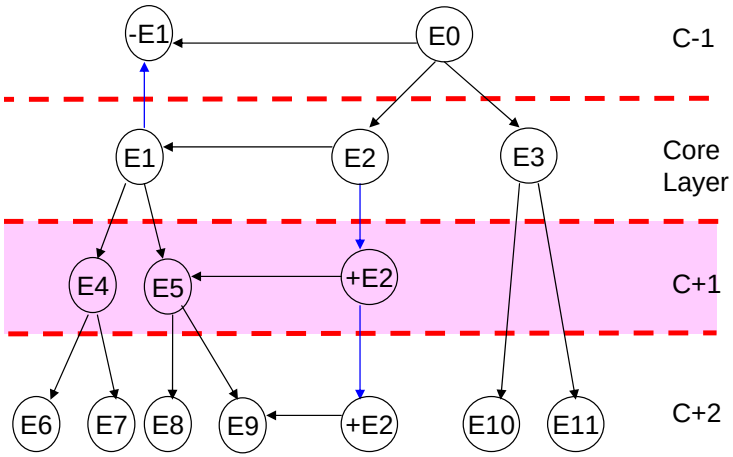
- In general, when working with SODA, we start from a certain layer, we could call *core layer*, and it is labelled with "c"
- The core layer is always **complete**
- In the other layer we find only the in/out zoomed entities and the projection entities.
- The in-zoomed layers are labelled with "c+1",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

Example

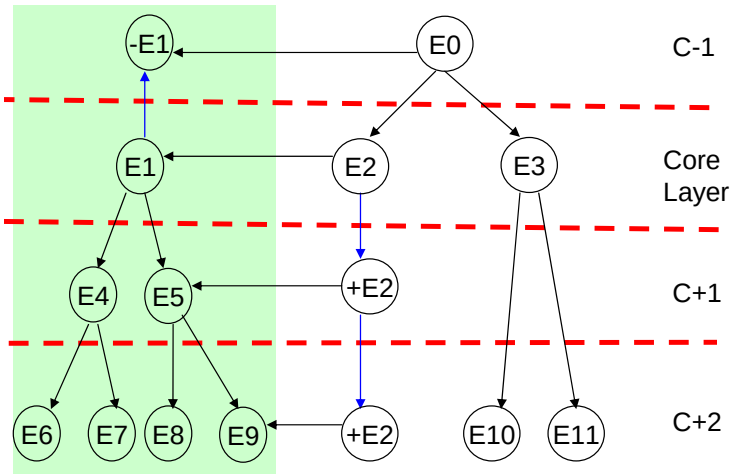


System's Views

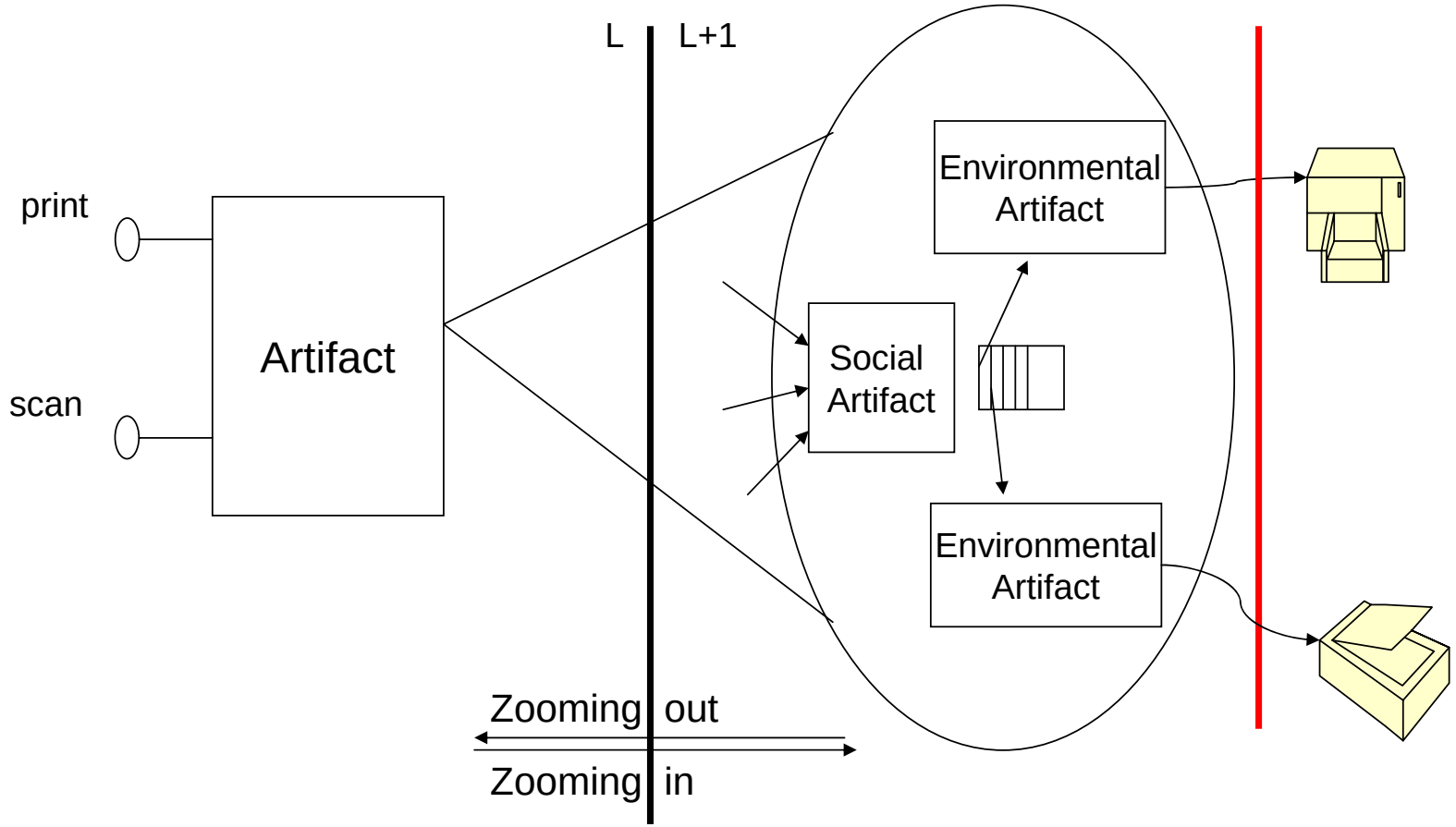
Horizontal view: analyse the system in one level of detail



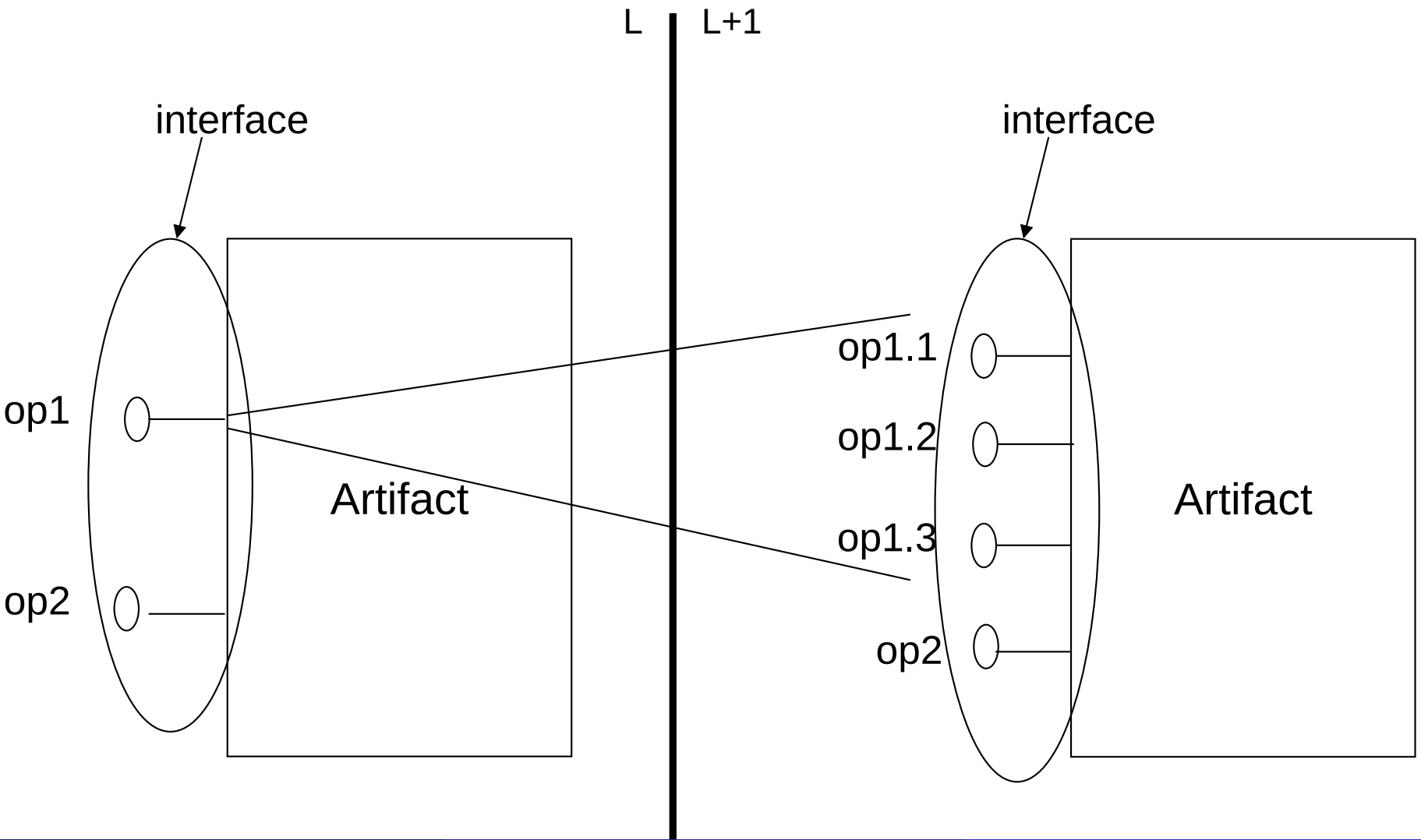
Vertical view: analyse one kind of abstract entity



