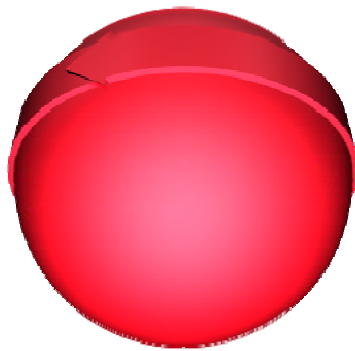




ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

SODA

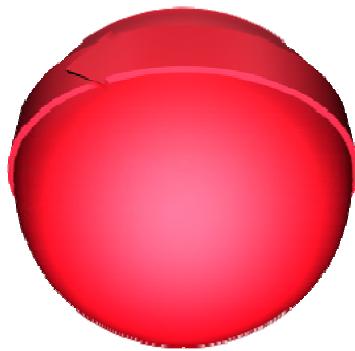


Societies in Open Distributed Agent spaces



ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

SODA



Societies in Open Distributed Agent spaces

Date:	24th June 2010
Version:	First Draft
Pages:	
Responsible:	Ambra Molesini
Authors:	Ambra Molesini Andrea Omicini
Contact:	ambra.molesini@unibo.it

Contents

1	Introduction	2
1.1	The SODA Process Lifecycle	3
1.2	The Meta-model	4
1.3	The Layering	7
1.3.1	The Layering work products	7
	Bibliography	12

1

Introduction

SODA (Societies in Open and Distributed Agent spaces) [1, 2, 3, 5] is an agent-oriented methodology for the analysis and design of agent-based systems, which adopts the Agents & Artifacts (A&A) meta-model [6], and introduces a *layering principle* as an effective tool for scaling with the system complexity, applied throughout the analysis and design process. Since its first version [5], **SODA** is not concerned with *intra-agent* issues: designing a multi-agent system with **SODA** leads to defining agents in terms of their required observable behaviour and their role in the multi-agent system. Then, whichever methodology one may choose to define the agent structure and inner functionality, it could be easily used in conjunction with **SODA**. **SODA** concentrated on *inter-agent* issues, like the engineering of societies and environment for multi-agent systems.

SODA is organised in two phases, each structured in two sub-phases called steps: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps. In this way the requirements analysis and the architectural design are now part of the **SODA** process and they do not require the ad hoc approach as in the first **SODA** definition.

An overview of the methodology structure is shown in Figure 1.1: each step is practically described in terms of a set of relational tables, listed in the figure.

SODA provides only a tabular representation of the system. Each table in the methodology has both a full name and a unique acronym, which specifies the layer (see Section 1.3) where the table belongs (in round brackets) and the involved entities. For instance, $(L)AR_t$ refers to the Actor (specified by the “A”)-Requirement (specified by the “R”) table (specified by the “t”) at layer L.

