Environment in Agent-Oriented Software Engineering Methodologies

Ambra Molesini
Alma Mater Studiorum—Università di Bologna
viale Risorgimento 2, 40136 Bologna, Italy
ambra.molesini@unibo.it

Andrea Omicini, Mirko Viroli
Alma Mater Studiorum—Università di Bologna a Cesena
via Venezia 52, 47023 Cesena, Italy
andrea.omicini@unibo.it, mirko.viroli@unibo.it

Abstract

The key role of environment as a first-class abstraction in the engineering of MAS (multi-agent systems) is today generally acknowledged in the MAS community. However, the support for the notion of environment in today AOSE (agent-oriented software engineering) methodologies is still either absent, weak, or incomplete at best.

In this paper we first review, classify and compare existing AOSE methodologies according to their support for the notion of MAS environment. Then we suggest a general approach for extending existing AOSE methodologies toward full environment support.

1 Introduction

Different perspectives exist on the role that environment plays within agent systems: however, it is clear at least that all non-agent elements of a multi-agent system (MAS) are typically considered to be part of the MAS environment. Such elements typically include databases, Web services, communication infrastructures, physical devices, as well as the topology of the physical or virtual spatial domain [45]. Current practice in MAS considers the environment as an implicit part of the MAS that is often dealt with in ad hoc way. Indeed, following [45], we think that the environment should be considered as an *explicit* part of MAS, to be modelled and designed as a *first-class abstraction*. Along this line, MAS environment should be explicitly accounted for in the engineering of MAS, working as a new design dimension for agent-oriented software engineering (AOSE) methodologies.

While this general perspective on MAS environment is commonly shared in the agent community [46], today AOSE methodologies actually provide little or even no support to the modelling and design of MAS environment: for instance, a number of them—such as PASSI [7] and INGENIAS [34]—just support environment modelling, while others do not consider MAS environment at

all. Even more, it is often difficult to understand how the concept of environment is actually supported by AOSE methodologies, also in the cases when it is explicitly mentioned. In the PASSI meta-model, for instance, the modelling of environment—from the agent's viewpoint—is somehow "hidden" in the ontology description, and is scarcely highlighted; then, typical environmental elements like physical resources are represented as mere agent-related entities and are strangely not related to the ontology description. At the best of our knowledge, currently only three methodologies, SODA [25], OperA + Environment [9] and ADELFE [3], provide explicit support for both modelling and design of the environment in a MAS: however, each of them introduces its own characterisation of the environment, its constituents and topology.

Since a commonly shared viewpoint is lacking, it is seemingly useful first of all to analyse several AOSE methodologies studying the support they provide for the environment. The aim of this study is to understand how the different AOSE methodologies model and design the environment, and what environment abstractions and topology abstractions they provide. Hence, our aim here is not to present a general-purpose survey of AOSE methodologies, but rather to review the most relevant representatives by adopting the environment viewpoint—understanding if and how they deal with the engineering of MAS environment, in fact. As our first contribution, in this paper we classify AOSE methodologies in three different groups: *(strong-env)* strong environment viewpoint—methodologies that support both modelling and design of MAS environment; *(weak-env)* weak environment viewpoint—methodologies that support only the modelling of MAS environment; *(no-env)* no environment viewpoint—methodologies that do not explicitly model MAS environment. Accordingly, we first characterise the notion of environment in MAS (Section 2), then we detail our classification of AOSE methodologies (Section 3).

Next, we focus on the second contribution of this paper: in Section 4, we show how an explicit notion of MAS environment could be generally introduced in any AOSE methodology. Starting from no-env AOSE methodologies, we suggest how to transform them in weak-env methodologies (Subsection 4.2), and subsequently in strong-env methodologies (Subsection 4.3). Related works are presented in Section 5, then conclusions and future works follow in Section 6.

2 Environment

The MAS approach to software development is based on the idea of modelling software as a composition of autonomous entities, i.e. *agents*, situated in a common computational and/or physical *environment*, and interacting with each other and with resources in order to achieve individual and social goals.