Zooming Artifact 1/2



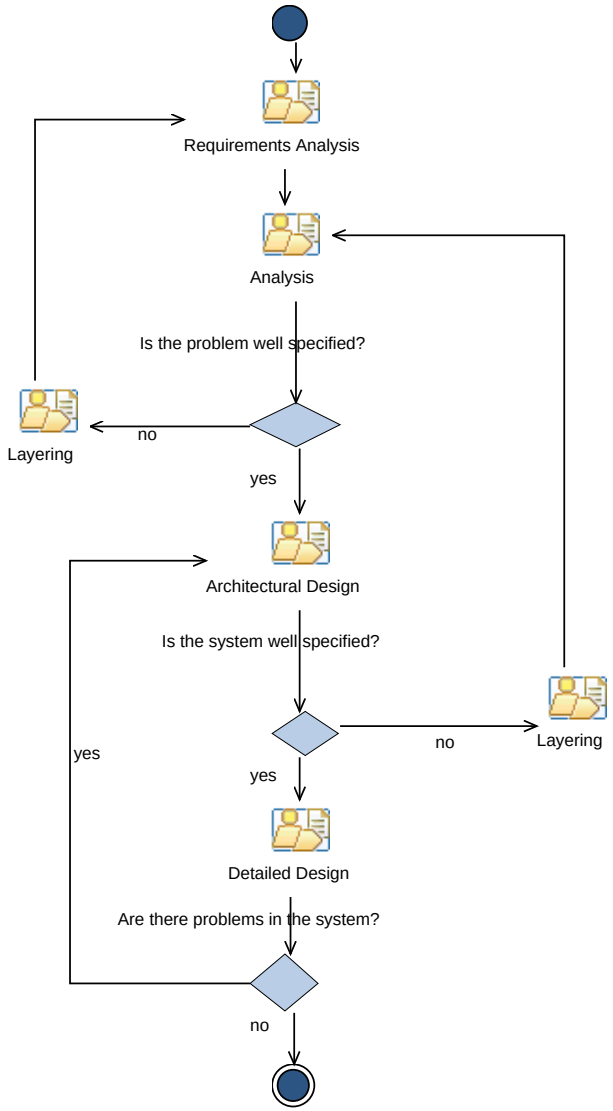
Zooming Artifact 2/2



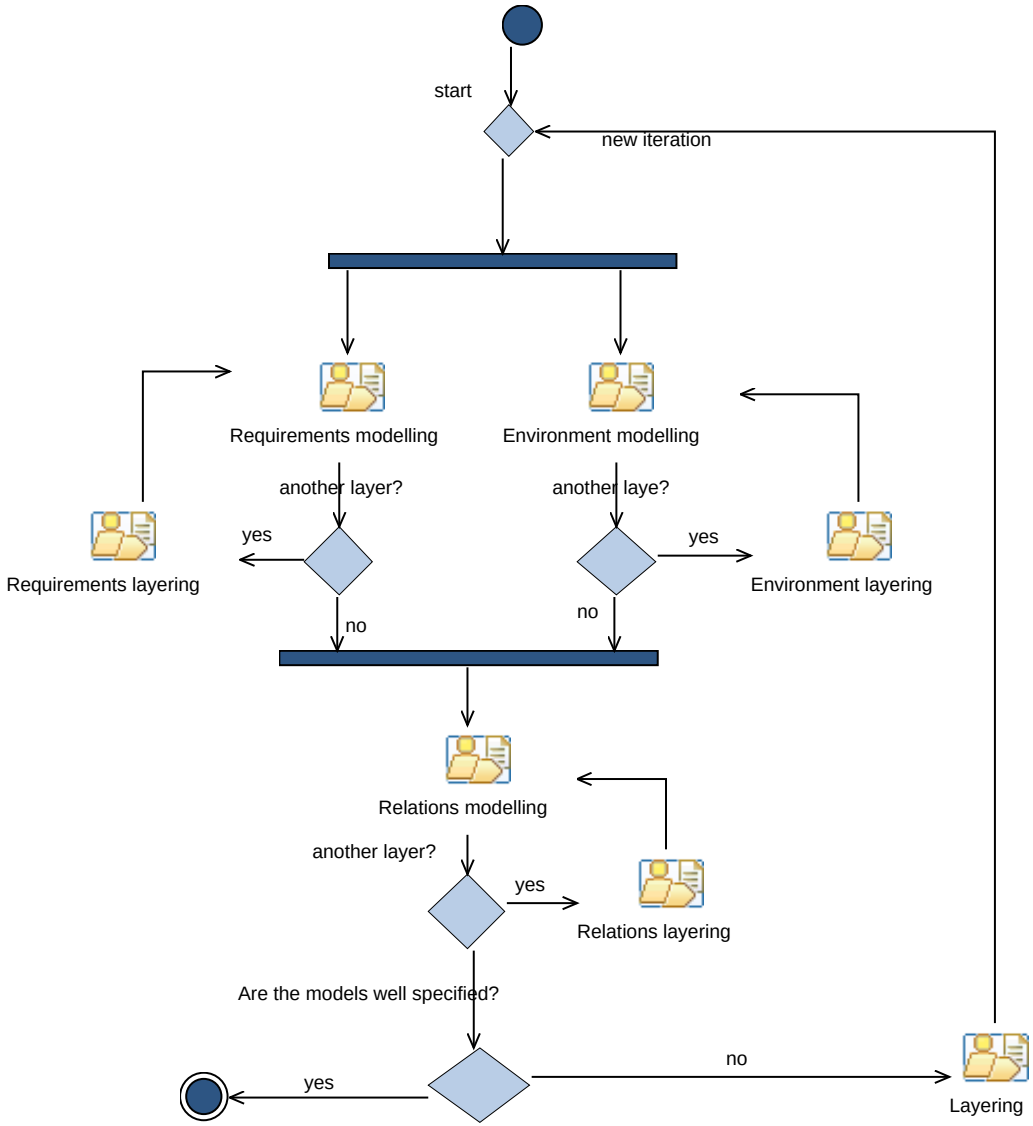
SODA Process Organisation

- Four sub-phases: Requirements Analysis, Analysis, Architectural Design, Detailed Design
- Each sub-phase is modelled as a separate and independent Method Content
- A specific process is defined for each sub-phase
- Reusing of these processes to create the whole SODA process
- Each sub-phase – and each model in the sub-processes – is represented as an activity, related to the corresponding SPEM's Task in the specific Method Content
- Adoption of the Layering Capability pattern

The SODA Process [Molesini et al., 2008b]



