



SODA

Multiagent Systems LS Sistemi Multiagente LS

Ambra Molesini

`ambra.molesini@unibo.it`

ALMA MATER STUDIORUM—Università di Bologna

Academic Year 2006/2007





1 SODA

- Agents and Artifacts Meta-model
- Principles and Mechanisms
- The SODA Methodology
- Case Study: a sketch





Introduction

- 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 meta-model**) as building blocks for MAS development
- SODA introduces a simple layering principle in order to manage the complexity of the system description
- SODA adopts a tabular representations



Outline

- 1 SODA
 - Agents and Artifacts Meta-model
 - Principles and Mechanisms
 - The SODA Methodology
 - Case Study: a sketch



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
- 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



Features

- An artifact exposes
 - **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
- Other interesting artifact features 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 should be modifiable at execution time in order to adapt to the changing needs or mutable external conditions of a MAS.
 - *Linkability*: artifacts can be used encapsulate and model reusable services in a MAS. To scale up with complexity of an environment, it might be useful to compose artifacts, by allowing artifacts to invoke operations on other artifacts.



Classification

- A possible classification
 - **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 MASs with a service which is in the first place meant to achieve a social goal of the MAS, rather than an individual agent goal.
 - **Resource artifacts**: which mediate between a MAS and an external resource. In principle, resource artifacts can be conceived as a means to raise external MAS resources up to the agent cognitive level.



Agents & Artifacts

- Artifacts constitute the basic building block 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 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



Outline

- 1 SODA
 - Agents and Artifacts Meta-model
 - Principles and Mechanisms
 - The SODA Methodology
 - Case Study: a sketch



Complex systems and layering

- As advocate in the *Theory of Hierarchies* all biological systems are amenable to be represented as organised on different layers ranging from genes and cells up to organisms, species and clades.
- 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
- When generally ascribed to complex system, this sort of “hierarchy principle” might also be seen as a defining one: that is, a complex system is a system requiring layers—independent but strongly correlated ones—in order to fully understand and reproduce its dynamics and behaviour.



Layering and MASs

- When applied to the engineering of MASs, the hierarchy 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
- Allowing engineers to design and develop MAS along different levels of abstractions
 - a number of independent, but strictly related, MAS layers

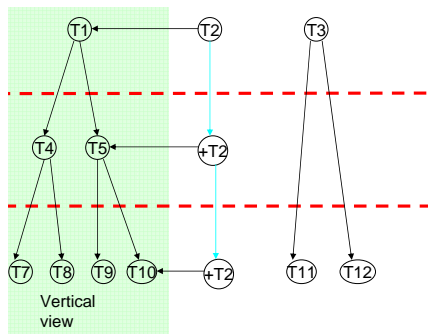
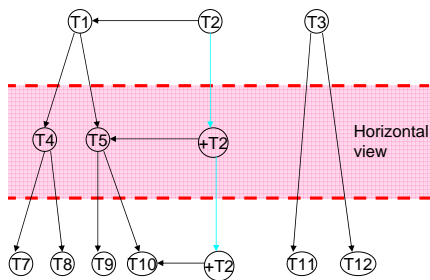


Layering in SODA

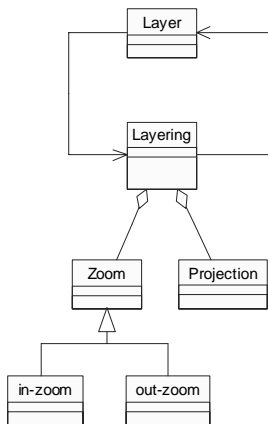
- We achieve the layering principle by means of the zooming and projection mechanisms.
- In the zooming mechanism we have two kinds of zoom
 - *in-zoom*: when passing from abstract layer to another more detailed
 - *out-zoom*: when passing from detailed layer to another more abstract
- The *projection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



