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 2007/2008

Overview

SODA Foundations: A&A Meta-Model, Layering Principle

- The Agents and Artifacts Meta-model
- Principles and Mechanisms

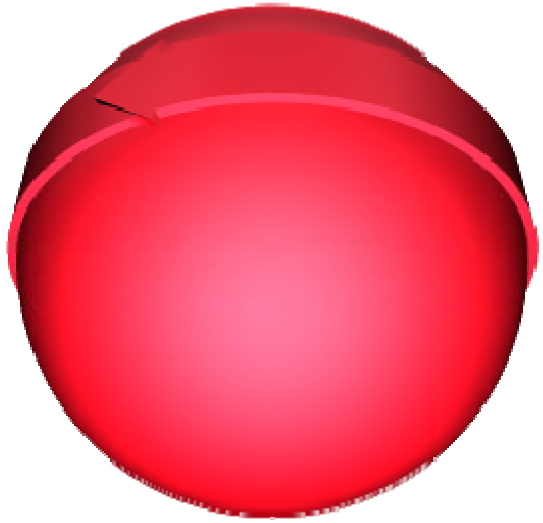
The SODA Methodology

- Analysis Phase
- Design Phase
- Web Resources and Conclusions

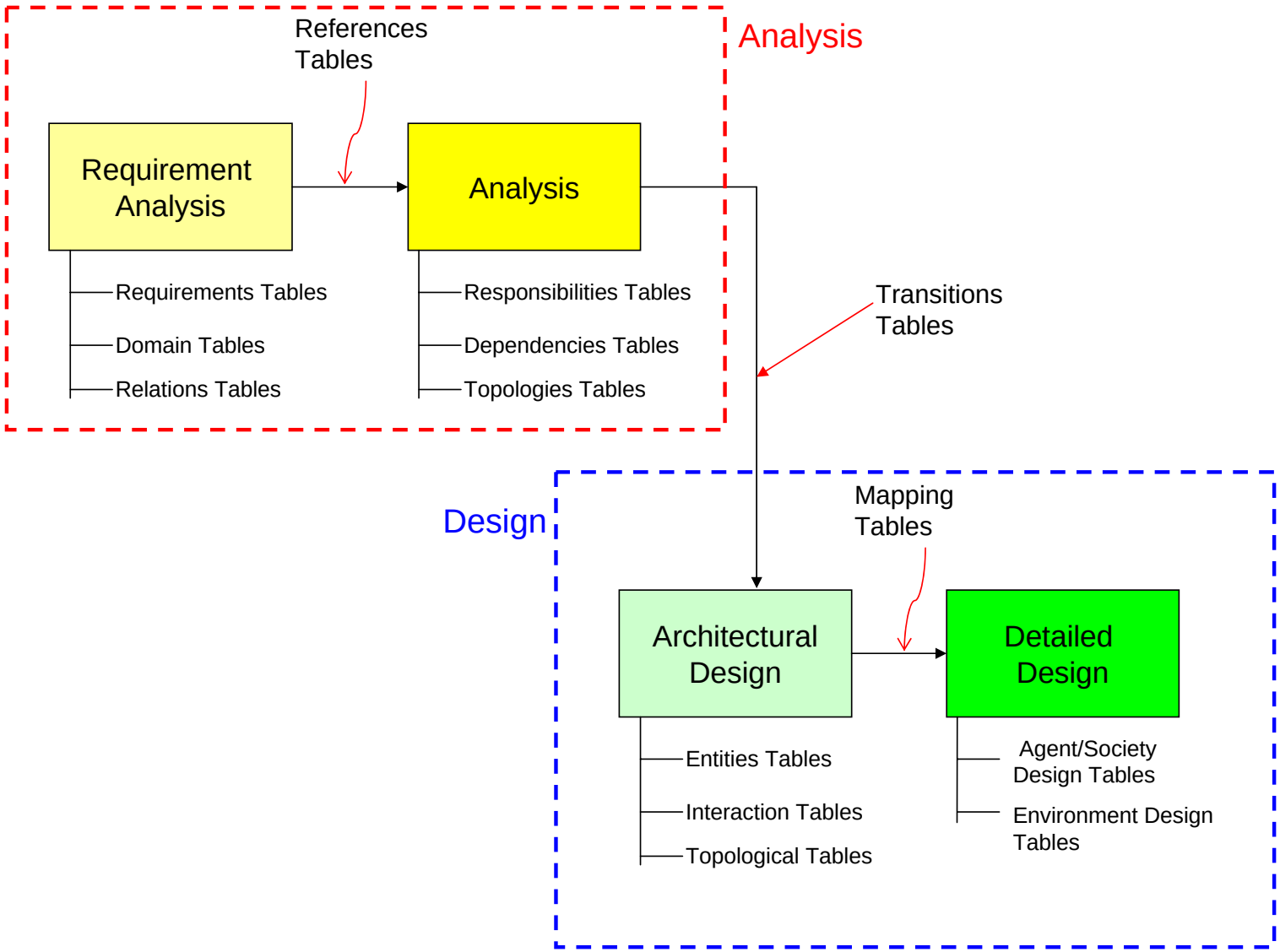
SODA: Societies in Open and Distributed Agent spaces

SODA ...