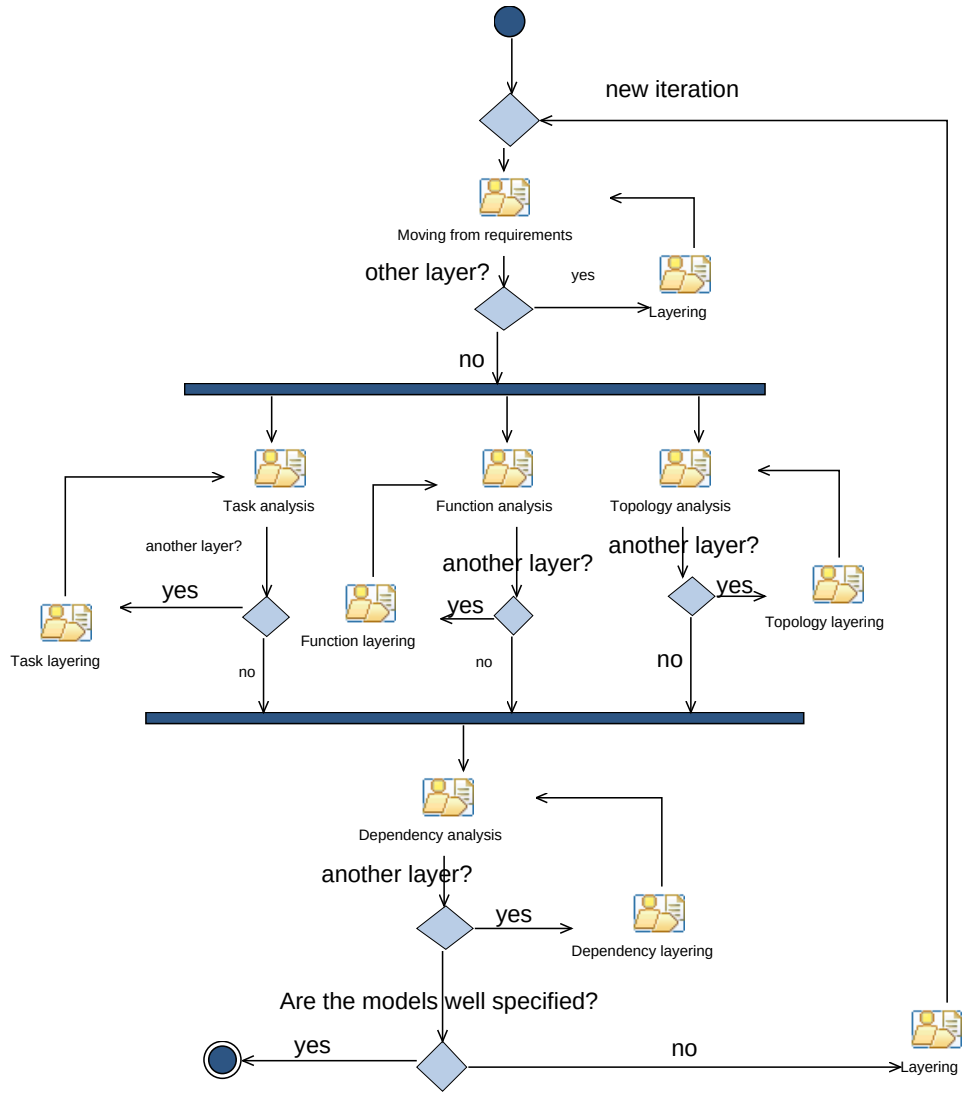
Requirements Analysis Process



Requirements Analysis Process

- *Requirements modelling* activity: *requirement* and *actor* are used for modelling the customers' requirements and the requirement sources
- *Environment modelling* activity: *external-environment* notion is used as a container of the *legacy-systems* that represent the legacy resources of the environment
- *Relation modelling* activity: *relation* is used for modelling the relationships between requirements and legacy systems

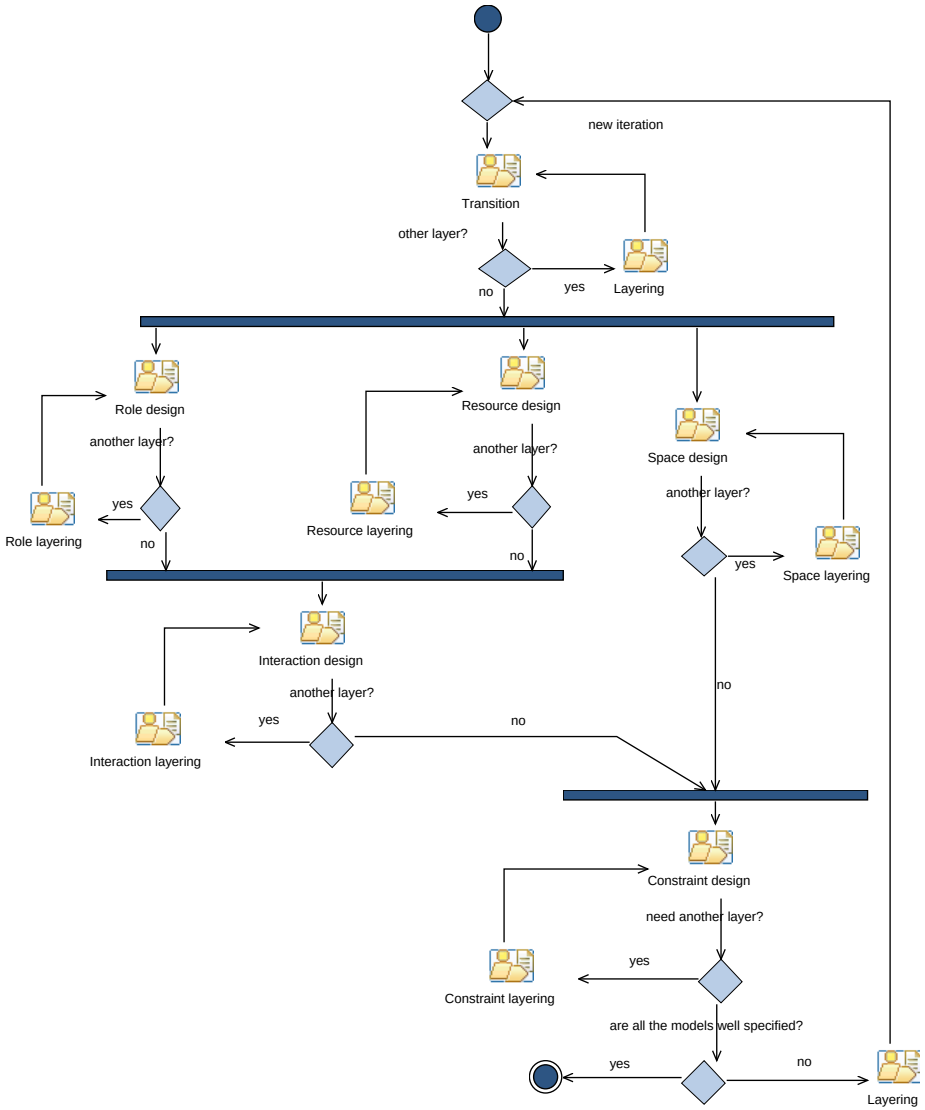
Analysis Process



Analysis Process

- *Moving from Requirements* activity: the abstractions identified in the previous step are mapped onto the abstractions adopted in this stage to generate the initial version of the Analysis models
- *Task analysis* activity: *task* is exploited for analysing the system's active part
- *Function analysis* activity: *function* is exploited for analysing the system's functional part
- *Topology analysis* activity: *topology* is exploited for analysing the topological constraints over the environment
- *Dependency analysis* activity: *dependency* is exploited for the analysis of the relations highlighted in the previous step and for the definition of new relationship among abstract entities identified in this stage