Traditionally, the environment of a multi-agent system is however considered simply as the *deployment context* where agents are immersed in—which includes e.g. the communication infrastructure, the network topology, the physical resources available. In this case, the environment of a MAS is basically considered as an output of system analysis, and designers are passively subject to it. On the other hand, it is now increasingly recognised that the environment is a true design dimension of multi-agent applications [45]. It can encapsulate a significant portion of the system's complexity, in terms of services, mechanisms and responsibilities that the agents can fruitfully be freed of. So, along the line

of [42], in this paper we adopt a notion of environment that is not limited to the *external (MAS) environment* – the deployment context where the MAS has to work within –, instead we mostly focus on the *agent environment*, that is, the portion of the MAS that is external to the agents, but is anyhow part of the MAS and subject to the work of MAS engineers.

Accordingly, as far engineering is concerned, we recognise at least two main ingredients shaping the agent environment, which will be describe in detail in this section: environment abstractions—entities of the environment encapsulating some functions—and topology abstractions—entities of MAS environment that support the (either logical or physical) spatial structure.

2.1 Environment Abstractions

Following the work in [42], we start by considering a meta-model for MAS where the agent abstraction is not the only one populating the MAS: rather, we see also the MAS environment as decomposed in building blocks we call *environment abstractions*. An environment abstraction is an entity of the MAS environment encapsulating some functions or services for the agents. An agent perceives the existence of such abstractions in the environment, and it has a (possibly implicit) awareness of the opportunities they provide, and accordingly interacts with them in order to achieve individual as well as social goals. From the development viewpoint, environment abstractions are seen as *loci* where the designer can enforce rules, norms, and functions, regulating the agent social behaviour.

The notion of environment abstraction is a means for interpreting a number of components and entities that existing researches in the MAS context introduced which populates the MAS environment, both at the conceptual and at the implementation levels—all of them sharing the idea of being the target of agent interaction. Most existing infrastructures for agent environment are in fact seen as providing some incarnation of environment abstractions for the application at hand. Examples include *tuple centres* in TuCSoN [31], *co-fields* in TOTA [24], *shared virtual environments* of the automatic guided vehicles application in [47], *stigmergic fields* of the infrastructure for military operations in [38], *e-institution scenes* in AMELI [15], and so on.

Figure 1 provides a visualisation of how the concept of environment abstraction fits the different layers of a MAS—as discussed in detail in [42] [48]. At the lower layer the whole MAS (agents and environment) is immersed in a software and hardware deployment context; at the middleware layer, agent infrastructures (e.g. Jade [21]) are coupled with environment infrastructures—TOTA, TuCSoN, and the other above mentioned ones. At the application layer, while the former provides the agent abstraction (circles in the picture) —agent life-cycle, and typically direct agent communication—the latter provides environment abstractions (boxes in the picture), with which agents interact by an action/perception mechanism resembling more closely physical action rather than communication.

2.2 Topology Abstractions

Other than considering the various abstractions populating the environment, it is also useful to consider its spatial structure, or *topology*. In particular, abstrac-

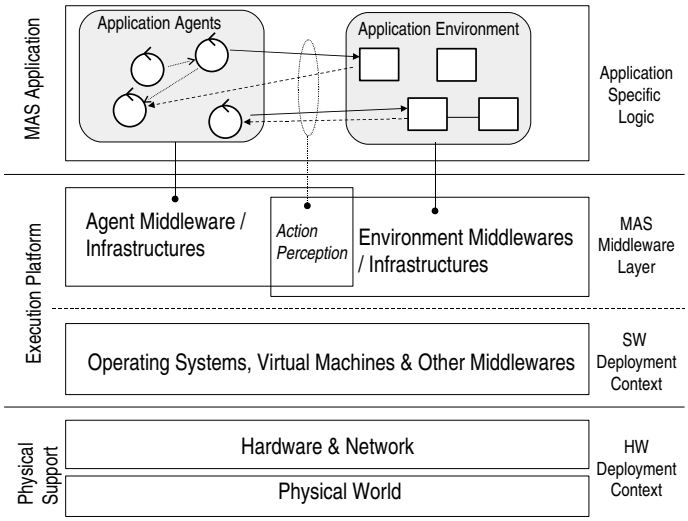

Figure 1: MAS layers with environment-based supports [42] [48]