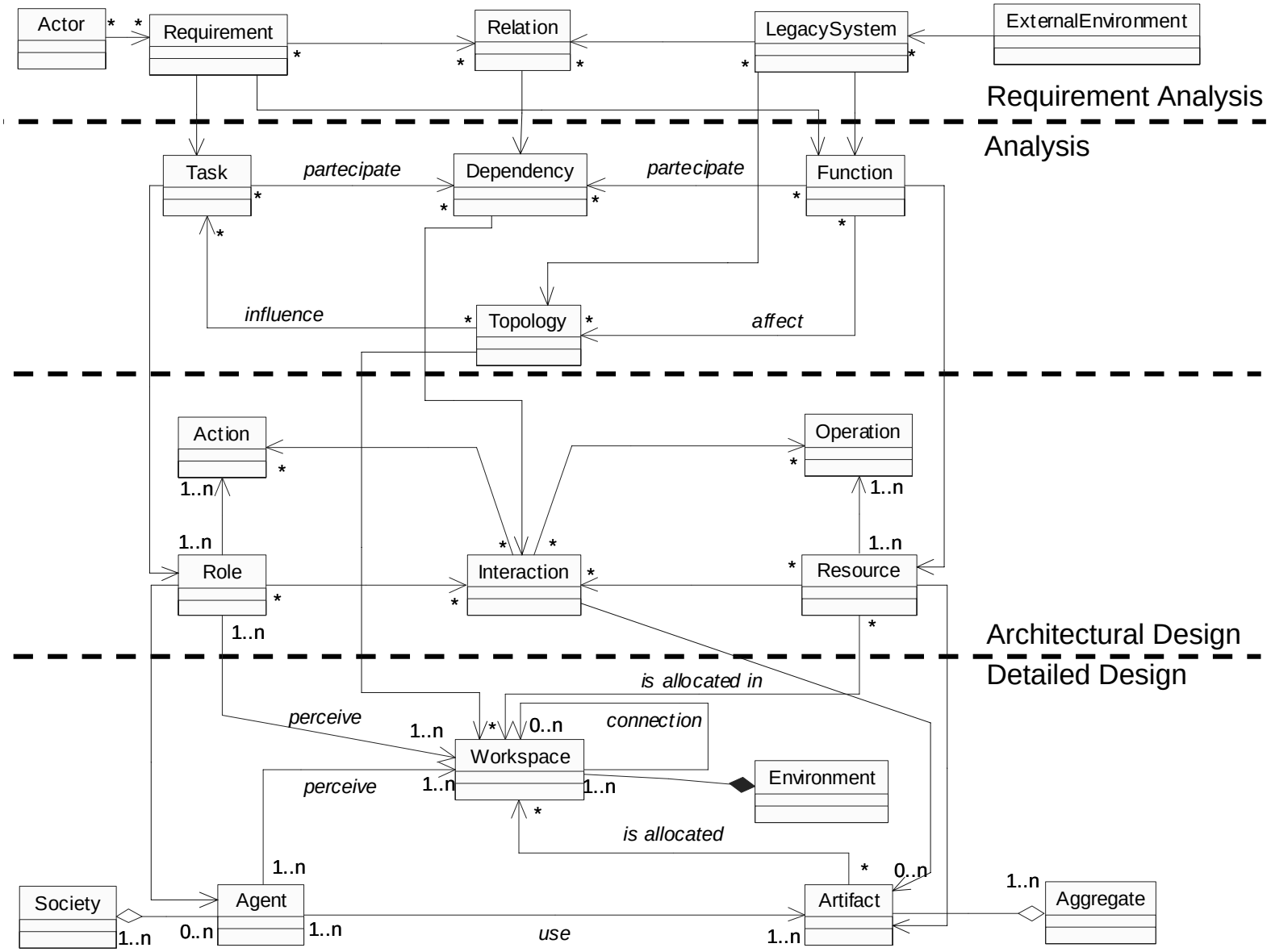
- ... is an agent-oriented methodology for the analysis and design of agent-based systems
- ... focuses on inter-agent issues, like the engineering of societies and environment for MAS [Omicini, 2001]
- ... adopts agents and artifacts – after the A&A meta-model – as the main building blocks for MAS development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



SODA: Overview



SODA: Meta-model



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 [Omicini et al., 2006]

- 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

Cognitive Features of Artifacts

- In order to promote its own use by cognitive agents, an artifact should expose

  Usage interface — the set of operations provided by an artifact

  Operating instructions — are a description of the procedure an agent has to follow to meaningfully interact with an artifact over time

  Function description — a description of the functionality provided by the artifact, which agents can use essentially for artifact selection

Features for Artifact Classification

- Other features of artifacts, among the many others, are

  Inspectability — the state of an artifact, the laws governing its behaviour might be all or partially inspectable by agents

  Malleability — the behaviour of an artifact could be modifiable at execution time in order to adapt to the changing needs or mutable external conditions of a MAS.

  Linkability — in order to scale up with complexity it might be useful to compose artifacts, by allowing artifacts to invoke operations on other artifacts

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

Meta-model Ingredients

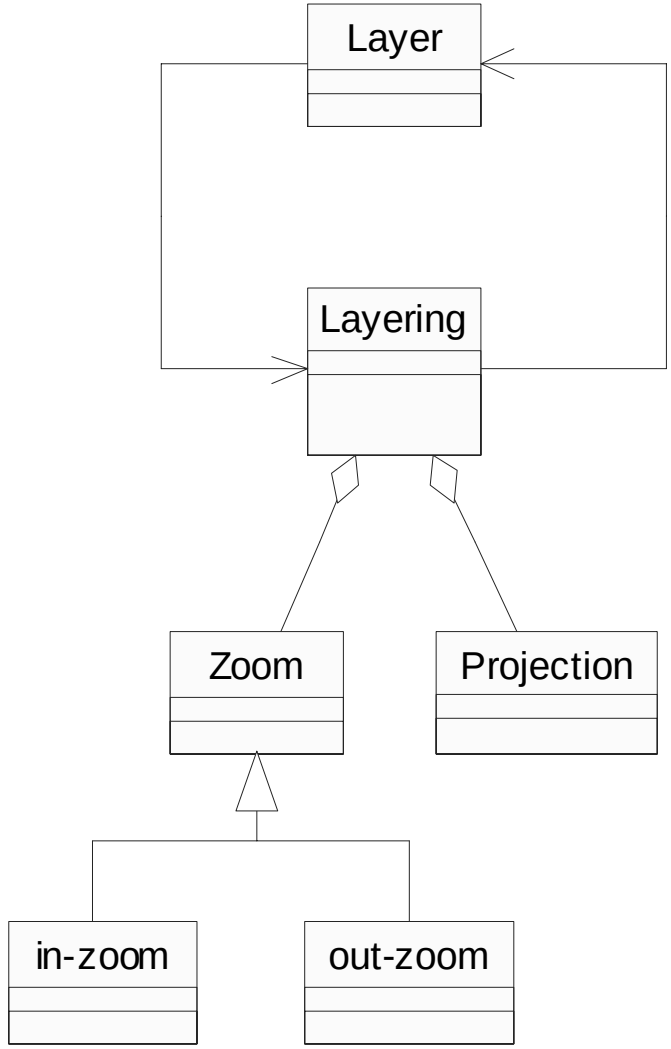
- Agents & artifacts lead to new ontological meta-model for MAS
- Artifacts allow to
  - model the environment as a first-class entity
  - engineer the space of interaction among agents (not only mere conversations between agents, but complex agent interaction patterns)
  - enrich MAS design with social/organisational structure, topological models, as well as (complex) security models
- In particular in SODA [Molesini et al., 2006a]...
  - agents model individual/social activities
  - artifacts *glue* agents together, as well as MAS and the enviromnent
    - they mediate between individual agents and MAS
    - they build up agent societies
    - they wrap up and bring MAS resources to the cognitive level of agents