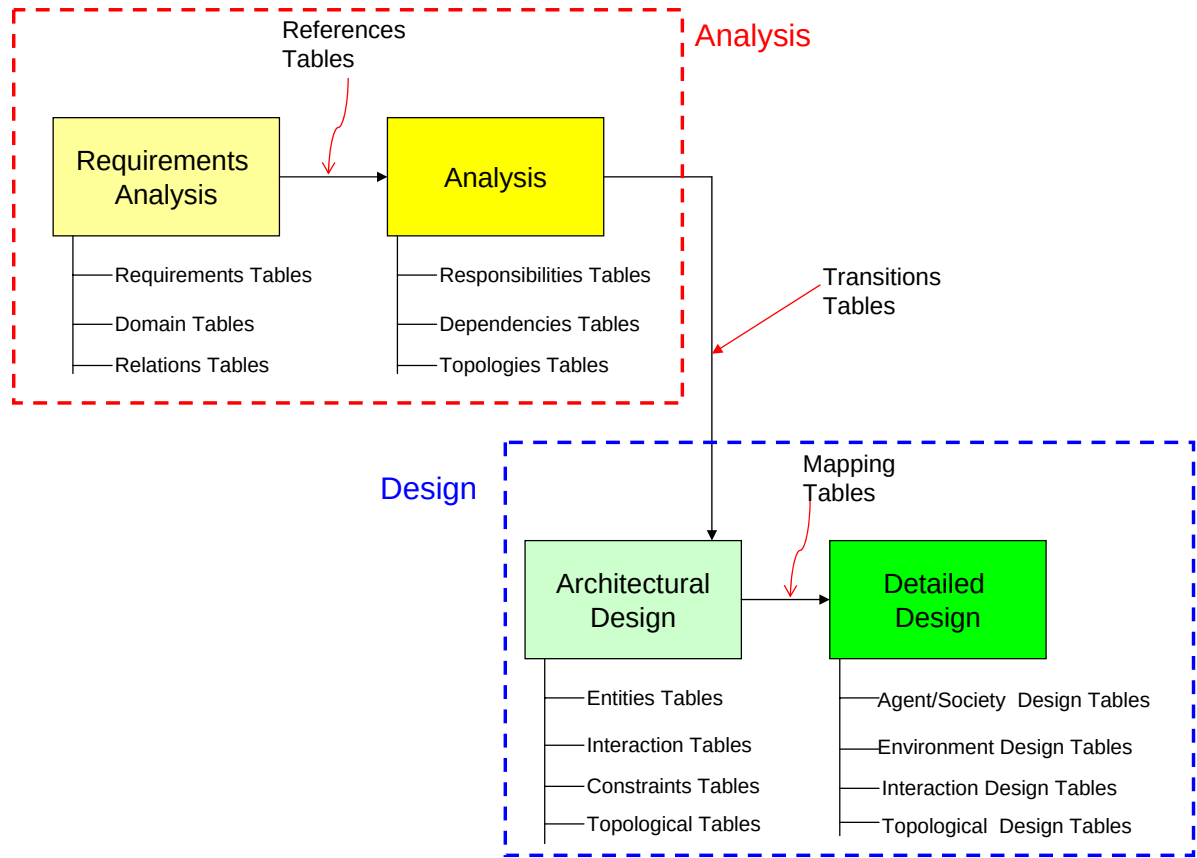


Figure 1.1: An overview of the SODA process

1.1 The SODA Process Lifecycle

SODA includes two phases each of them structure in two sub-phases: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps.

These phases are arranged in iterative process model (see Figure 1.2):

- Requirements Analysis covers all the phases related to the Actor identification, the Requirements Elicitation and Analysis and the Analysis of the existing environment.
- Analysis investigates all the aspects related to the problem domain trying to understand the tasks satisfying the requirements, their connected functions, the environment topology and all the dependencies among these entities.

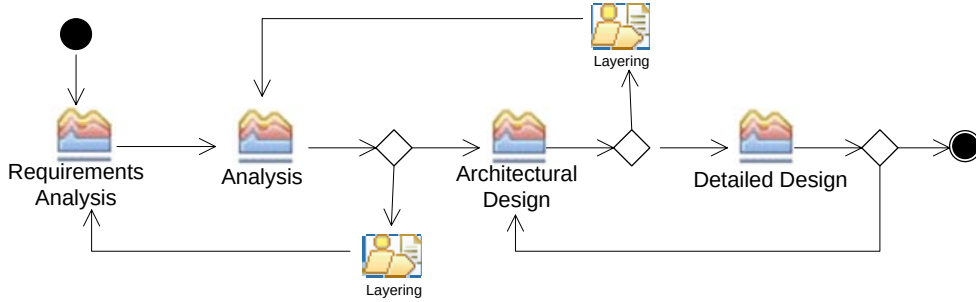


Figure 1.2: The **SODA** process phases

- Architectural Design defines a family of possible architectures for the final system.
- Detailed Design establish the best system architecture and design the environment and the system’s interactions.

Each step produces several sets of relational tables each of them describes a specific MAS Meta-model Element (MMMElement) and its relationships with the other MMMElements.