System's views



Meta-Model of Layering

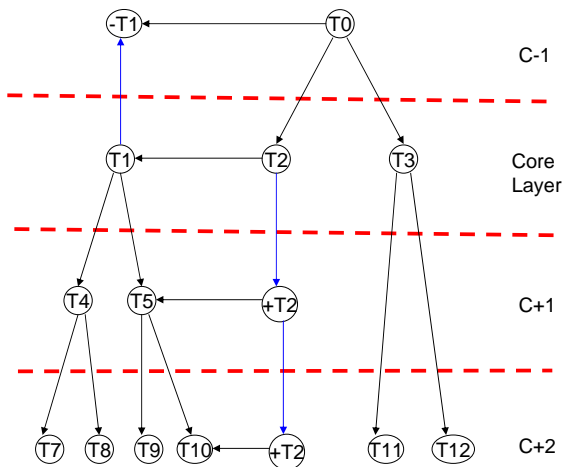


Layering in SODA

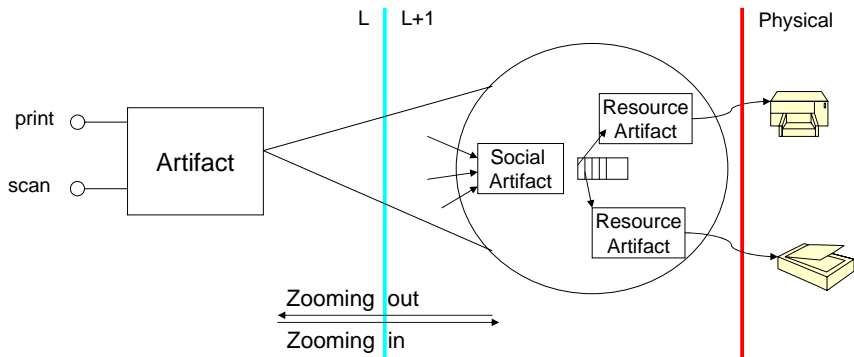
- In general, when working with SODA, we start from a certain layer, we could call *core layer*, and it is labelled with “c”
- The core layer is always **complete**
- In the other layer we find only the in/out zoomed entities and the projection entities.
- The in-zoomed layers are labelled with “c+1”, “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* expressed by means of zooming table (in the following)
- If we have relation between entities belonging to different layers we have to project these entities in the same layer



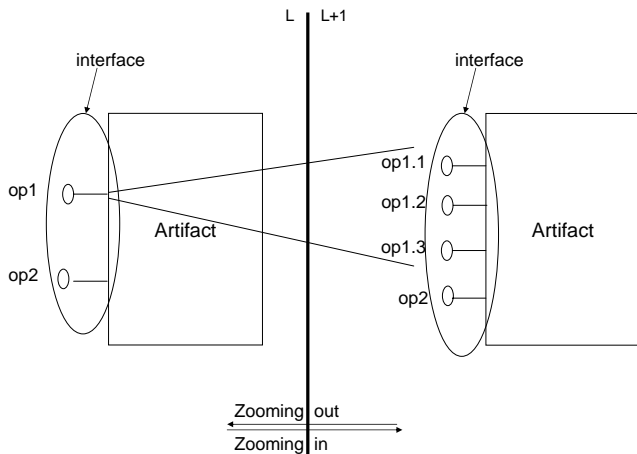
Example



Zooming Artifact 1/2



Zooming Artifact 2/2



Outline

- 1 SODA
 - Agents and Artifacts Meta-model
 - Principles and Mechanisms
 - The SODA Methodology
 - Case Study: a sketch

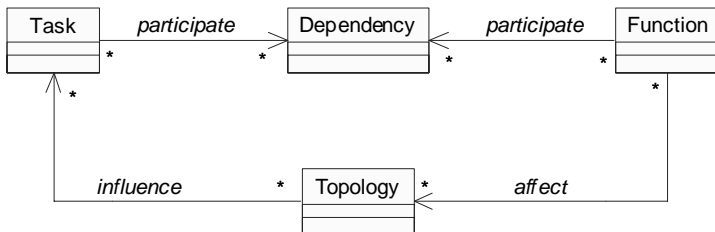


SODA

- SODA is organised in three phase:
 - *Analysis phase*: the system's requirements are analysed and modelled in terms of tasks, functions, topologies and dependencies
 - *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



Analysis Phase Meta-model



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.
- *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.