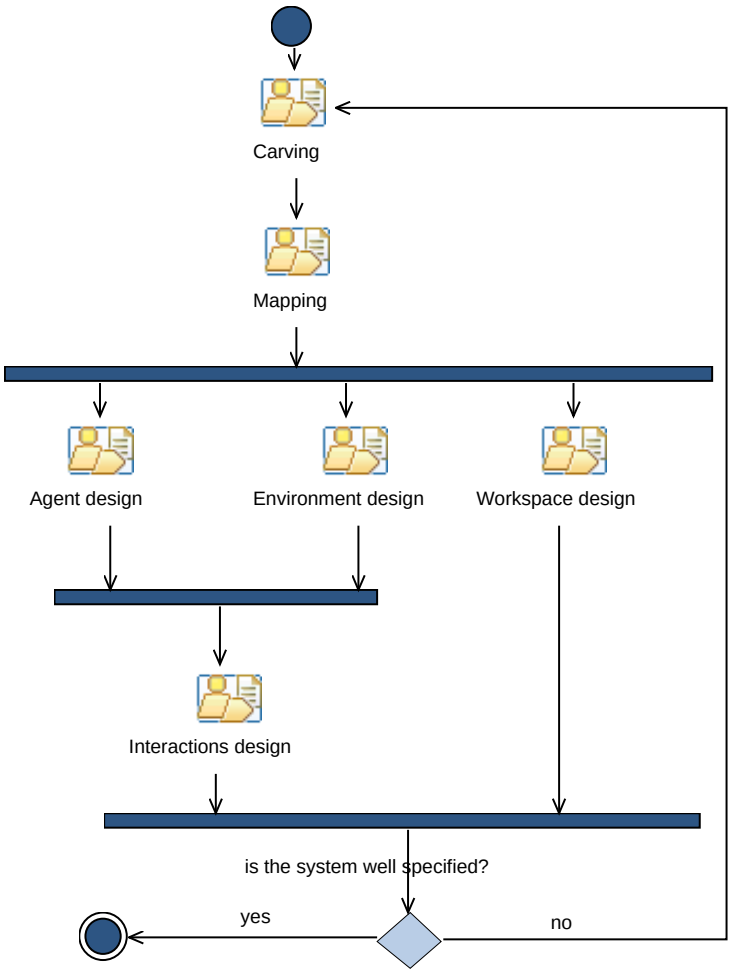
Architectural Design Process



Architectural Design Process

- *Transition* activity: the abstractions identified in the previous step are mapped onto the abstractions adopted in this stage so as to generate the initial version of the Architectural Design models
- *Role design* activity: assignment tasks to *roles* and identification of the *action*s necessary in order to achieve each specific tasks
- *Resource design* activity: assignment of functions of responsibilities for providing to *resources* and identification of the *operation*s necessary for providing each specific function.
- *Space design* activity: identification of the *space*s starting from the topology constraints analysed in the previous step
- *Interaction design* activity: identification of the *interaction*s that represent the acts of the interaction among roles, among resources and between roles and resources starting from the dependencies analysed in the previous step
- *Constraint design* activity: identification of the *rule*s that enable and bound the entities' behaviour starting from the dependencies analysed in the previous step

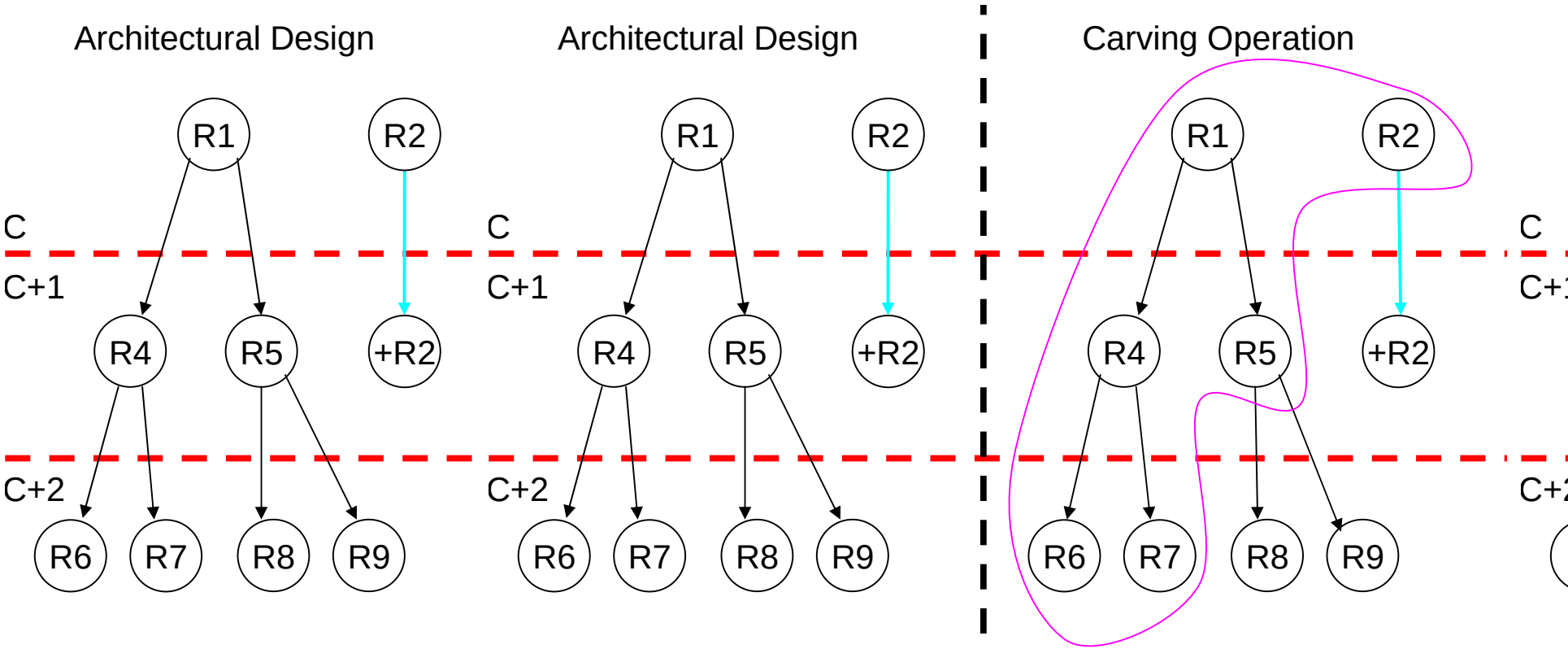
Detailed Design Process



Carving

- In the Architectural Design step our system could be potentially composed by all the layers detected in the previously steps
- Each complete layer represents a different system architecture...
- It is not possible to first design and then physically implement a system specified by different levels of abstraction
- So, for each entity, we need to choose the appropriate layer of representation...
- ... the chosen system architecture is "carved out" from all the possible architectures
- So, each carving represents a specific development...
- ... in an iterative incremental process we can made different carving in order to refine the system

Carving


Architectural Design
Architectural Design
Carving Operation
R1
R2
C
C+1
R4
R5
+R2
C+2
R6
R7
R8
R9

Detailed Design Process

- *Mapping* activity: the carved abstractions are mapped onto the abstractions adopted in this stage, thus generating the initial version of the Detailed Design models
- *Agent design* activity: design of *agent*s and *societies*
- *Environment design* activity: design of *artifact*s and *aggregate*s
- *Workspace design* activity: design of *workspace*s
- *Interactions design* activity: design of *uses*, *manifests*, *speaks to* and *links to*

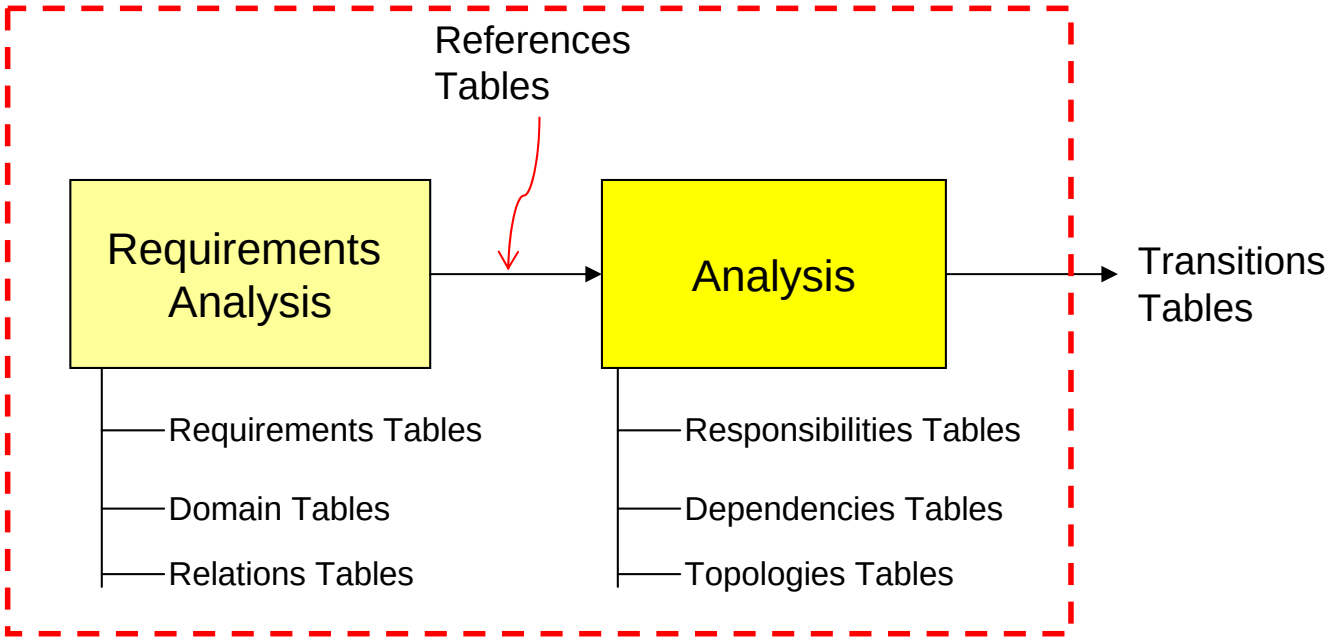
The SODA Notation

The design of a complex system

calls for both a clear and disciplined notation and tools for supporting the consistency checks