The details of each step will be discussed in the following section.

1.2 The Meta-model

The meta-model that represents the abstract entities adopted by **SODA** is depicted in Figure 1.3.

Requirements Analysis. Several abstract entities are introduced for requirement modelling (see Figure 1.3 “requirements analysis” part): in particular, *requirement* and *actor* are used for modelling the customers’ requirements and the requirement sources, respectively, while the *external-environment* notion is used as a container of the *legacy-systems* that represent the legacy resources of the environment. The relationships between requirements and legacy systems are then modelled in terms of suitable *relation* entities.

Analysis. The Analysis step expresses the abstract requirement representation in terms of more concrete entities such as *tasks* and *functions* (see Figure 1.3, “analysis” part). Tasks are activities requiring one or more competences, while functions are reactive activities aimed at supporting tasks. The relations highlighted in the previous step are now the starting point for

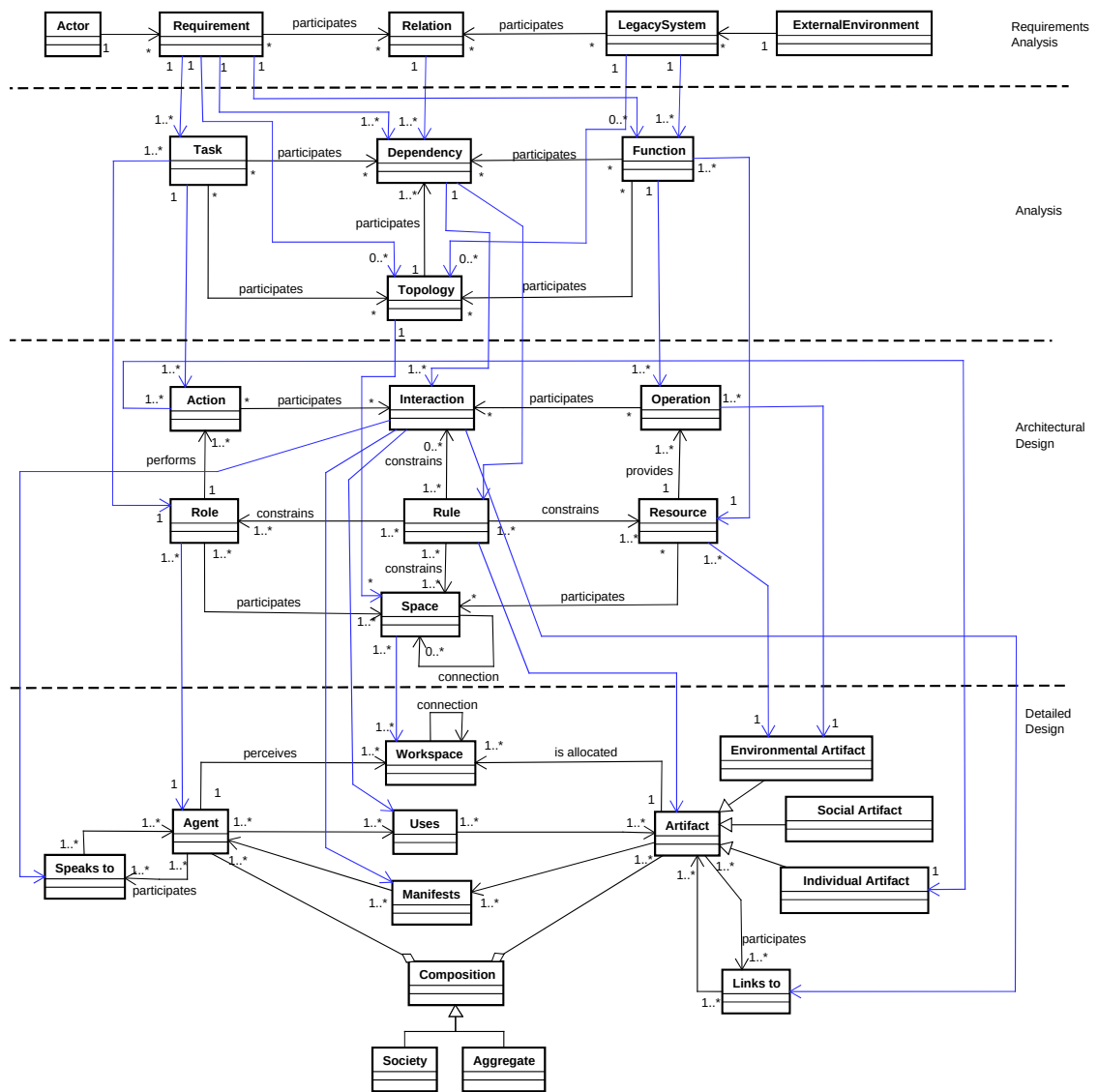


Figure 1.3: The SODA Meta-model

the definition of *dependencies* (interactions, constraints, etc.) among the abstract entities. The structure of the environment is also modelled in terms of *topologies*, i.e. topological constraints over the environment.

Topologies are often derived from functions, but can also constrain / affect task achievement.

Architectural Design. The main goal of this stage is to assign responsibilities of achieving tasks to *roles*, and responsibilities of providing functions to *resources* (see Figure 1.3, “architectural design” part). To this end, roles should be able to perform *actions*, and resources should be able to execute *operations* providing one or more functions. The dependencies identified in the previous phase become here *interactions*, i.e. “rules” enabling and bounding the entities’ behaviour. Finally, the topology constraints lead to the definition of *workspaces*, i.e. conceptual places structuring the environment.

Detailed Design. Detailed Design is expressed in terms of *agents*, agent *societies*, *artifacts* and *aggregates* (see Figure 1.3 “detailed design” part). Agents are intended here as autonomous entities able to play several roles, and the resources identified in the previous step are now mapped onto suitable artifacts. In the meta-model the artifact is reported specifying its “type” derived from the following taxonomy [7]:

- *Individual artifact* handles the interaction of a single agent within a MAS, and essentially works as a mediator between the agent and the MAS itself. Since they can be used to shape of admissible interactions of individual agents in MAS, individual artifacts play an essential role in engineering both organisational and security concerns in MAS.