Analysis Phase: Tabular Representation

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

Task	Description
<i>task name</i>	<i>task description</i>

Function	Description
<i>function name</i>	<i>function description</i>

- Dependencies Table: $(L)D_t$

Dependency	Description
<i>dependency name</i>	<i>dependency description</i>

- Topology Table: $(L)Top_t$

Topology	Description
<i>Topology name</i>	<i>topology description</i>



Analysis Phase: 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 Table relates the abstract entities among them.
 - *Dependency Table* $((L)D_t)$ lists all the dependency among abstract entities and provides a description to them.
- Topology Table expresses the topological needs.
 - *Topology Table* $((L)Top_t)$ lists all the topological requirements and provides a description to them.



Analysis Phase: Tabular Representation

- Link Tables, in top – down order: $(L)TD_t$, $(L)FD_t$, $(L)TTop_t$, $(L)FTop_t$

Task	Dependency
<i>task name</i>	<i>dependency names</i>
Function	Dependency
<i>function name</i>	<i>dependency names</i>
Task	Topology
<i>task name</i>	<i>topology names</i>
Function	Topology
<i>function name</i>	<i>topology names</i>



Analysis Phase: Tabular Representation

- Link Tables relate the abstract entities, dependencies and topological requirements.
 - *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.
 - *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.



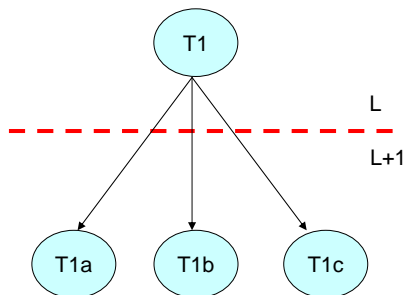
Zooming: Tabular Representation

- Zooming Table: $(L)Z_t$

Layer L	Layer L+1
<i>out-zoomed entity</i>	<i>in-zoomed entities</i>



Example: In-zoom task

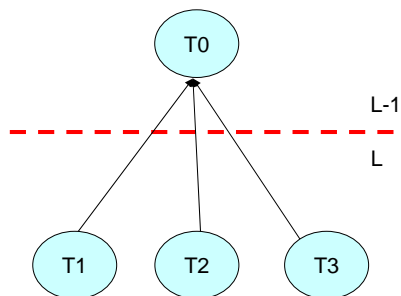


- Zooming Table: $(L)Z_t$

Layer L	Layer L+1
$T1$	$T1a, T1b, T1c, \dots$



Example: Out-zoom tasks



- Zooming Table: $(L)Z_t$

Layer L-1	Layer L
T_0	T_1, T_2, T_3, \dots

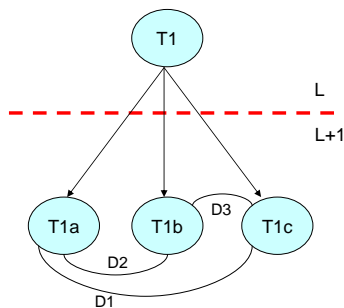


Important:

- 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

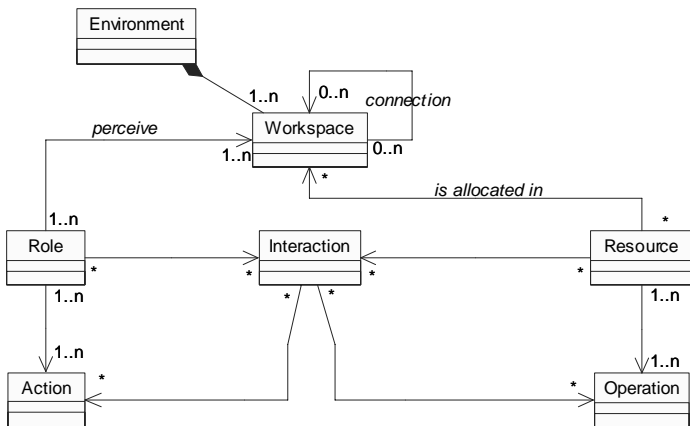


- Zooming Table: $(L)Z_t$

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



Architectural Design Phase Meta-model



Architectural Design Phase

- *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. The 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 Phase to Architectural Design Phase

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

Role	Task
<i>role name</i>	<i>task names</i>
Resource	Function
<i>resource name</i>	<i>function names</i>
Dependency	Interaction
<i>dependency name</i>	<i>interaction names</i>
Topology	Workspace
<i>topology name</i>	<i>workspace names</i>



From Analysis Phase to Architectural Design Phase

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 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 Phase: Tabular Representation

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

Action	Description
<i>action name</i>	<i>description</i>
Operation	Description
<i>operation name</i>	<i>description</i>
Role	Action
<i>role name</i>	<i>action names</i>
Resource	Operation
<i>resource name</i>	<i>operation names</i>



Architectural Design Phase: 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 Phase: Tabular Representation

- Interactions Tables in top – down order: $(L)I_t$, $(L)Rol_t$, $(L)Rel_t$

Interaction	Description
<i>interaction name</i>	<i>description</i>
Role	Interaction
<i>role name</i>	<i>interaction names</i>
Resource	Interaction
<i>resource name</i>	<i>interaction names</i>



Architectural Design Phase: 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.
- *Role-Interaction Table* ($(L)Rol_t$) specifies the list of interactions where roles are involved
- *Resource-Interaction Table* ($(L)Rel_t$) specifies the list of interactions where resources are involved



Architectural Design Phase: Tabular Representation

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

Workspace	Description
<i>workspace name</i>	<i>description</i>
Workspace	Connection
<i>workspace name</i>	<i>workspace names</i>
Workspace	Resource
<i>workspace name</i>	<i>resource names</i>
Role	Workspace
<i>role name</i>	<i>workspace names</i>



Architectural Design Phase: Tabular Representation

Topological Tables:

- *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.