- SODA adopts a tabular representation
- Tables
  - clear way for analysing and designing the systems
  - very easy for automatic tools
  - very easy for consistency and completeness checkers

Analysis Phase: Tables



Requirements Analysis: Tabular Representation

- Requirements Tables: $(L)Ac_t$, $(L)AR_t$ and $(L)Re_t$

Actor	Description
actor name	*actor description*

Actor	Requirement
actor name	*requirement names*

Requirement	Description
requirement name	*requirement description*

- Domain Tables: $(L)EELS_t$ and $(L)LS_t$

External-Environment	Legacy-System
external-environment name	*Legacy-System names*

Legacy-System	Description
legacy-system name	*legacy-system description*

Requirements Analysis: Tabular Representation

- Requirements Tables define and describe the abstract entities tied to the concept of "requirement"
  - *Actor table* ($(L)Ac_t$) describes each single actor
  - *Actor-Requirement Table* ($(L)AR_t$) specifies the list of the requirements for each actors
  - *Requirement Table* ($(L)Re_t$) lists all the requirement and describe them.
- Domain Tables define and describe the abstract entities tied to the external environment
  - *ExternalEnvironment-LegacySystem Table* ($(L)EELS_t$) specifies the list of the contexts for external-environment
  - *Legacy-System Table* ($(L)LS_t$) lists all the contexts and describe them

Requirements Analysis: Tabular Representation

- Relations Tables: $(L)Rel_t$, $(L)RR_t$ and $(L)RLS_t$

Relation	Description
relation name	*relation description*

Requirement	Relation
requirement name	*relation names*

Legacy-System	Relation
legacy-system name	*relation names*

Requirements Analysis: Tabular Representation

- Relations Tables relate the abstract entities among them
  - *Relation Table*($(L)Rel_t$) lists all the relationship among abstract entities and provides a description to them
  - *Requirement-Relation Table*($(L)RR_t$) specifies the list of relations where requirement is involved
  - *LegacySystem-Relation Table* ($(L)LSR_t$) specifies the list of relations where context is involved

From Requirements Analysis to Analysis (II)

- References Tables in top- down order: $(L)RRT_t$, $(L)RRF_t$, $(L)RRTo_t$, $(L)RReqD_t$, $(L)RLSF_t$, $(L)RLST_t$, $(L)RRelD_t$

Requirement	Task
requirement name	*task names*

Requirement	Function
requirement name	*function names*

Requirement	Topology
requirement name	*topology names*

Requirement	Dependency
requirement name	*dependency names*

Legacy-System	Function
legacy-system name	*function names*

Legacy-System	Topology
legacy-system name	*topology names*

Relation	Dependency
relation name	*dependency names*

From Requirements Analysis to Analysis (III)

References Tables identify the relations among the abstractions of the requirement analysis phase and the abstractions used in analysis phase.

- *Reference Requirement-Task Table*($(L)RRT_t$) specifies the mapping between requirement and tasks
- *Reference Requirement-Function Table* ($(L)RRF_t$) specifies the mapping between requirement and resources
- *Reference Requirement-Topology table* ($(L)RRTo_t$) specifies the mapping between each requirement and the generated topologies
- *Reference Requirement-Dependency table* ($(L)RReqD_t$) specifies the mapping between each requirement and the generated dependencies
- *Reference LegacySystem-Function Table* ($(L)RLSF_t$) specifies the mapping between legacy-system and functions
- *Reference LegacySystem-Topology Table* ($(L)RLST_t$) specifies the mapping between legacy-system and topologies
- *Reference Relation-Dependency Table* ($(L)RRD_t$) specifies the mapping between relations and dependencies

Analysis: Tabular Representation

- Responsibilities Tables: $(L)T_t$ and $(L)F_t$

Task	Description
task name	*task description*

Function	Description
function name	*function description*

- Dependencies Tables: $(L)D_t$, $(L)TD_t$, $(L)FD_t$ and $(L)TopD_t$

Dependency	Description
dependency name	*dependency description*

Task	Dependency
task name	*dependency names*

Function	Dependency
function name	*dependency names*

Topology	Dependency
topology name	*dependency names*

Analysis: Tabular Representation

- Responsibilities Tables define and describe the abstract entities tied to the concept of "responsibility"
  - *Task Table* ($(L)T_t$) lists all the tasks and describes them
  - *Function Table* ($(L)F_t$) lists all the functions and describe them
- Dependencies Tables relate the abstract entities among them.
  - *Dependency Table* ($(L)D_t$) lists all the dependency among abstract entities and provides a description to them.
  - *Task-Dependency Table* ($(L)TD_t$) specifies the list of dependencies where task is involved.
  - *Function-Dependency Table* ($(L)FD_t$) specifies the list of dependencies where function is involved.
  - *Topology-Dependency table* ($(L)TopD_t$) specifies the list of dependencies where each topology is involved.

Analysis: Tabular Representation

- Topologies Tables in top-down order – $(L)Top_t$, $(L)TTop_t$, $(L)FTop_t$

Topology	Description
Topology name	*topology description*

Task	Topology
task name	*topology names*

Function	Topology
function name	*topology names*

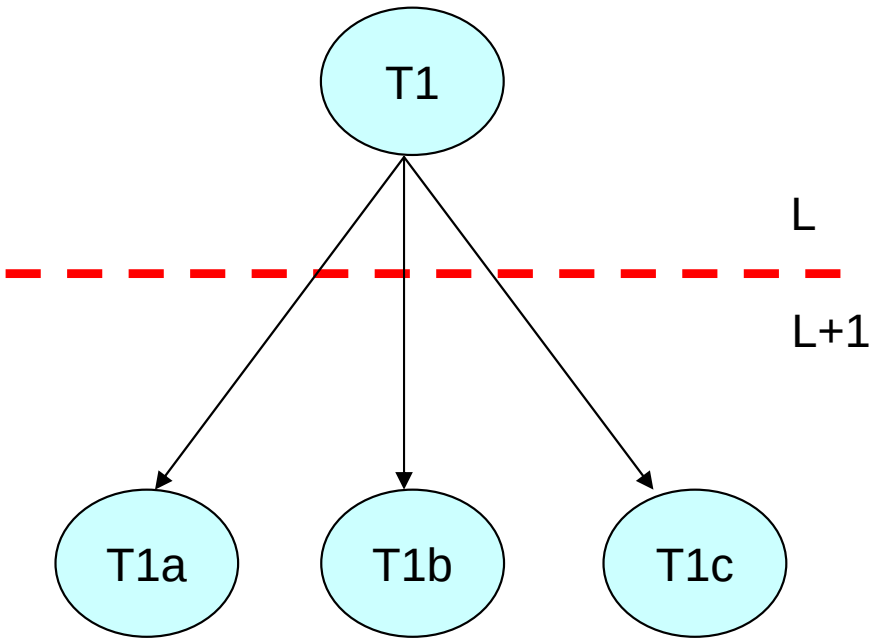
- Topologies Tables express the topological needs
  - *Topology Table* ($(L)Top_t$) lists all the topological requirements and provides a description to them.
  - *Task-Topology Table* ($(L)TTop_t$) specifies the list of topological requirements those influence the task.
  - *Function-Topology Table* ($(L)FTop_t$) specifies the list of topological requirements affected by the function.