- *Environmental artifact* brings an external resource within a MAS, by mediating agent actions and perceptions over resources. As such, environmental artifacts play an essential role in enabling, disciplining and governing the interaction between agents and MAS environment.
- *Social artifact* rules social interactions within a MAS—even though indirectly, since it technically mediates interactions between individual, environmental, and possibly other social artifacts. Social artifacts in **SODA** play the role of the coordination artifacts that embody the rules around which societies of agents can be built.

In **SODA** a society can be seen as a group of interacting agents and artifacts when its overall behaviour is essentially an autonomous, proactive one; it can be seen an aggregate when its overall behaviour is essentially a functional, reactive one.

The *workspaces* defined in the Architectural Design step take now the form of an open set of artifacts and agents—that is, artifacts can be dynamically added to or removed from workspaces, and agents can dynamically enter (join) or exit workspaces.

1.3 The Layering

Complexity is inherent in real-life systems. While *modelling* complex systems and understanding their behaviour and dynamics is the most relevant concern in many areas, such as economics, biology, or social sciences, in the software systems also the complexity of *construction* becomes an interesting challenge. An integral part of a system development methodology must therefore be a set of tools for controlling and managing this complexity. So, soda has introduced a tool in order to deal with the complexity of the system representation: the system is represented as composed by different layers of abstraction and it is possible to move from one layer to another layer by means of a *layering operation*.

This tool in **SODA** is called “Layering” and it is represented as a *capability pattern* [4], i.e., a reusable portion of the process, as shown in Figure 1.4 where the layering process is detailed. In particular, the layering presents two different functionalities: *(i)* the selection of a specific layer for refining / completing the abstractions models in the methodology process, and *(ii)* the creation of a new layer in the system by in-zooming (i.e., increasing the system detail) or out-zooming (i.e., increasing the system abstraction) activities. In latter case, the layering process terminates with the projection activity needed to project the abstractions from one layer to another “as they are”, so as to maintain the consistency in each layer.