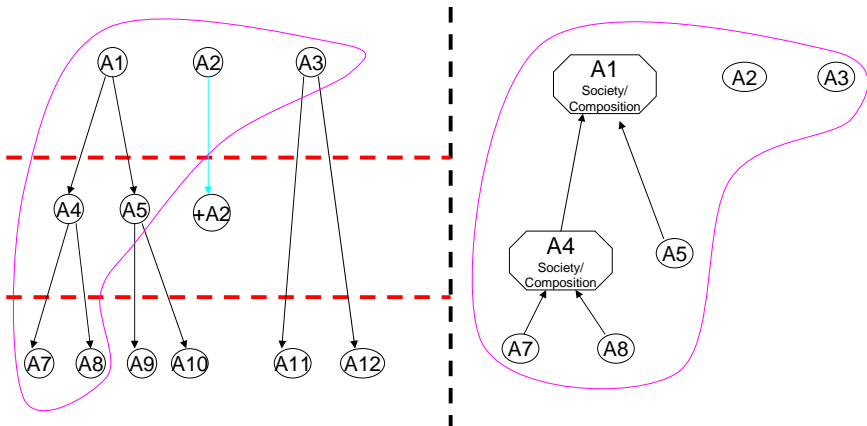


Detailed Design Phase

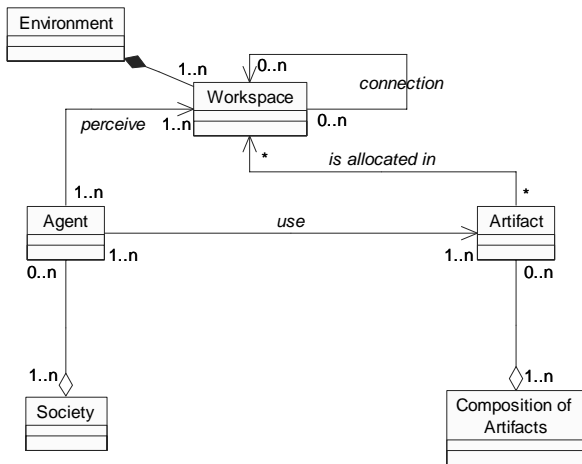
- In this phase we do not adopt the zooming, potentially our system could be composed by all the layers detected in previously phases, we can speak about several *design views*
- This is unacceptable in the design phase because we should give to developer a clear representation of the system, then the deliverable of this phase will be composed of only one layer.
- This method of work leads us to have several design views of the system
- This is important to allow several degrees of systems' prototyping and to facilitate the organisation of work in team.



Design views



Detailed Design Phase Meta-model



Detailed Design Phase

- *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.
- *Composition of Artifacts*: 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 Phase to Detailed Design Phase

- Mapping Tables in top- down order: $(L)MAR_t$, $(L)MARR_t$, $(L)MARl_t$

Agent	Role
<i>agent name</i>	<i>role names</i>
(Resource) Artifact	Resource
<i>artifact name</i>	<i>resource names</i>
Interaction	(Social) Artifact
<i>interaction name</i>	<i>artifact names</i>



From Architectural Design Phase to Detailed Design Phase

Mapping Tables:

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



Detailed Design Phase: Tabular Representation

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

Agent	(Individual) Artifact
<i>agent name</i>	<i>artifact names</i>
Society	Agent
<i>society name</i>	<i>agent names</i>
Society	(Social)Artifact
<i>society name</i>	<i>artifact names</i>



Detailed Design Phase: 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 (social) artifacts related to societies.



Detailed Design Phase: Tabular Representation

- Environment Design Tables in top- down order: $(L)AUI_t$, $(L)CA_t$, $(L)WA_t$

Artifact	Usage Interface
<i>artifact name</i>	<i>list of operations</i>

Composition of Artifacts	Artifact
<i>composition of Artifacts name</i>	<i>artifact names</i>

Workspace	Artifact
<i>workspace name</i>	<i>artifact names</i>



Detailed Design Phase: Tabular Representation

Environment Design Tables:

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



SODA: Pros and Cons

- Pros:
 - Design societies
 - Design environments
 - Layering principle
- Cons:
 - SODA does not explicitly deal with the activities of the requirements capturing
 - Only inter-agents aspects: need of other methodology for design intra-agents aspects



Outline

- 1 SODA
 - Agents and Artifacts Meta-model
 - Principles and Mechanisms
 - The SODA Methodology
 - Case Study: a sketch

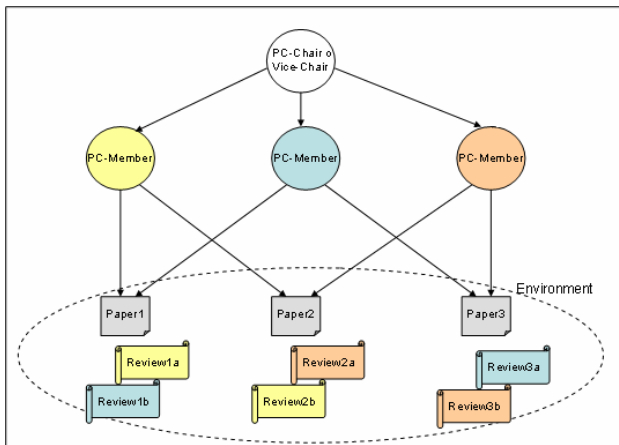


A case study: distributed paper review

- The “*Program Chair*” (PC-Chair) publishes a call for papers
- Authors submit a papers
- A number of scientists (called “PC-Members”) review the papers and give marks
- To ensure fairness, the reviewers must be anonymous, expert, and must be willing to do the review
- Also, each paper should receive a minimum number of review from different scientists
- All accepted papers will be published on a book