Zooming: Tabular Representation

- Zooming Table: $(L)Z_t$

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

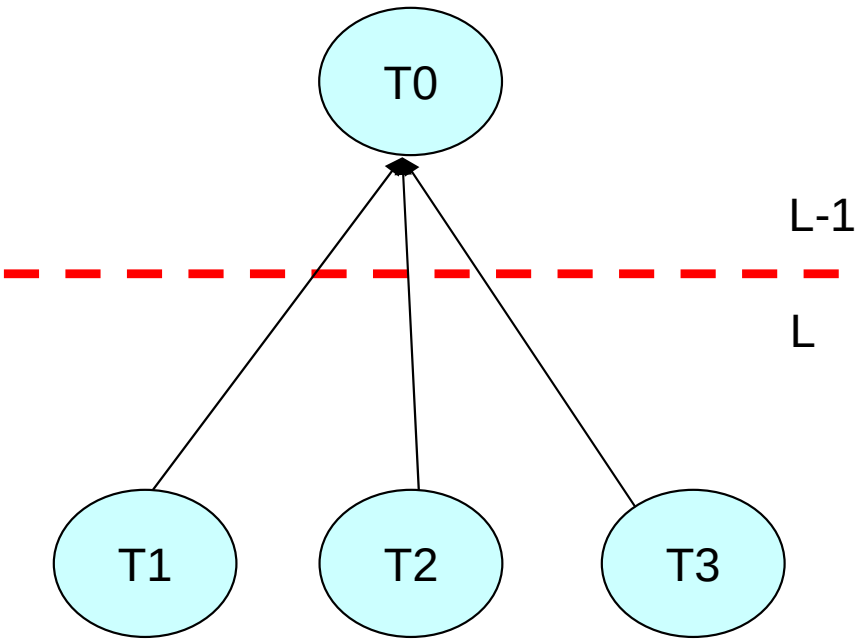
Example: In-zoom Task



- Zooming Table: $(L)Z_t$

Layer L	Layer L+1
T1	*T1a, T1b, T1c, . . .*

Example: Out-zoom Tasks



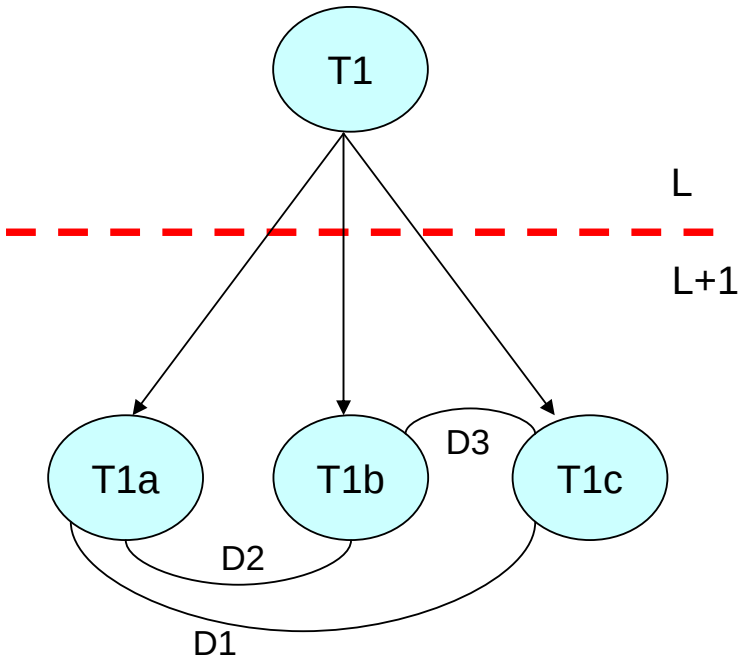
- Zooming Table: $(L)Z_t$

Layer L-1	Layer L
T0	*T1, T2, T3, . . .*

Remarks

- The organisational structure of the system is *implicitly managed by means of zooming relation*
- For example when we in-zoom a task, we obtain new tasks, new dependencies and potentially new functions and topologies.
- By means of new dependencies we can express all the social rules that allow to new task to work together to achieve the original tasks.
- In the same way in the architectural design phase when we in-zoom a role, we obtain new roles, new actions, new interactions and potentially new resources and operations. By means of new interactions we can express all the social rules that allow to new roles to work together to achieve the "social task(s)" assigned to the original role.

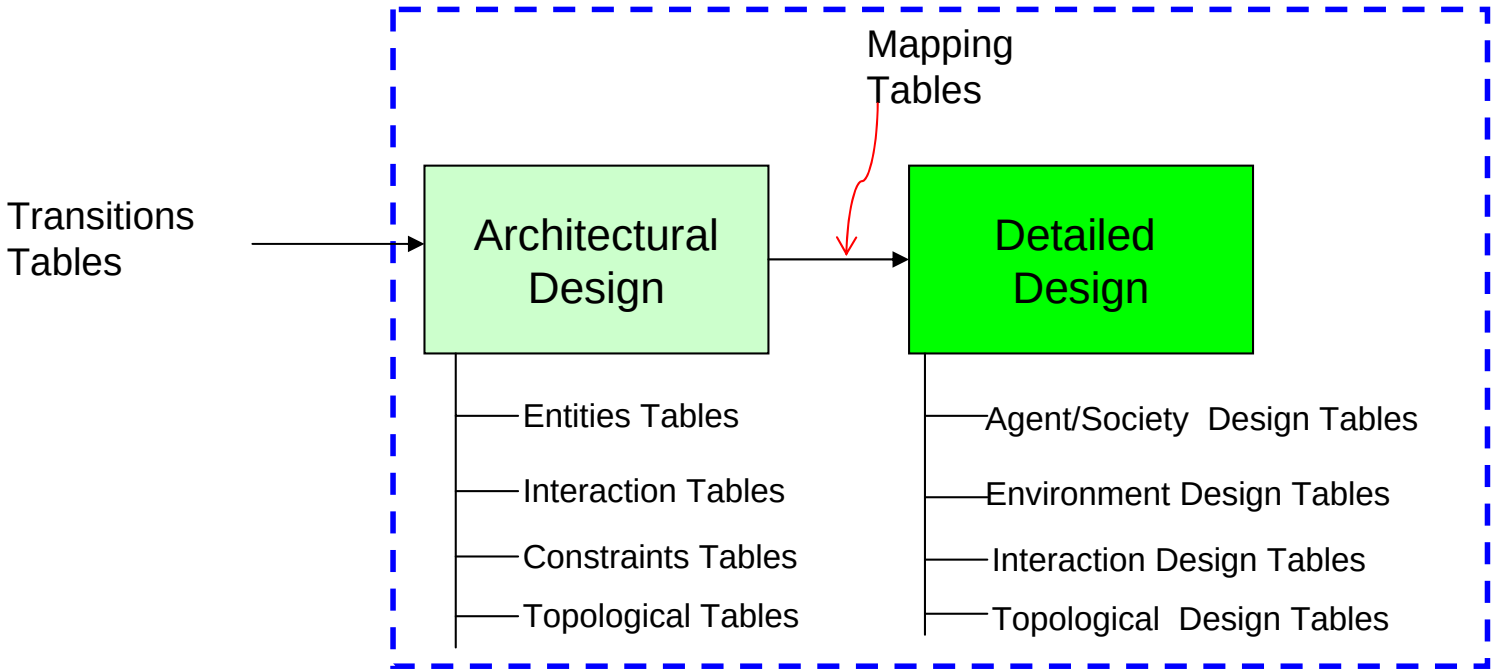
Complete Example: In-zoom Task



- Zooming Table: $(L)Z_t$

Layer L	Layer L+1
T1	*T1a, T1b, T1c* *D1, D2, D3*

Design Phase: Tables



From Analysis to Architectural Design

Transition Tables in top-down order – $(L)TRT_t$, $(L)TTA_t$, $(L)TRF_t$, $((L)TFO_t)$, $(L)TID_t$, $(L)TTopS_t$

Role	Task
role name	*task names*

Task	Action
task name	*action names*

Resource	Function
resource name	*function names*

Function	Operation
function name	*operation names*

Dependency	Interaction
dependency name	*interaction names*

Dependency	Rule
dependency name	*rule names*

Topology	Space
topology name	*space names*

From Analysis to Architectural Design

- Transition Tables identify the relations among the abstractions of the requirement analysis phase and the abstractions used in analysis phase
  - *Transition Role-Task Table* ($(L)TRT_t$) specifies the mapping between tasks and roles.
  - *Transition Task-Action table* ($(L)TTA_t$) relates tasks and actions.
  - *Transition Resource-Function Table* ($(L)TRF_t$) specifies the mapping between functions and resources.
  - *Transition Function-Operation table* ($(L)TFO_t$) relates functions and operations.
  - *Transition Interaction-Dependency Table* ($(L)TID_t$) specifies the mapping between dependencies and interaction.
  - *Transition Rule-Dependency table* ($(L)TRuD_t$) maps dependencies onto rules
  - *Transition Topology-Space Table* ($(L)TTopS_t$) specifies the mapping between topologies and spaces.