Layering, Systems, and 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

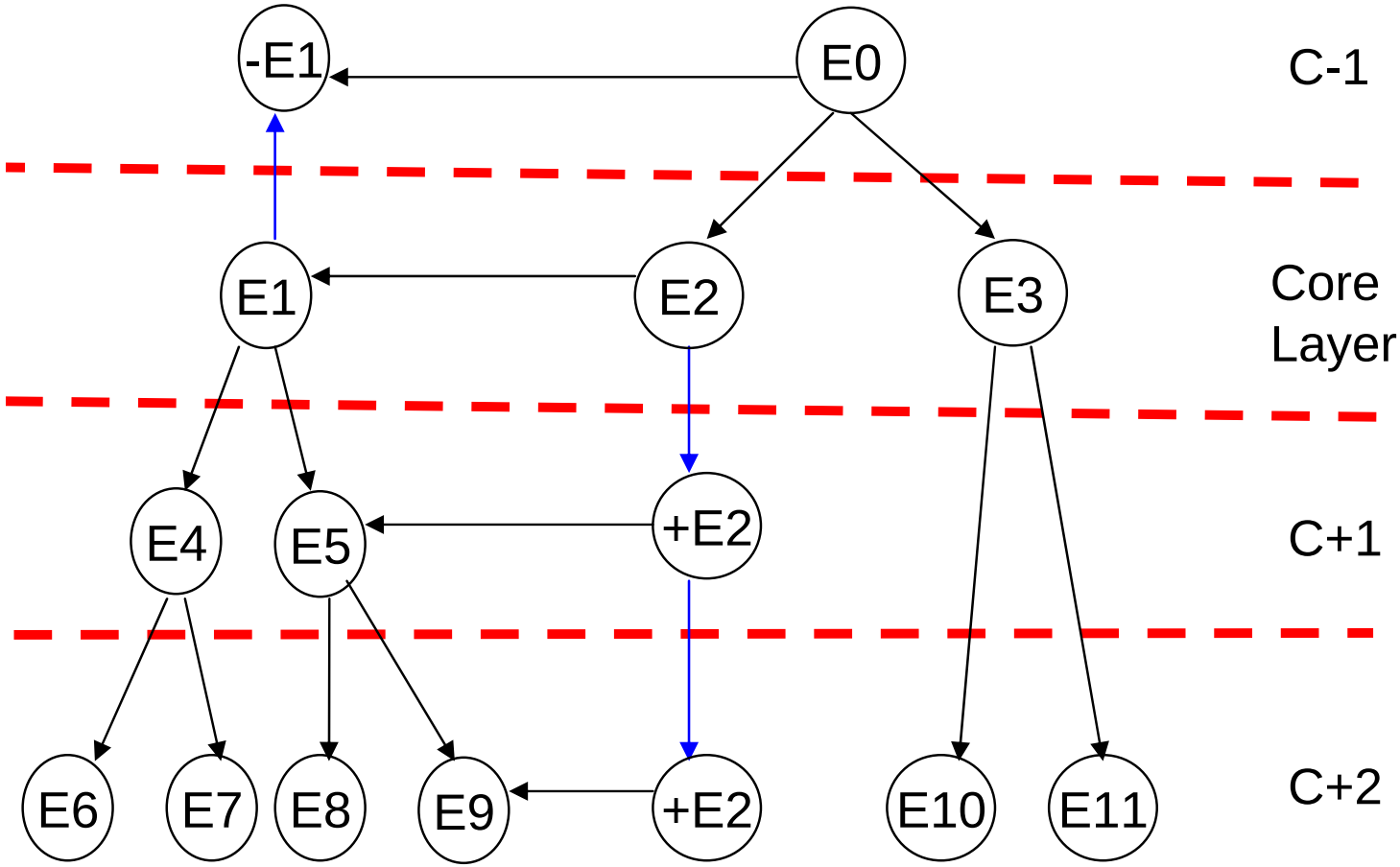
- The layering principle is achieved by means of the zoom and projection mechanisms [Molesini et al., 2006b, Molesini et al., 2007]
- Two kinds of zoom
  - in-zoom — from an abstract to a more detailed layer
  - out-zoom — from a detailed to a more abstract layer
- The *projection mechanism* projects entities from one to another layer



Layering Principle

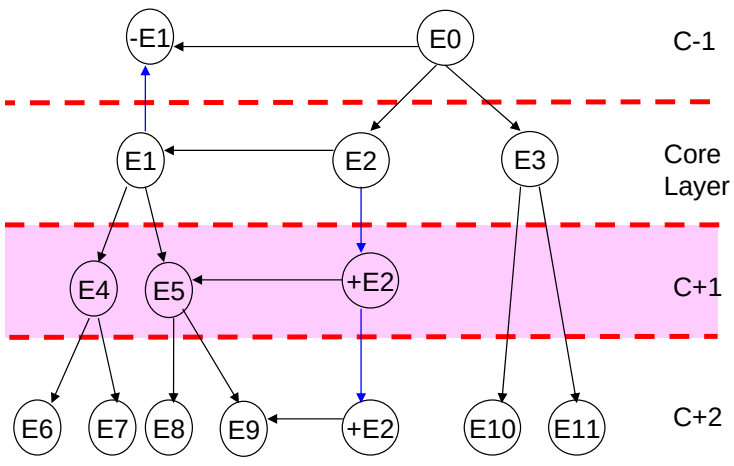
- In general, when working with SODA, we start from a certain layer, we could call *core layer*, and it is labelled with "c"
- The core layer is always **complete**
- In the other layer we find only the in/out zoomed entities and the projection entities.
- The in-zoomed layers are labelled with "c+1",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