Case Study



A case study: why Agents?

- It is a typical case of distributed workflow management
 - There are actions to do on common documents
 - According to specific rules
- Each of the human actors involved in the process
 - Could be supported by a personal agents
 - Helping him to submit documents, filling in, respect deadlines, etc.



Analysis Phase

- $(C)T_t$

Task	Description
start up	insertion of the setup information
submission	submission of paper
paper partitioning	partitioning of the set of papers
assignment papers	assignment papers to PC-members
review process	creation and submission of the reviews



Analysis Phase

- $(C)F_t$

Function	Description
management user	managing users' information
management review	managing reviews' information
management paper	managing papers' information
management assignment	managing assignments' information
management partitioning	managing partitioning's information
management process	managing start-up's information



Analysis Phase

- $(C)D_t$

Dependency	Description
start up information	access to all the information about start up process
user information	access to all the users' information
paper information	access to all the papers' information
partitioning information	access to all the information about partitioning process
submission information	access to all the information about submission process
assignment information	access to all the information about assignment process. A reviewer cannot be the author of the papers that are assigned to him
review information	access to all the information about review process



Analysis Phase

- $(C)TD_t$

Task	Dependency
start up	start up information
submission	submission information user information paper information
paper partitioning	partitioning information paper information
assignment papers	assignment information paper information user information
review process	review information paper information



Analysis Phase

- $(C)FD_t$

Function	Dependency
management user	user information submission information assignment information
management review	review information
management paper	paper information submission information assignment information review information partitioning information
management assignment	assignment information
management partitioning	partitioning information
management process	start up information



Analysis Phase

- $(C)Top_t$

Topology	Description

- $(C)TTop_t$

Task	Topology

- $(C)TTop_t$

Function	Topology



From Analysis Phase to Architectural Design Phase

- $(C)TRT_t$

Role	Task
PC-Chair	paper partitioning start-up assignment papers
Author	submission
PC-member	review process

- $(C)TRF_t, (L)TID_t, (L)TTopW_t$

Resource	Function
People DB	management user
Paper DB	management paper management review management partitioning management assignment
Process DB	management process



From Analysis Phase to Architectural Design Phase

- $(C)TID_t$

Dependency	Interaction
start up information	
user information	User-Rule
paper information	Author-Rule
partitioning information	Match-Rule
submission information	Deadline-Rule
assignment information	AutRev-Rule Review-Rule
review information	Author-Rule

- $(C)TTopW_t$

Topology	Workspace



Architectural Design Phase

- $(C)A_t$

Action	Description
login	user authentication
send paper	user compiles form and sends his paper
publish deadline	user generates/modifies deadline
partition	user splits papers according to key words
assignment	user assigns papers
read paper	user reads papers
write review	user writes the review
...	...



Architectural Design Phase

- $(C)RA_t$

Role	Action
PC-Chair	login publish deadline partition assignment ...
Author	login send paper ...
PC-member	login read paper write review ...



Architectural Design Phase

- $(C)O_t$

Operation	Description
store paper	storing paper and its information
get paper	providing paper and its information
store user	storing user information
get user	providing user information
store process	storing process information
get process	providing process information
store assignment	storing assignment information
...	...



Architectural Design Phase

- $(C)RO_t$

Resource	Operation
People DB	store user get user ...
Paper DB	store paper get paper store assignment ...
Process DB	get process store process ...



