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
1 Overview

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

3 The SODA Methodology
   - Analysis Phase
   - Design Phase
   - Web Resources and Conclusions
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: Meta-model
Outline

1. Overview

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

3. The SODA Methodology
   - Analysis Phase
   - Design Phase
   - Web Resources and Conclusions
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
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.
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
 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 environment
  - 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
Outline

1. Overview

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

3. The SODA Methodology
   - Analysis Phase
   - Design Phase
   - Web Resources and Conclusions
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

- Artifact
- Print
- Scan

L - L+1

- Environmental Artifact
- Social Artifact
- Environmental Artifact

Zooming in
Zooming out
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
The SODA Methodology

Analysis Phase

Outline

1. Overview

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

3. The SODA Methodology
   - Analysis Phase
   - Design Phase
   - Web Resources and Conclusions
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

- **Actor**
- **Requirement**
- **Relation**
- **External-Environment**
- **Legacy-System**

The SODA Methodology

Analysis Phase

Omicini & Molesini (Università di Bologna)

SODA

A.Y. 2007/2008
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.
The SODA Methodology

Analysis Phase

**Requirement Analysis: Tabular Representation**

- **Requirements Tables**: $\text{(L)AR}_t$ and $\text{(L)Re}_t$
  
<table>
<thead>
<tr>
<th>Actor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>actor name</td>
<td>requirement names</td>
</tr>
</tbody>
</table>
  
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirement name</td>
<td>requirement description</td>
</tr>
</tbody>
</table>

- **Domain Tables**: $\text{(L)EELS}_t$ and $\text{(L)LS}_t$
  
<table>
<thead>
<tr>
<th>External-Environment</th>
<th>Legacy-System</th>
</tr>
</thead>
<tbody>
<tr>
<td>external-environment name</td>
<td>Legacy-System names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Legacy-System</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>legacy-system name</td>
<td>legacy-system description</td>
</tr>
</tbody>
</table>
Requirement Analysis: Tabular Representation

- Requirements Tables define and describe the abstract entities tied to the concept of “requirement”
  - *Actor-Requirement Table* ($\text{(L)AR}_t$) specifies the list of the requirements for each actor
  - *Requirement Table* ($\text{(L)Re}_t$) lists all the requirements and describes them.

- Domain Tables define and describe the abstract entities tied to the external environment
  - *ExternalEnvironment-LegacySystem Table* ($\text{(L)EELS}_t$) specifies the list of the contexts for external-environment
  - *Legacy-System Table* ($\text{(L)LS}_t$) lists all the contexts and describes them.
Requirement Analysis: Tabular Representation

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

<table>
<thead>
<tr>
<th>Relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation name</td>
<td>relation description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirement name</td>
<td>relation names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Legacy-System</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>legacy-system name</td>
<td>relation names</td>
</tr>
</tbody>
</table>
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
Analysis Meta-model

Task \textit{participate} Dependency \textit{participate} Function

\textit{influence} \textit{affect}
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 \((L)RRD_t\)

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirement name</td>
<td>task names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirement name</td>
<td>function names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Legacy-System</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>legacy-system name</td>
<td>function names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Legacy-System</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>legacy-system name</td>
<td>topology names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relation</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation name</td>
<td>dependency names</td>
</tr>
</tbody>
</table>
References Tables identify the relations among the abstractions of the requirement analysis phase and the abstractions used in analysis phase.

- **Reference Requirement-Task Table** $((L)\text{RRT}_t)$ specifies the mapping between requirement and tasks.
- **Reference Requirement-Function Table** $((L)\text{RRF}_t)$ specifies the mapping between requirement and resources.
- **Reference LegacySystem-Function Table** $((L)\text{RLSF}_t)$ specifies the mapping between legacy-system and functions.
- **Reference LegacySystem-Topology Table** $((L)\text{RLST}_t)$ specifies the mapping between legacy-system and topologies.
- **Reference Relation-Dependency Table** $((L)\text{RRD}_t)$ specifies the mapping between relations and dependencies.
Analysis: Tabular Representation

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task name</td>
<td>task description</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>function name</td>
<td>function description</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependency name</td>
<td>dependency description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>task name</td>
<td>dependency names</td>
</tr>
<tr>
<td>Function</td>
<td>Dependency</td>
</tr>
<tr>
<td>function name</td>
<td>dependency names</td>
</tr>
</tbody>
</table>
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
### Topologies Tables in top-down order – $(L)Top_t$, $(L)TTop_t$, $(L)FTop_t$

<table>
<thead>
<tr>
<th>Topology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology name</td>
<td>topology description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>task name</td>
<td>topology names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>function name</td>
<td>topology names</td>
</tr>
</tbody>
</table>
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 Table: \((L)Z_t\)

<table>
<thead>
<tr>
<th>Layer L</th>
<th>Layer L+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{out-zoomed entity}</td>
<td>\textit{in-zoomed entities}</td>
</tr>
</tbody>
</table>
Example: In-zoom task

Zooming Table: $(L)Z_t$

<table>
<thead>
<tr>
<th>Layer $L$</th>
<th>Layer $L+1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T1$</td>
<td>$T1a, T1b, T1c,\ldots$</td>
</tr>
</tbody>
</table>
Example: Out-zoom tasks

Zooming Table: $(L)Z_t$

<table>
<thead>
<tr>
<th>Layer L-1</th>
<th>Layer L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T0$</td>
<td>$T1, T2, T3, \ldots$</td>
</tr>
</tbody>
</table>
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: \((L)Z_t\)