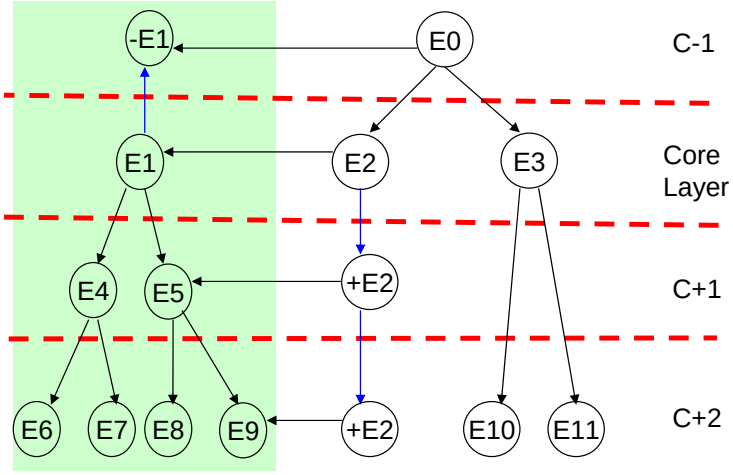


System's views

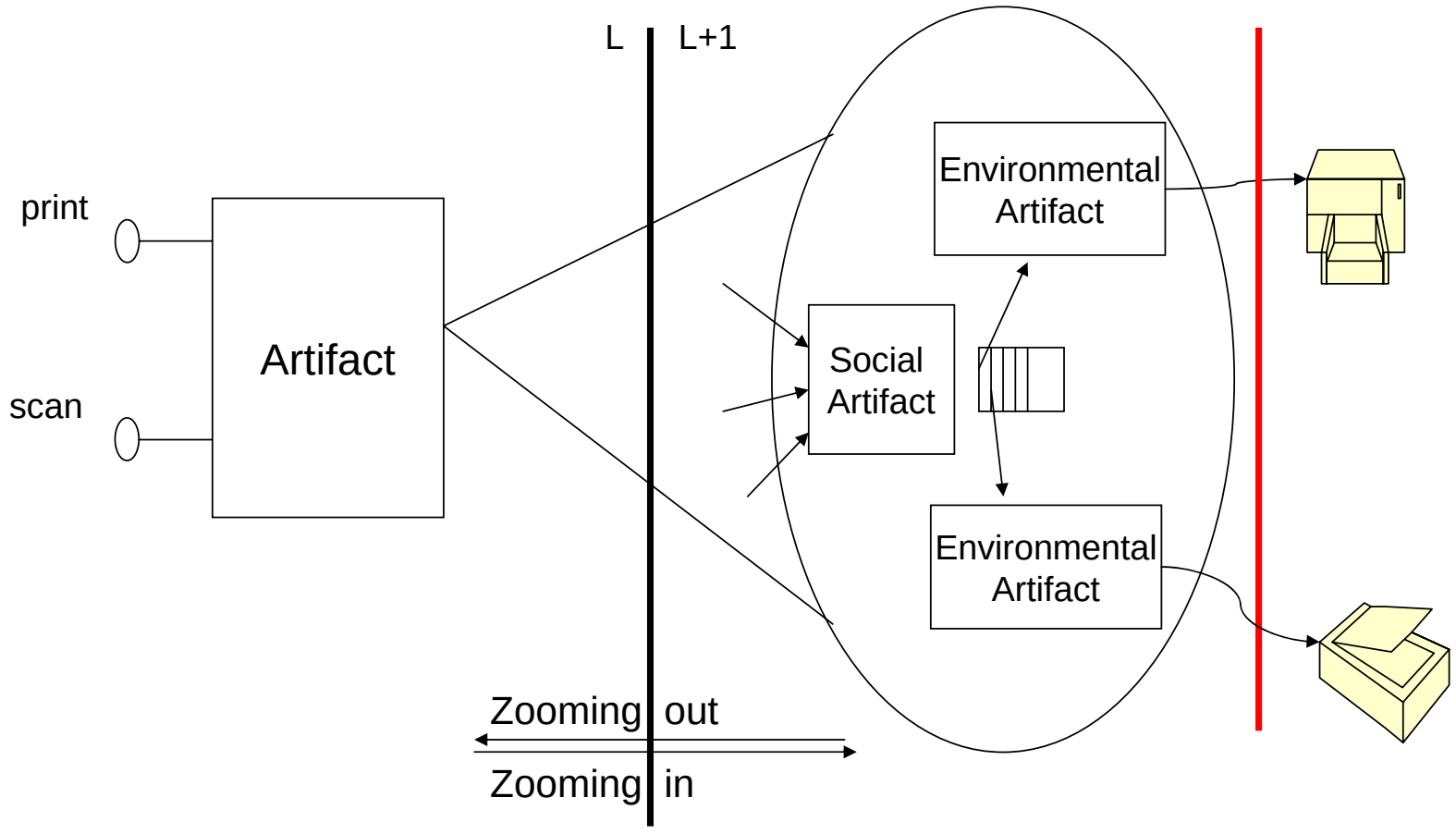
Horizontal view: analyse the system in one level of detail



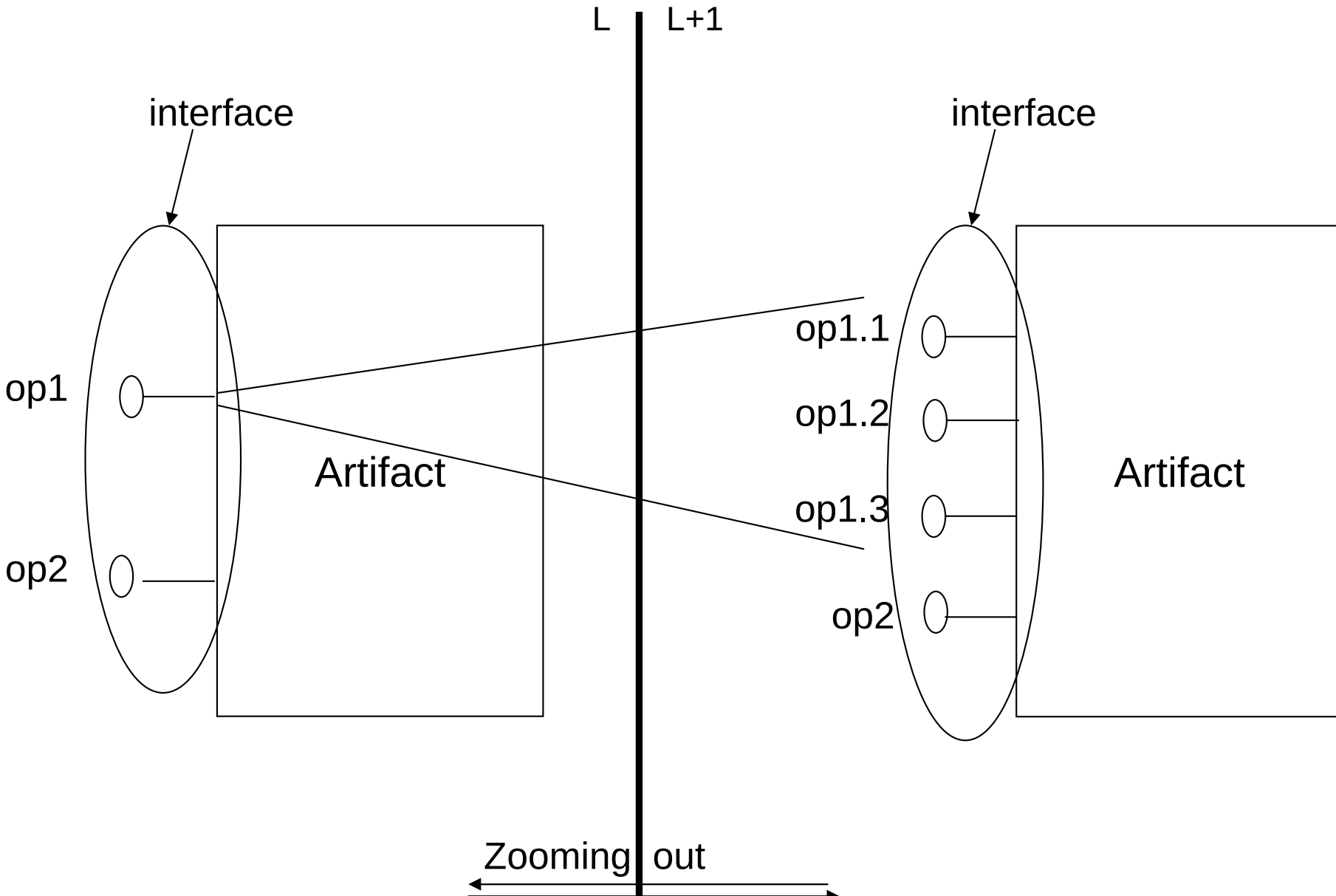
Vertical view: analyse one kind of abstract entity