In general, when working with **SODA**, the starting layer, called *core layer*, is labelled with “C” and is always *complete*—that is, it contains all the entities required to fully describe a given abstract layer. Any other layer contains just *i)* the entities that have been possibly (in/out-) zoomed from another layer, as well as *ii)* the entities possibly projected “as they are” from other layers: so, in general, these layers are not necessarily complete—though of course they might be so.

1.3.1 The Layering work products

The Layering generates one work product (the Zooming table). Its relationships with the MMMElements are described in Figure 1.6

This diagram represents the Layering in terms of the Work Product and

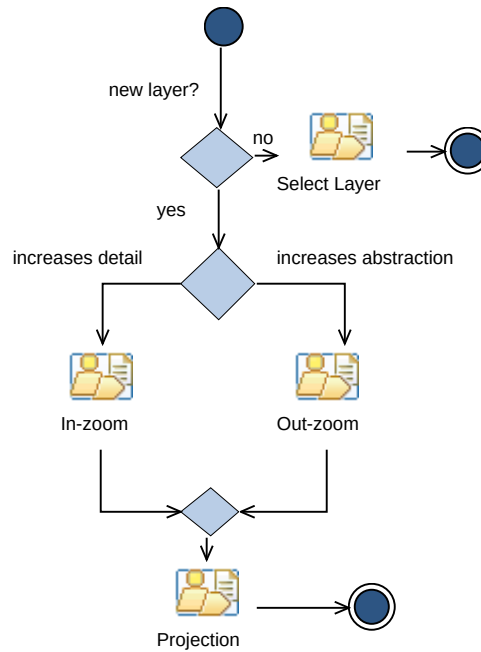


Figure 1.4: The Layering process

its relationships with the **SODA** meta-model (Section 1.2) elements. Each MMMElement is represented using an UML class icon (yellow filled) and, in the documents, such elements can be Defined, reFined, Quoted, Related or Relationship Quoted as described below:

- *defined* (D label): this means that the element is introduced for the first time in the design in this artefact (the MMMElement is instantiated in this artefact);
- *reFined* (F label): this means that the MMMElement is refined in the work product (for instance by means of attribute definition);
- *related* (R label): this means that an already defined element is related to another or from a different point of view that one of the MAS meta-model relationships is instantiated in the document;
- *quoted* (Q label): this means that the element has been already defined and it is reported in this artefact only to complete its structure but no work has to be done on it;
- *relationship quoted* (RQ label): this means that the relationship is reported in the work product but it has been defined in another part of the process