Architectural Design Phase

- $(C)I_t$

Interaction	Description
Deadline-Rule	send paper is possible if and only if time j deadline submission
User-Rule	get user is possible if the request user is the requester or the requester is the PC-Chair
Author-Rule	author can access and modify only his public paper information
Match-Rule	papers can be partitioned according key words
AutRev-Rule	the PC-member cannot be the author of paper
Review-Rule	the PC-member cannot access to private information about his papers
...	...



Architectural Design Phase

- $(C)Rol_t$

Role	Interaction
PC-Chair	Deadline-Rule User-Rule Author-Rule Match-Rule AutRev-Rule Review-Rule ...
Author	Deadline-Rule User-Rule Author-Rule ...
PC-member	User-Rule Author-Rule AutRev-Rule Review-Rule ...



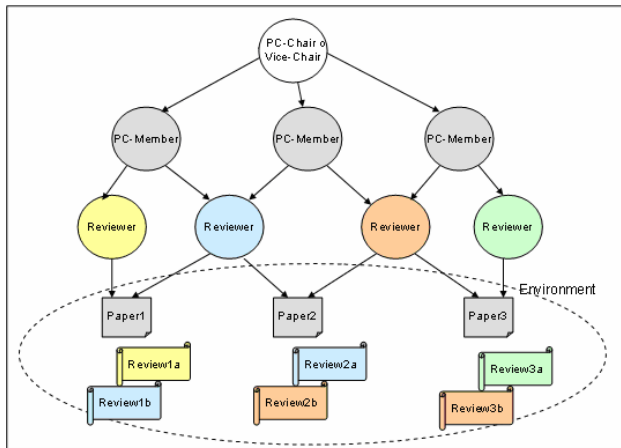
Architectural Design Phase

- $(C)Rel_t$

Resource	Interaction
People DB	User-Rule Match-Rule ...
Paper DB	Author-Rule AutRev-Rule Review-Rule ...
Process DB	Deadline-Rule ...



What changes if....



Web



<http://www.alice.unibo.it:8080/SODA/>



Bibliography

-  **Ambra Molesini, Enrico Denti, and Andrea Omicini.**
MAS meta-models on test: UML vs. OPM in the SODA case study.
 In Michal Pěchouček, Paolo Petta, and László Zsolt Varga, editors, *Multi-Agent Systems and Applications IV*, volume 3690 of *LNAI*, pages 163–172. Springer, 2005.
 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05), Budapest, Hungary, 15–17 September 2005, Proceedings.
-  **Ambra Molesini, Andrea Omicini, Enrico Denti, and Alessandro Ricci.**
SODA: A roadmap to artefacts.
 In Öğuz Dikenelli, Marie-Pierre Gleizes, and Alessandro Ricci, editors, *Engineering Societies in the Agents World VI*, volume 3963 of *LNAI*, pages 49–62. Springer, June 2006.
 6th International Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 October 2005. Revised, Selected & Invited Papers.
-  **Ambra Molesini, Andrea Omicini, Alessandro Ricci, and Enrico Denti.**
Zooming multi-agent systems.
 In Jörg P. Müller and Franco Zambonelli, editors, *Agent-Oriented Software Engineering VI*, volume 3950 of *LNCS*, pages 81–93. Springer, 2006.
 6th International Workshop (AOSE 2005), Utrecht, The Netherlands, 25–26 July 2005. Revised and Invited Papers.
-  **Andrea Omicini.**
SODA: Societies and infrastructures in the analysis and design of agent-based systems.
 In Paolo Ciancarini and Michael J. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 185–193. Springer-Verlag, 2001.
 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.



SODA

Multiagent Systems LS Sistemi Multiagente LS

Ambra Molesini

`ambra.molesini@unibo.it`

ALMA MATER STUDIORUM—Università di Bologna

Academic Year 2006/2007