Zooming Artifact 1/2



Zooming Artifact 2/2



SODA

SODA is organised in two phase and each of them is composed of two steps

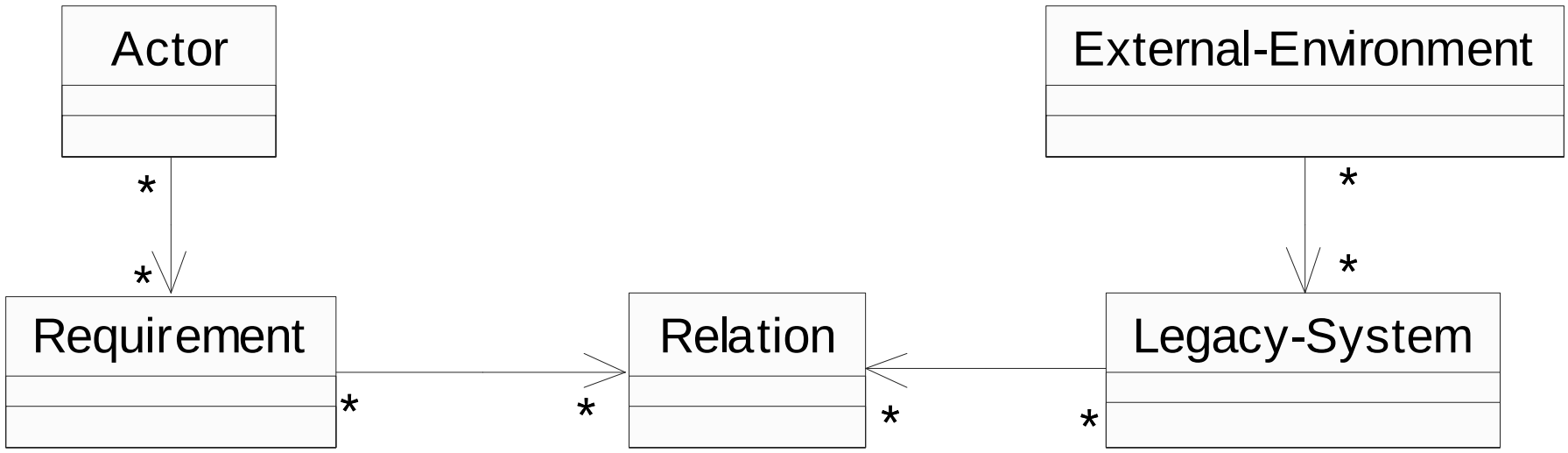
- *Analysis phase* —
  - *Requirement Analysis* — the system's requirements and the external environment are analysed and modelled.
  - *Analysis* — the system's requirements are modelled in terms of tasks, functions, topologies and dependencies
- *Design phase* —
  - *Architectural Design phase*: in this phase we analyse the solution domain, the system is modelled in terms of roles, resources, actions, operations, interactions, workspaces and environment
  - *Detailed Design phase* — in this phase we design the system in terms of agents, societies, artifacts, composition of artifacts, workspaces and environment

Requirements

The requirements can be categorised in:

- *Functional Requirement* — statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- *Non-Functional Requirement* — constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- *Domain Requirement* — requirements that come from the application domain of the system and that reflect characteristics of that domain.

Requirement Analysis Meta-model



Requirement 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.

Requirement Analysis: Tabular Representation

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

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*

Requirement Analysis: Tabular Representation

- Requirements Tables define and describe the abstract entities tied to the concept of "requirement"
  - *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

Requirement Analysis: Tabular Representation

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

Relation	Description
relation name	*relation description*

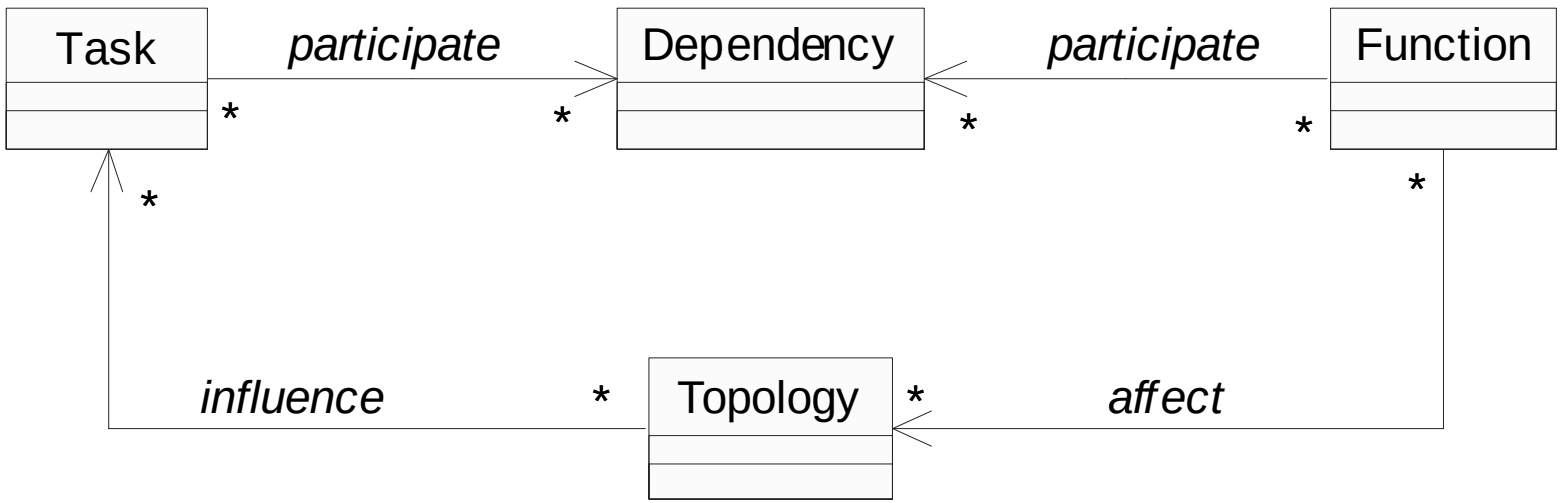
Requirement	Relation
requirement name	*relation names*

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

Requirement 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

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

From Requirement Analysis to Analysis (I)

- References Tables in top- down order: $(L)RRT_t$, $(L)RRF_t$, $(L)RLSF_t$, $(L)RLST_t$ and $(L)RRD_t$

Requirement	Task
requirement name	*task names*

Requirement	Function
requirement name	*function names*

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

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

Relation	Dependency
relation name	*dependency names*

From Requirement Analysis to Analysis (II)

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 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: $(L)D_t$, $(L)TD_t$ and $(L)FD_t$

Dependency	Description
dependency name	*dependency description*

Task	Dependency
task name	*dependency names*

Function	Dependency
function name	*dependency names*

Analysis: Tabular Representation I

- 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

Analysis: Tabular Representation II

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

Analysis: Tabular Representation III

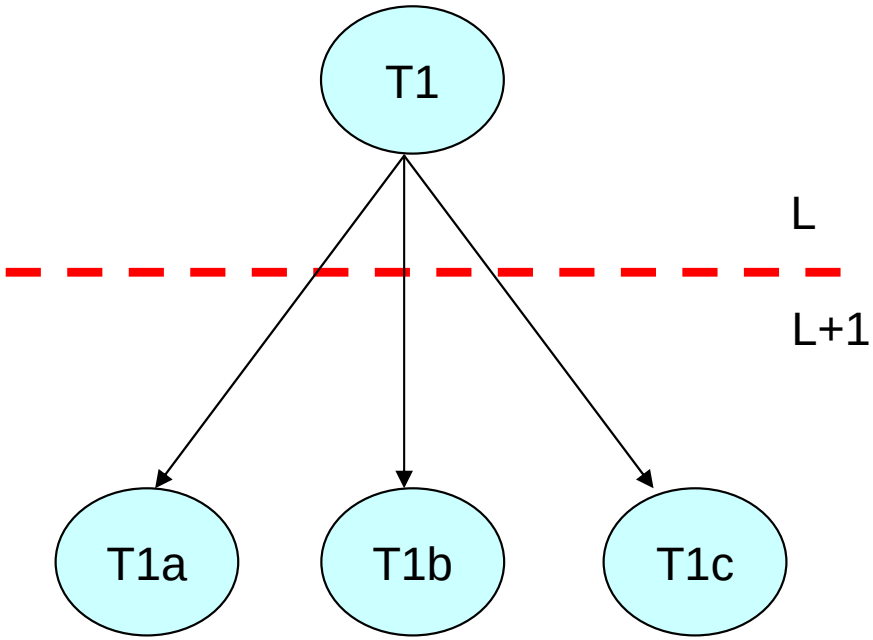
- *Topology Table* ($(L)Top_t$) lists all the topological requirements and provides a description to them.
- *Task-Topology Table* ($(L)TTop_t$) specifies the list of topological requirements those influence the task.
- *Function-Topology Table* ($(L)FTop_t$) specifies the list of topological requirements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