Architectural Design: Tabular Representation

- Entities Tables in top-down order – $(L)A_t$, $(L)O_t$, $(L)RA_t$, $(L)RO_t$

Action	Description
action name	*description*

Operation	Description
operation name	*description*

Role	Action
role name	*action names*

Resource	Operation
resource name	*operation names*

Architectural Design: Tabular Representation

- The Entities Tables that describe roles and resources of the system
  - *Action Table*($(L)A_t$) specifies the actions that roles could be able to execute and describes them the mapping between tasks and roles.
  - *Operation Table* ($(L)O_t$) specifies the operations that resources could provide and describes them the mapping between tasks and roles.
  - *Role-Action Table* ($(L)RA_t$) specifies the list of actions that a specific role is able to do.
  - *Resource-Operation Table* ($(L)RO_t$) specifies the list of operations that a specific resource is able to provide.

Architectural Design: Tabular Representation

- Interactions Tables in top-down order – $(L)I_t$, $(L)AcI_t$, $(L)OpI_t$

Interaction	Description
interaction name	*description*

Action	Interaction
action name	*interaction names*

Operation	Interaction
operation name	*interaction names*

Architectural Design: Tabular Representation

- The Interactions Tables that describe the interaction where roles and resources are involved
  - *Interaction Table* ($(L)I_t$) specifies the interactions and describes them. the mapping between tasks and roles.
  - *Action-Interaction table* ($(L)AcI_t$) specifies the interactions where each action is involved.
  - *Operation-Interaction table* ($(L)OpI_t$) specifies the interactions where each operation is involved.

Architectural Design: Tabular Representation

- Constraints Tables in top-down order – $(L)Ru_t$, $(L)IRu_t$, $(L)ReRu_t$, $(L)RoRu_t$, $(L)SRu_t$

Rule	Description
rule name	*description*

Interaction	Rule
interaction name	*rule names*

Resource	Rule
resource name	*rule names*

Role	Rule
role name	*rule names*

Space	Rule
space name	*rule names*

Architectural Design: Tabular Representation

- The Constraints Tables describe the constraints over the entities behaviours
  - *Rule table* ($(L)Ru_t$) defines the single rule.
  - *Rule-Interaction table* ($(L)IRu_t$) specifies the constraints over the interactions.
  - *Resource-Rule table* ($(L)ReRu_t$) specifies the rules where each resource is involved.
  - *Role-Rule table* ($(L)RoRu_t$) specifies the rules where each role is involved.
  - *Space-Rule table* ($(L)SRu_t$) specifies the rules where each space is involved.

Architectural Design: Tabular Representation

- Topological Tables in top-down order – $(L)S_t$, $(L)SC_t$, $(L)SRe_t$ and $(L)SRo_t$

Space	Description
space name	*description*

Space	Connection
space name	*space names*

Resource	Space
resource name	*space names*

Role	Space
role name	*space names*

Architectural Design: Tabular Representation

- Topological Tables
  - *Space table* ($(L)S_t$) specifies the spaces and describes them.
  - *Space-Connection table* ($(L)SC_t$) shows the connections between spaces at the same layer of abstraction (the hierarchical relations among spaces are managed by means of zooming table)
  - *Space-Resource table* ($(L)SRe_t$) shows the allocation of the resources to spaces. A resource could be allocated in several different spaces. In particular, a single, distributed resource can in principle be used to model a distributed service, accessible from more nodes of the network.
  - *Space-Role table* ($(L)sRo_t$) shows the list of space that the roles can perceive in the system.

From Architectural Design to Detailed Design

- Mapping Tables in top-down order – MAR_t, MSR_t, $MAAc_t$, $MArR_t$, $MAggR_t$, $MArOp_t$, $MArRu_t$, MSW_t, MIU_t, MIM_t, $MISp_t$, MIL_t

Agent	Role
agent name	*role names*

Society	Role
society name	*role name*

(Individual) Artifact	Action
artifact name	*action names*

(Environmental) Artifact	Resource
artifact name	*resource names*

Aggregate	Resource
aggregate name	*resource name*

(Environmental) Artifact	Operation
artifact name	*operation names*

From Architectural Design to Detailed Design

Rule	Artifact
rule name	*artifact names*

Workspace	Space
workspace name	*space names*

Interaction	Use
interaction name	*use names*

Interaction	Manifest
interaction name	*manifest names*

Interaction	Speak to
interaction name	*speak names*

Interaction	Linked to
interaction name	*linked names*

From Architectural Design to Detailed Design I

- Mapping Tables
  - *Mapping Agent-Role table (MAR_t)* maps roles onto agents
  - *Mapping Society-Role table (MSR_t)* maps role onto society
  - *Mapping Artifact-Action table ($MAAc_t$)* maps actions onto individual artifacts
  - *Mapping Artifact-Resource table ($MArR_t$)* maps resources onto artifacts
  - *Mapping Aggregate-Resource table ($MAggR_t$)* maps resources onto aggregate
  - *Mapping Artifact-Operation table ($MArOp_t$)* maps operation onto environmental artifacts
  - *Mapping Artifact-Rule table ($MArRu_t$)* maps the rules specified in the Architectural Design onto the artifacts that implement and enforce them
  - *Mapping Artifact-Operation table (MSW_t)* maps spaces onto workspaces

From Architectural Design to Detailed Design II

- *Mapping Interaction-Uses table (MIU_t)* maps interactions onto uses
- *Mapping Interaction-Manifests table (MIM_t)* maps interactions onto manifests
- *Mapping Interaction-SpeaksTo table ($MISp_t$)* maps interactions onto speaks to
- *Mapping Interaction-LinksTo table (MIL_t)* maps interactions onto links to

Detailed Design: Tabular Representation

- Agent/Society Design Tables in top-down order – $(L)AA_t$, $(L)SA_t$, $(L)SAr_t$

Agent	(Individual) Artifact
agent name	*artifact names*

Society	Agent
Society name	*agent names*

Society	Artifact
society name	*artifact names*

Detailed Design: Tabular Representation

- Agent/Society Design Tables
  - *Agent-Artifact Table* ($(L)AA_t$) specifies the (individual) artifacts related to agents.
  - *Society-Agent Table* ($(L)SA_t$) specifies which agents work in the society
  - *Society-Artifact Table* ($(L)SAr_t$) specifies the artifacts related to societies.

Detailed Design: Tabular Representation

- Environment Design Tables in top-down order – AUI_t, $AggArt_t$, $AggAge_t$

Artifact	Usage Interface
artifact name	*list of operations*

Aggregate	Artifact
aggregate name	*artifact names*

Aggregate	Agent
aggregate name	*agent names*

Detailed Design: Tabular Representation

- Environment Design Tables
  - *Artifact-UsageInterface Table* ($(L)AUI_t$) specifies the operations provided by artifacts.
  - *Aggregate-Artifact table* ($AggArt_t$) lists the artifacts belonging to a specific aggregate.
  - *Aggregate-Agent table* ($AggAge_t$) lists the agents belonging to a specific aggregate

Detailed Design: Tabular Representation

- Interaction Design Tables in top-down order – UP_t, $UAge_t$, SP_t, $SpAge_t$, MP_t, $MArt_t$, LP_t, $LArt_t$

Use	Protocol
uses name	*protocol description*

Agent	Use
agent name	*uses names*

Speak To	Protocol
speaks name	*protocol description*

Agent	Speaks To
agent name	*speaks names*

Manifests	Protocol
manifests name	*protocol description*

Artifact	Manifests
artifact name	*manifests names*

Links To	Protocol
links name	*protocol description*

Artifact	Links To
artifact name	*links names*

Detailed Design: Tabular Representation

- Interaction Design Tables concern the design of interactions among entities
  - *Use-Protocol table (UP_t)* details the protocols for each "uses interaction"
  - *Uses-Agent table ($UAge_t$)* specifies the "uses" where each agent is involved
  - *SpeaksTo-Protocol table (SP_t)* details the protocols for each "speaks to interaction"
  - *SpeaksTo-Agent table ($SpAge_t$)* specifies the "speaks to" where each agent is involved
  - *Manifests-Protocol table (MP_t)* details the protocols for each "manifests interaction"
  - *Manifests-Artifact table ($MArt_t$)* specifies the "manifests" where each artifact is involved
  - *LinksTo-Protocol table (LP_t)* details the protocols for each "links to" interaction
  - *LinksTo-Artifact table ($LArt_t$)* specifies the "links to" where each artifact is involved