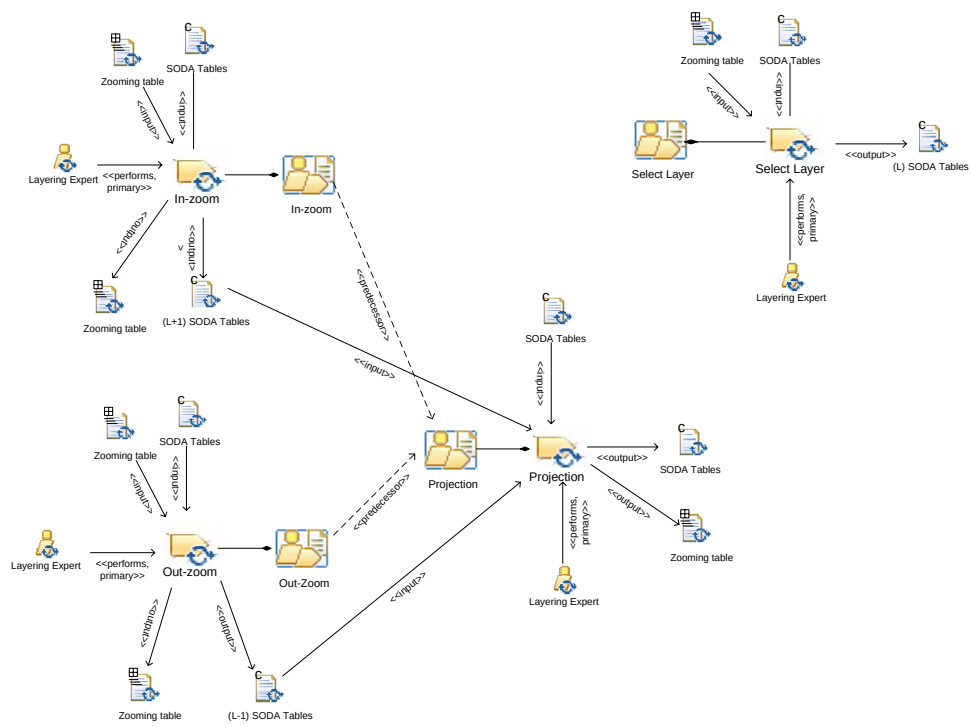


Figure 1.5: The Layering flow of activities, roles and workproducts

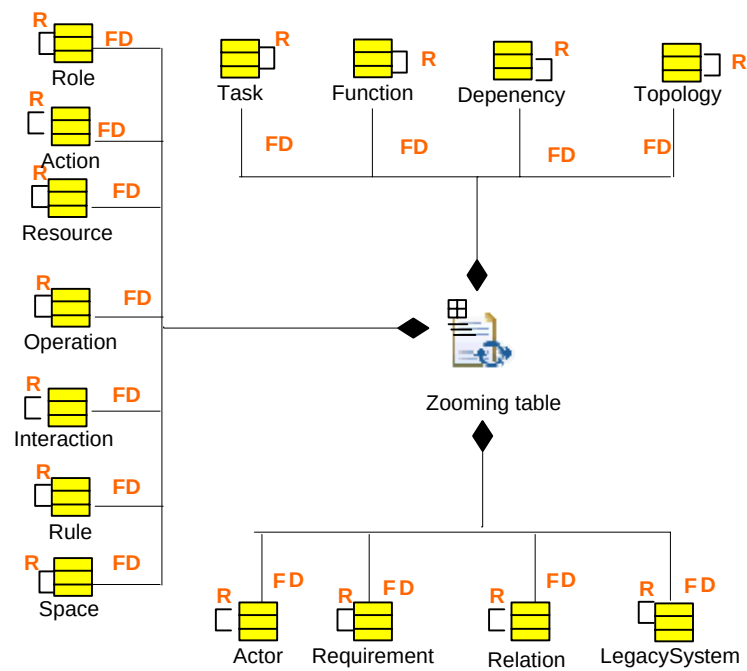


Figure 1.6: The Layering work products

The Layering is represented by means of a Zooming Table $((C)Z_t)$ (Figure 1.7) The Zooming table formalises the in-zoom of a layer into the more detailed layer; of course, the same table can be used to represent the dual out-zoom process.

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

Figure 1.7: $(L)Z_t$

Bibliography

- [1] aliCE Research Group. **SODA** home page. <http://soda.alice.unibo.it>.
- [2] Ambra Molesini, Andrea Omicini, Enrico Denti, and Alessandro Ricci. **SODA**: A roadmap to artefacts. In Oğuz 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.
- [3] 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.
- [4] Object Management Group. Software & Systems Process Engineering Meta-Model Specification 2.0. <http://www.omg.org/spec/SPEM/2.0/PDF>, April 2008.
- [5] 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, 2001. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.
- [6] Andrea Omicini. Formal **ReSpecT** in the A&A perspective. *Electronic Notes in Theoretical Computer Sciences*, 175(2):97–117, June 2007. 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA’06), CONCUR’06, Bonn, Germany, 31 August 2006. Post-proceedings.
- [7] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber*: Toward a theory of artifacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 2005. 1st International Workshop “Coordination and Organization” (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.