Zooming: Tabular Representation

- Zooming Table: $(L)Z_t$

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

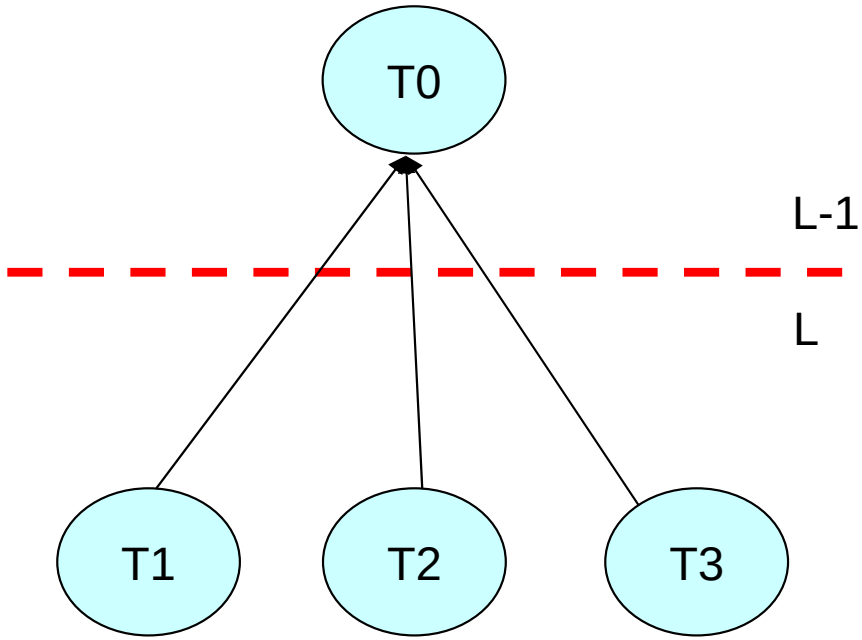
Example: In-zoom task



- Zooming Table: $(L)Z_t$

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

Example: Out-zoom tasks



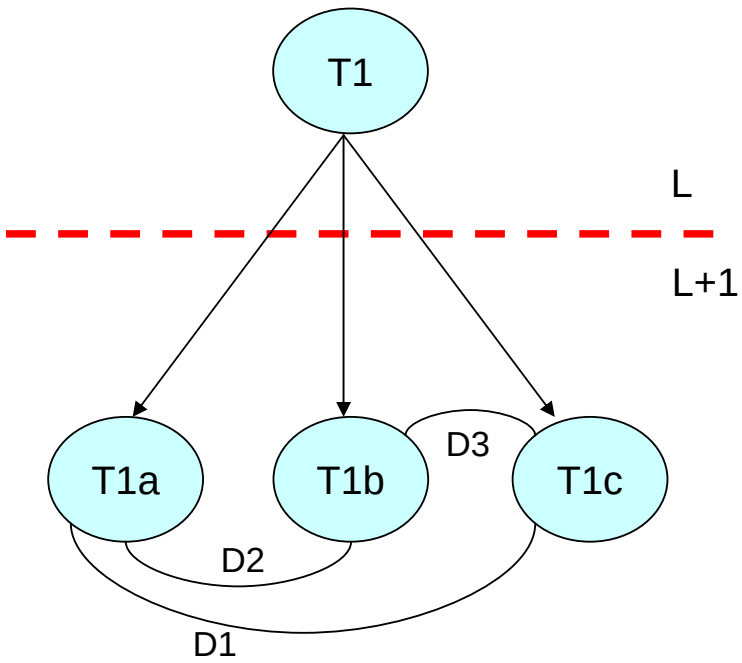
- Zooming Table: $(L)Z_t$

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.