Detailed Design: Tabular Representation

- Topological Design Tables in top-down order: $(L)W_t$, $(L)WC_t$, $(L)WArt_t$ and $(L)WA_t$

Workspace	Description
workspace name	*description*

Workspace	Connection
workspace name	*workspace names*

Workspace	Artifact
workspace name	*artifact names*

Agent	Workspace
agent name	*workspace names*

Detailed Design: Tabular Representation

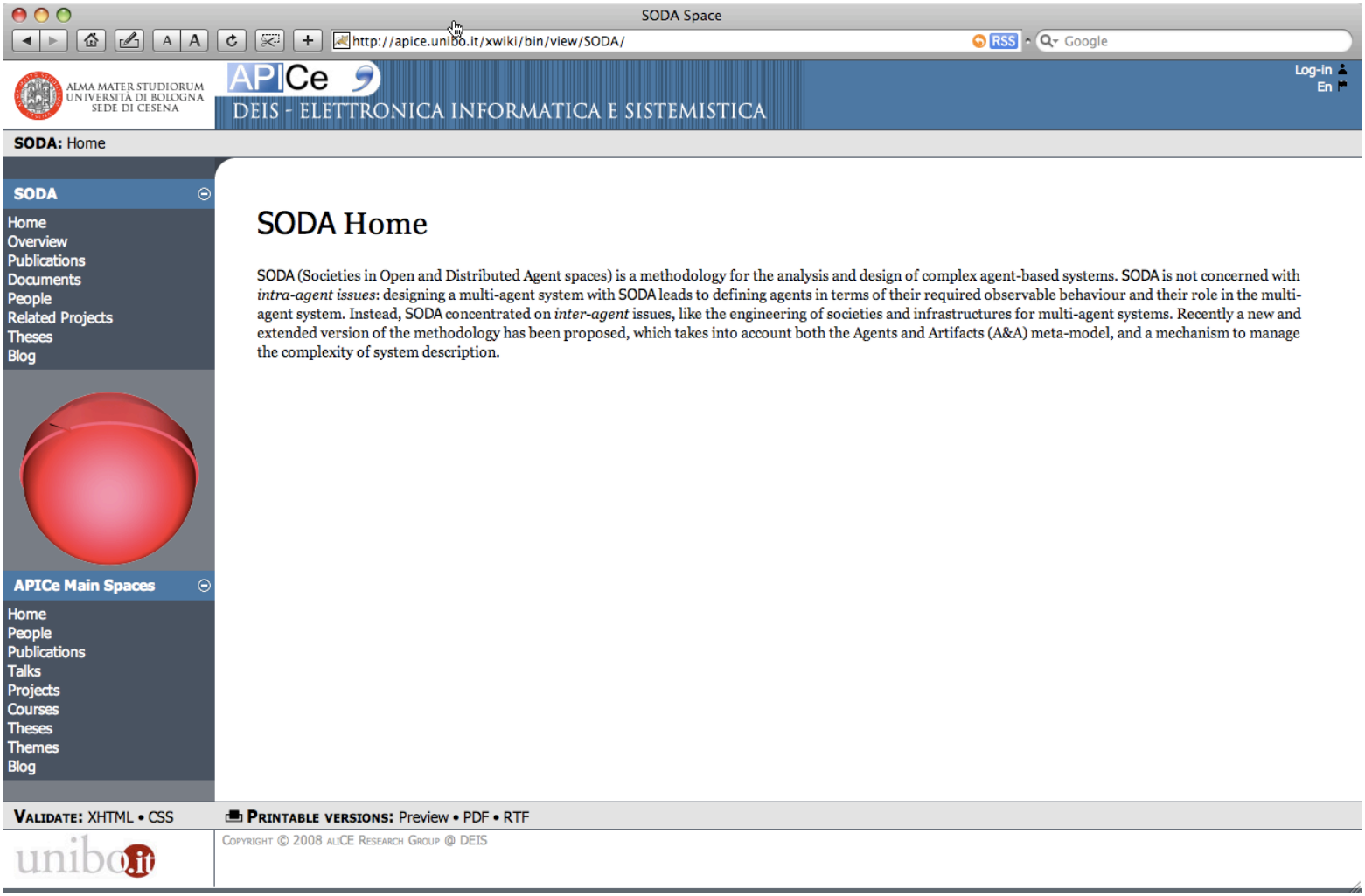
- Topological Design Tables describe the structure of the environment:
  - *Workspace table* ($(L)W_t$) describes the workspaces
  - *Workspace-Connection table* ($(L)WC_t$) shows the connections among the workspaces
  - *Workspace-Artifact table* ($(L)WArt_t$) shows the allocation of the artifacts to workspaces
  - *Workspace-Agent table* ($(L)WA_t$) lists the workspaces that each agent can perceive

Conclusions & Future Work

- SODA allows engineers to
  - design societies
  - design environments
  - support the complexity of system description (layering principle)
- Future works
  - testing SODA
  - refining the meta-model
  - building the tools
  - extracting fragments from SODA according to IEEE-FIPA Method Engineering
  - integration between methodologies' and infrastructures' processes

Web Site

http://soda.apice.unibo.it



Available Theses & Projects

On the web site

- Kit&SODA: A Toolkit for SODA
- Graph&SODA: A Graphical Tool for SODA
- Check&SODA: A Consistency Checker for SODA
- Testing of SODA in different case studies

Bibliography I

Bernon, C., Cossentino, M., Gleizes, M. P., Turci, P., and Zambonelli, F. (2004).
A study of some multi-agent meta-models.
In Odell, J., Giorgini, P., and Müller, J. P., editors, *AOSE*, volume 3382 of *Lecture Notes in Computer Science*, pages 62–77. Springer.

Cernuzzi, L., Cossentino, M., and Zambonelli, F. (2005).
Process models for agent-based development.
Engineering Applications of Artificial Intelligence, 18(2):205–222.

Fuggetta, A. (2000).
Software process: a roadmap.
In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 25–34, New York, NY, USA. ACM Press.

Bibliography II


Ghezzi, C., Jazayeri, M., and Mandrioli, D. (2002).
Foundamental of Software Engineering.
Prentice Hall, second edition.


Molesini, A., Denti, E., and Omicini, A. (2005).
MAS meta-models on test: UML vs. OPM in the SODA case study.
In Pěchouček, M., Petta, P., and Varga, L. Z., editors, *Multi-Agent Systems and Applications IV*, volume 3690 of *LNAI*, pages 163–172. Springer.
4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05), Budapest, Hungary, 15–17 September 2005, Proceedings.

Bibliography III


Molesini, A., Denti, E., and Omicini, A. (2008a).
Agent-based conference management: a case study in SODA.
International Journal of Agent-Oriented Software Engineering.
In press.


Molesini, A., Nardini, E., Denti, E., and Omicini, A. (2008b).
Advancing object-oriented standards toward agent-oriented methodologies: SPEM 2.0 on SODA.
In *9th Workshop "From Objects to Agents" (WOA 2008) – Evolution of Agent Development: Methodologies, Tools, Platforms and Languages*, Palermo, Italy.

Bibliography IV


Molesini, A., Omicini, A., Ricci, A., and Denti, E. (2006).
Zooming multi-agent systems.
In Müller, J. P. and Zambonelli, F., editors, *Agent-Oriented Software Engineering VI*, volume 3950 of *LNCS*, pages 81–93. Springer.
6th International Workshop (AOSE 2005), Utrecht, The Netherlands, 25–26 July 2005. Revised and Invited Papers.


Object Management Group (2008).
Software & Systems Process Engineering Meta-Model Specification 2.0.
`http://www.omg.org/spec/SPEM/2.0/PDF`.

Bibliography V


Omicini, A. (2001).
SODA: Societies and infrastructures in the analysis and design of agent-based systems.
In Ciancarini, P. and Wooldridge, M. J., editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 185–193. Springer-Verlag.
1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.


Simon, H. A. (1996).
The Sciences of the Artificial.
The MIT Press, 3rd edition.

The SODA AOSE Methodology

Multiagent Systems LS

Sistemi Multiagente LS

Andrea Omicini & Ambra Molesini
{andrea.omicini, ambra.molesini}@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2009/2009