<table>
<thead>
<tr>
<th>Layer L</th>
<th>Layer L+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T1)</td>
<td>(T1a, T1b, T1c)</td>
</tr>
<tr>
<td></td>
<td>(D1, D2, D3)</td>
</tr>
</tbody>
</table>
Outline

1. Overview

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

3. The SODA Methodology
   - Analysis Phase
   - Design Phase
   - Web Resources and Conclusions
Architectural Design Meta-model

- **Environment**
  - 1..n
  - 1..n
  - 1..n
  - 1..n

- **Workspace**
  - 0..n
  - connection
  - is allocated in

- **Role**
  - 1..n
  - 1..n
  - 1..n

- **Action**
  - 1..n
  - 1..n
  - 1..n

- **Interaction**
  - 1..n
  - 1..n
  - 1..n
  - 1..n

- **Resource**
  - 1..n

- **Operation**
  - 1..n

- **Omicini & Molesini (Università di Bologna)**

- **SODA**
  - 44 / 70
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 – \((L) TRT_t, (L) TRF_t, (L) TID_t, (L) TTTopW_t\)

<table>
<thead>
<tr>
<th>Role</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>role name</em></td>
<td><em>task names</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>resource name</em></td>
<td><em>function names</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>dependency name</em></td>
<td><em>interaction names</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Topology</th>
<th>Workspace</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>topology name</em></td>
<td><em>workspace names</em></td>
</tr>
</tbody>
</table>
From Analysis to Architectural Design II

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: Tabular Representation I

<table>
<thead>
<tr>
<th>Entities Tables in top-down order – ((L)A_t, (L)O_t, (L)RA_t, (L)RO_t)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action</strong></td>
</tr>
<tr>
<td><code>action name</code></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
</tr>
<tr>
<td><code>operation name</code></td>
</tr>
<tr>
<td><strong>Role</strong></td>
</tr>
<tr>
<td><code>role name</code></td>
</tr>
<tr>
<td><strong>Resource</strong></td>
</tr>
<tr>
<td><code>resource name</code></td>
</tr>
</tbody>
</table>
The SODA Methodology

Design Phase

Architectural Design: Tabular Representation II

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 III

### Interactions Tables in top-down order – \((L)I_t, (L)RoI_t, (L)Rel_t\)

<table>
<thead>
<tr>
<th>Interaction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interaction name</td>
<td>description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Role</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>role name</td>
<td>interaction names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource name</td>
<td>interaction names</td>
</tr>
</tbody>
</table>
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** ($(L)RoI_t$) 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: Tabular Representation V

#### Topological Tables in top-down order – (L)\(W_t\), (L)\(WC_t\), (L)\(WRe_t\) and (L)\(WRot\)

<table>
<thead>
<tr>
<th>Workspace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace name</td>
<td>description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workspace</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace name</td>
<td>workspace names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workspace</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace name</td>
<td>resource names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Role</th>
<th>Workspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>role name</td>
<td>workspace names</td>
</tr>
</tbody>
</table>
The SODA Methodology

Design Phase

Architectural Design: Tabular Representation VI

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.
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

The SODA Methodology

Workspace

0..n
connection

Artifact

is allocated in

Agent

perceive
1..n

Environment

0..n

 perceive
1..n

Society

use
1..n

0..n

Artifacts

1..n

Aggregate

1..n

0..n

Omicini & Molesini (Università di Bologna)
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 provide several services.

**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
## Mapping Tables in top-down order – \((L)MAR_t, (L)MArR_t, (L)MArI_t\)

<table>
<thead>
<tr>
<th>Agent</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{agent name}</td>
<td>\textit{role names}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(Environmental) Artifact</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{artifact name}</td>
<td>\textit{resource names}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interaction</th>
<th>(Social) Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{interaction name}</td>
<td>\textit{artifact names}</td>
</tr>
</tbody>
</table>
Mapping Tables

- **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** \(^{(L)MArl_t}\) maps the rules specified in architectural design onto the artifacts that improve them
### Agent/Society Design Tables in top-down order – \((L)AA_t, (L)SA_t, (L)SAr_t\)

<table>
<thead>
<tr>
<th></th>
<th>(Individual) Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Agent</strong></td>
<td></td>
</tr>
<tr>
<td><em>agent name</em></td>
<td><em>artifact names</em></td>
</tr>
<tr>
<td><strong>Society</strong></td>
<td></td>
</tr>
<tr>
<td><em>society name</em></td>
<td><em>agent names</em></td>
</tr>
<tr>
<td><strong>Society</strong></td>
<td>(Social)Artifact</td>
</tr>
<tr>
<td><em>society name</em></td>
<td><em>artifact names</em></td>
</tr>
</tbody>
</table>
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: Tabular Representation III

#### Environment Design Tables in top-down order – (L)AUI<sub>t</sub>, (L)AggA<sub>t</sub>, (L)WA<sub>t</sub>

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Usage Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifact name</td>
<td>list of operations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Aggregate</th>
<th>Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate name</td>
<td>artifact names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workspace</th>
<th>Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace name</td>
<td>artifact names</td>
</tr>
</tbody>
</table>
Environment Design Tables

- **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.
Outline

1. Overview

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

3. The SODA Methodology
   - Analysis Phase
   - Design Phase
   - Web Resources and Conclusions
Web Resources and Conclusions

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 II


SODA: Societies and infrastructures in the analysis and design of agent-based systems.  
1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.

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*
Bibliography IV

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