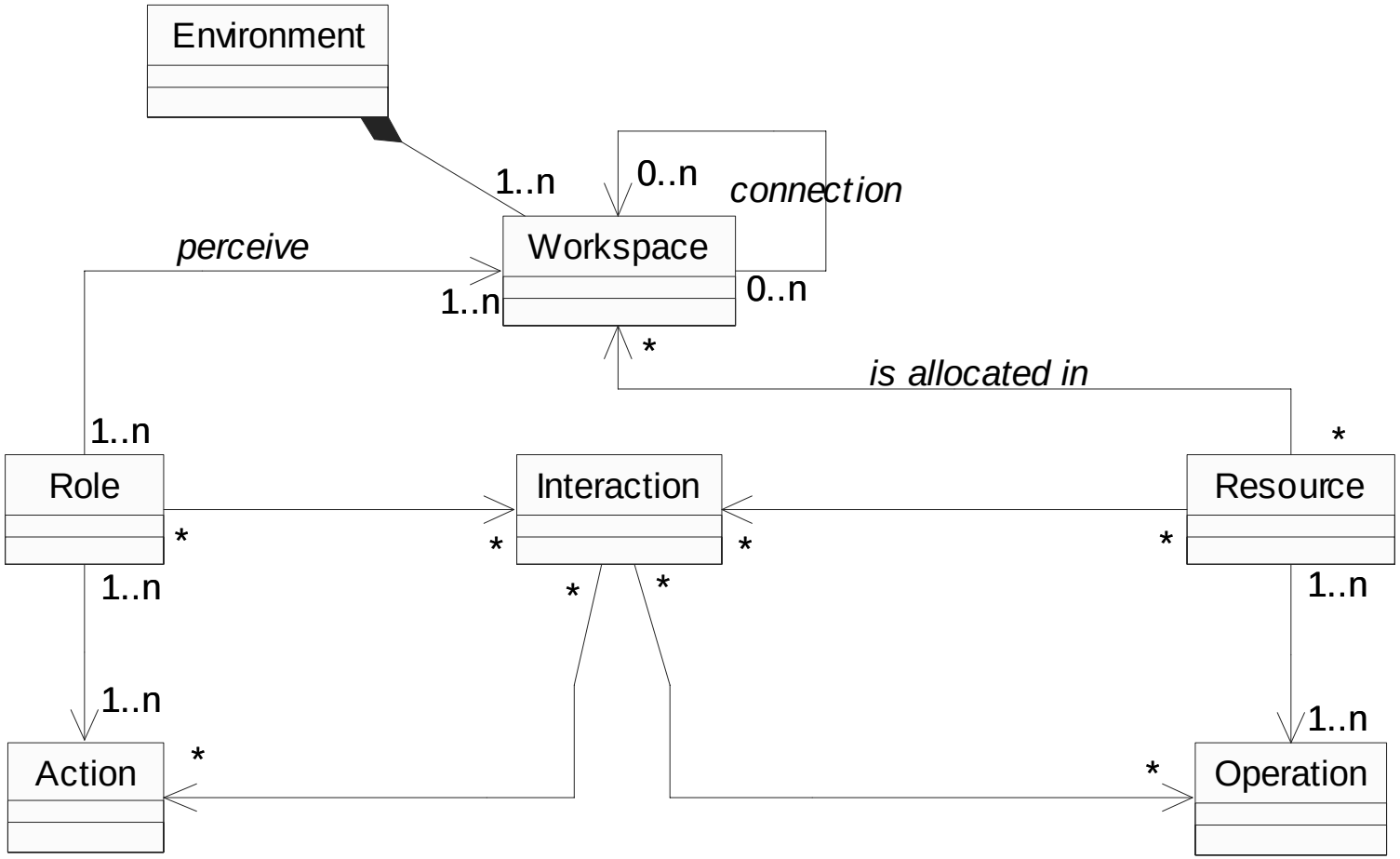
Complete Example: in-zoom task



- Zooming Table: $(L)Z_t$

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

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 — is defined as "rules" aimed to enable and bound both the behaviour of the abstract entities and the space of interactions. Bounds could be expressed by means of authorisation, prohibition and obligation concepts. Enabling could be expressed by means of rules that tie the actions with operations that support them

Environment — is the environment of the system

Workspace — is a conceptual locus in the environment

From Analysis to Architectural Design I

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

Role	Task
role name	*task names*

Resource	Function
resource name	*function names*

Dependency	Interaction
dependency name	*interaction names*

Topology	Workspace
topology name	*workspace names*

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

From Analysis to Architectural Design II

- *Transition Role-Task Table* ($(L)TRT_t$) specifies the mapping between tasks and roles.
- *Transition Resource-Function Table* ($(L)TRF_t$) specifies the mapping between functions and resources.
- *Transition Interaction-Dependency Table* ($(L)TID_t$) specifies the mapping between dependencies and interaction.
- *Transition Topology-Workplace Table* ($(L)TTopW_t$) specifies the mapping between topologies and workplaces.

Architectural Design: Tabular Representation I

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*

The Entities Tables that describe roles and resources of the system

Architectural Design: Tabular Representation II

- *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.

Interactions Tables in top-down order – $(L)I_t$, $(L)RoI_t$, $(L)ReI_t$

Architectural Design: Tabular Representation III

Interaction	Description
interaction name	*description*

Role	Interaction
role name	*interaction names*

Resource	Interaction
resource name	*interaction names*

The Interactions Tables that describe the interaction where roles and resources are involved

Architectural Design: Tabular Representation IV

- *Interaction Table* ($(L)I_t$) specifies the interactions and describes them. the mapping between tasks and roles.
- *Role-Interaction Table* ($(L)RoI_t$) specifies the list of interactions where roles are involved
- *Resource-Interaction Table* ($(L)ReI_t$) specifies the list of interactions where resources are involved

Topological Tables in top-down order – $(L)W_t$, $(L)WC_t$, $(L)WRe_t$ and $(L)WRo_t$

Architectural Design: Tabular Representation V

Workspace	Description
workspace name	*description*

Workspace	Connection
workspace name	*workspace names*

Workspace	Resource
workspace name	*resource names*

Role	Workspace
role name	*workspace names*

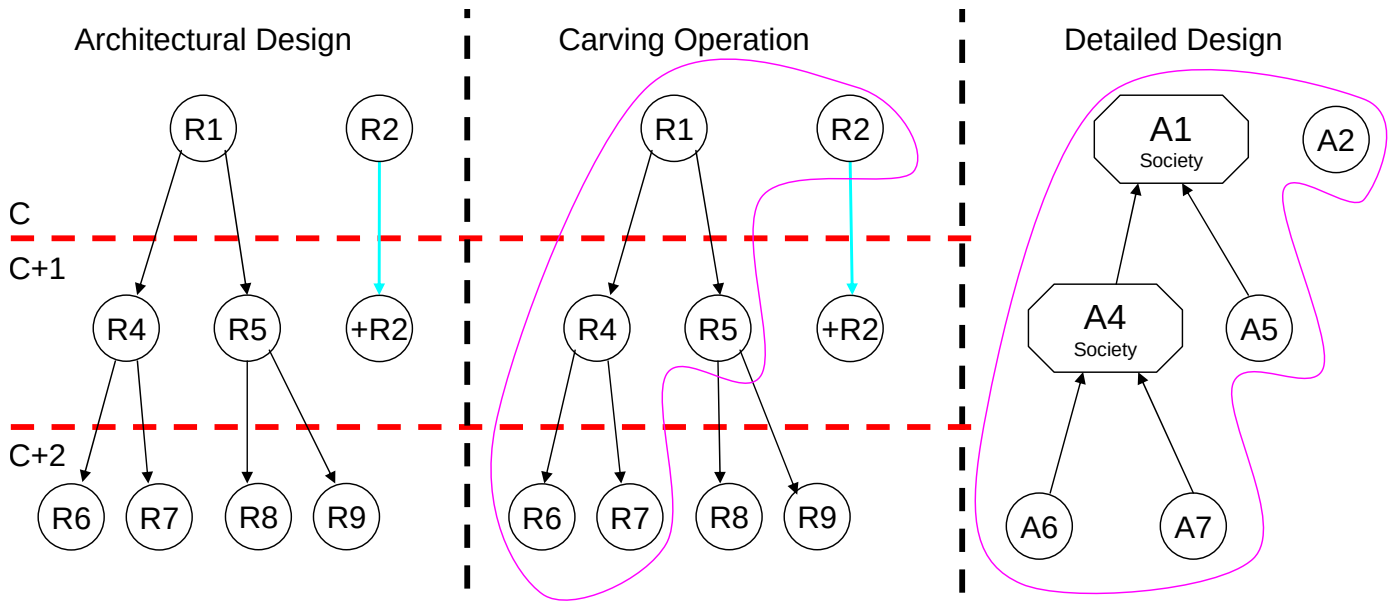
Topological Tables

Architectural Design: Tabular Representation VI

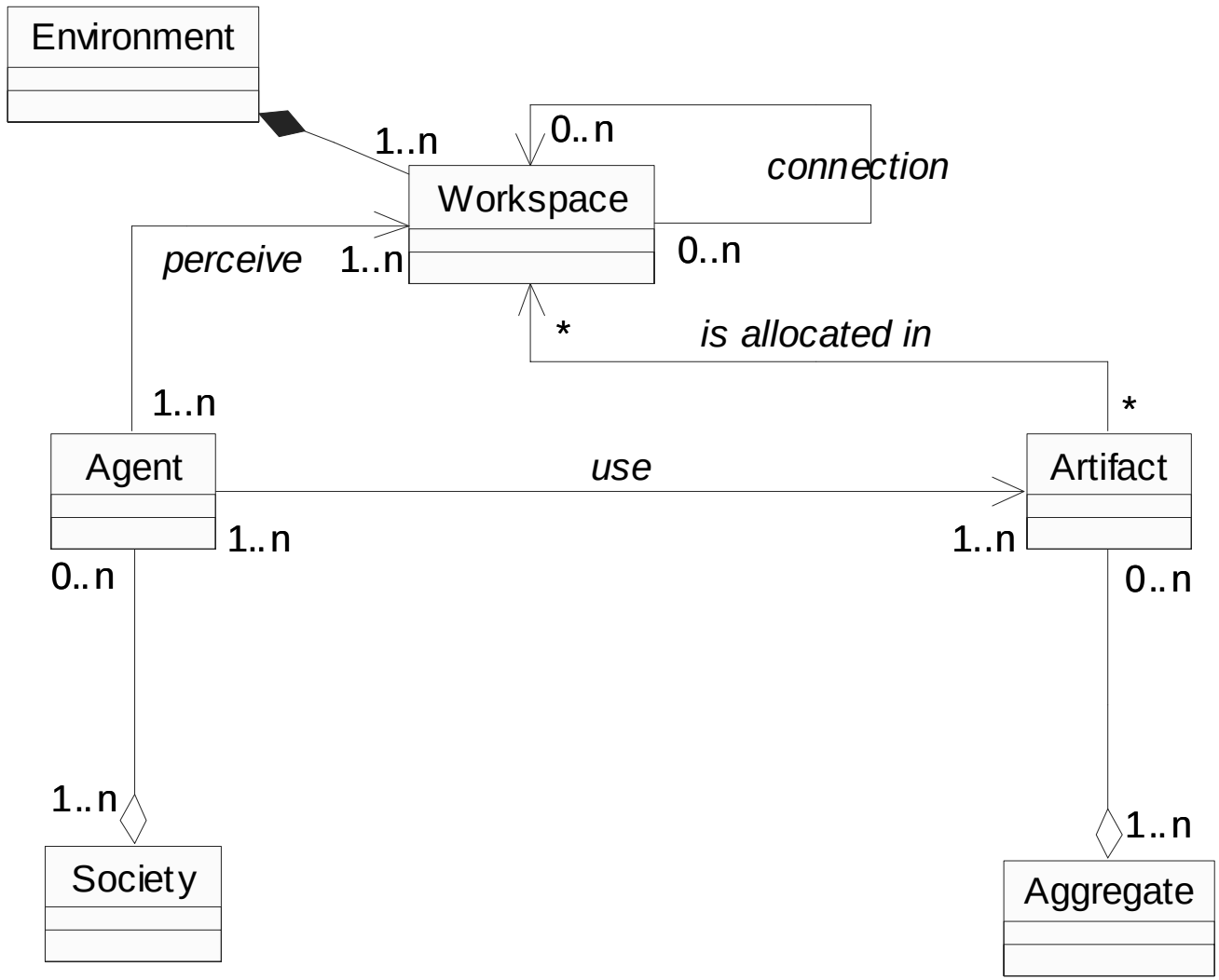
- *Workspace Table* ($(L)W_t$) specifies the workspaces and describes them.
- *Workspace-Connection Table* ($(L)WC_t$) shows the connections between workspaces at the same layer of abstraction (the hierarchical relations among workspaces are managed by means of zooming table)
- *Workspace-Resource Table* ($(L)WRe_t$) shows the allocation of the resources to workspaces. A resource could be allocated in several different workspaces. In particular, a single, distributed resource can in principle be used to model a distributed service, accessible from more nodes of the network.
- *Workspace-Role Table* ($(L)WRo_t$) shows the list of workspace that the roles can perceive in the system.

Design Views

- In this phase potentially our system could be composed by all the layers detected in the previously steps
- But the deliverable of the Detailed Design step will be composed of only one layer
- So, for each entity, we choose the appropriate layer of representation



Detailed Design Phase Meta-model



Detailed Design

Agent — is an autonomous entity able to play several roles

Society — is defined as the abstraction responsible for a collection of agents

Artifact — is an object able to provides several service.

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

Environment — is the environment of the system.

Workspace — is a conceptual locus in the environment

From Architectural Design to Detailed Design I

Mapping Tables in top-down order – $(L)MAR_t$, $(L)MArR_t$, $(L)MArI_t$

Agent	Role
agent name	*role names*

(Environmental) Artifact	Resource
artifact name	*resource names*

Interaction	(Social) Artifact
interaction name	*artifact names*

Mapping Tables

From Architectural Design to Detailed Design II

- *Mapping Agent-Role*($(L)MAR_t$) maps roles onto the agents
- *Mapping Artifact-Resource Table* ($(L)MArR_t$) maps resources onto the artifacts
- *Mapping Artifact-Interaction Table* ($(L)MArI_t$)maps the rules specified in architectural design onto the artifacts that improve them

Detailed Design: Tabular Representation I

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	(Social)Artifact
society name	*artifact names*

Agent/Society Design Tables

Detailed Design: Tabular Representation II

- *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 (social) artifacts related to societies.

Environment Design Tables in top-down order – $(L)AUI_t$, $(L)AggA_t$, $(L)WA_t$

Detailed Design: Tabular Representation III

Artifact	Usage Interface
artifact name	*list of operations*

Aggregate	Artifact
aggregate name	*artifact names*

Workspace	Artifact
workspace name	*artifact names*

Environment Design Tables

Detailed Design: Tabular Representation IV

- *Artifact-UsageInterface Table* ($(L)AUI_t$) specifies the operations provided by artifacts.
- *Aggregate-Artifact Table* ($(L)AggA_t$) specifies which artifact compose the composition.
- *Workspace-Artifact Table* ($(L)WA_t$) specifies the artifact located in the workspace

WebSite

- http://www.alice.unibo.it/soda/

Conclusions and Future Works

- SODA allows to
  - design societies
  - design environments
  - support the complexity of system description (layering principle)
- Future works
  - refining the meta-model
  - building the tools
  - modelling SODA according to SPEM (Software Process Engineering Meta-Model)
  - extracting fragments from SODA according to IEEE-FIPA Method Engineering

Bibliography I

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.

Molesini, A., Denti, E., and Omicini, A. (2007).
Agent-based conference management: A case study in SODA.
Submitted to the International Journal of Agent-Oriented Software Engineering (IJAOSE).

Bibliography II

Molesini, A., Omicini, A., Denti, E., and Ricci, A. (2006a).
SODA: A roadmap to artefacts.
In Dikenelli, O., Gleizes, M.-P., and Ricci, A., editors, *Engineering Societies in the Agents World VI*, volume 3963 of *LNAI*, pages 49–62. Springer.
6th International Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 October 2005. Revised, Selected & Invited Papers.

Molesini, A., Omicini, A., Ricci, A., and Denti, E. (2006b).
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.

Bibliography III

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.

Omicini, A., Ricci, A., and Viroli, M. (2006).
Coordination artifacts as first-class abstractions for MAS engineering: State of the research.
In Garcia, A. F., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., and Romanovsky, A., editors, *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications*, volume 3914 of *LNAI*, pages 71–90. Springer.
Invited Paper.

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 2007/2008