tions that support topology as a first-class notion should be eventually considered part of AOSE methodologies—which is what we call *topology abstractions*. As a general definition, a topology is seen as a collection of neighbourhood sets, providing a structured notion of locality for the MAS, from local aspects up to the whole shape of the MAS. When considering a MAS as made of situated agents and environment abstractions, it is then natural to conceive MAS topology as a collection of sets of agents and environment abstractions. This affects notions like visibility among agents – whether an agent can see and communicate with another one –, visibility between agents and environment abstractions – whether an agent can see and interact with an environment abstraction –, and agent mobility—which new neighborhhod can be reached by an agent. This concept is rather important in MAS, because the visibility and accessibility of other entities strictly identifies which goals can be delegated (to other agents), and which services can be obtained (by environment abstractions), ultimately defining agent capability of achieving its goals. In many cases, the concept of topology is tied with the physical structure of the deployment context, including both the network topology and the real physical environment—as in the case of robot environment—but in general it can suitably take into account a virtual notion of space.

As an example we consider two MAS infrastructures from the literature, TOTA [24] and CArtAgO [37], which explicitly deal with the notion of environment topology. In TOTA the environment is seen as a dynamic network of nodes, hence its structure is determined by the neighbourhood relations. TOTA assumes networking capability for recognising connection and disconnection events. The specific nature of the network scenario determines how each node can found its neighbours, and the overall logical structure of the TOTA network. Nodes host agents, which can move step-by-step, and each node has a local tuple space abstraction used to support the fabric of co-fields.

CArtAgO is a general-purpose infrastructure for environments, based on *arti-*

facts as an incarnation of the notion of environment abstraction—following the A&A meta-model [36]. CArtAgO adopts the concept of *workspace* as a topology abstraction that can be used to define the topology of the computational environment. A workspace can be defined as an open set of artifacts and agents: artifacts can be dynamically added to or removed from workspaces, agents can dynamically enter or exit workspaces. A workspace is typically spread over the nodes of an underlying network infrastructure. Workspaces define topologies to structure agents and artifacts organisation and interaction, in particular workspaces are used as scopes for event generation and perception.

We note that even though the interplay between environment abstractions and topology abstractions can be considered as a key point in the modelling and design of the environment, existing MAS approaches generally do not explicitly take into account either of them [2]. This happens in spite of the fact that a large class of problems is characterised by unavoidable spatial features—several domains deal with space itself (e.g. geographical location) or a model of it (e.g. information flow in an organisational structure)—and by non-autonomous computational entities in the environment playing a significant role in the overall MAS goals. In particular, very few AOSE methodologies provide an explicit support for this sort of environment representation.

3 AOSE Methodologies & Environment

In this section we review some of the most significant AOSE methodologies according to their approach to the engineering of MAS environment. First of all we classify the methodologies in three different groups, and explain the rationale of this choice (Subsection 3.1). Then we overview the methodologies according to the previous classification. In particular, Subsection 3.2 presents the so-called strong-env methodologies, Subsection 3.3 illustrates weak-env methodologies, while no-env methodologies are presented in Subsection 3.4.

3.1 Classification of AOSE Methodologies

As mentioned in the introduction, one of the goals of this paper is to study AOSE methodologies in order to understand how they deal with the engineering of MAS environment. In particular, we focus on how methodologies model and design the environment, and on what kind of environment abstractions and topology abstractions they provide MAS engineers with. Among the many methodologies proposed in the AOSE field, in this paper we review a significant set of them, including Gaia [52][51], PASSI [7][8], Tropos [18][4], Prometheus [33][32], ADELFE [3][35], MESSAGE [5][17], INGENIAS [34][20] MaSE [11][49], ROADMAP [22], OperA+Environment [9] and SODA [26][25].

Each of these methodologies has obviously its own strengths and drawbacks. For example, Tropos is a good methodology for the requirements capturing and analysis but it does not consider environment engineering, while SODA is good in the environment engineering but does not say enough on requirements capturing. In this paper, however, our aim is not to present a classical, general-purpose survey and comparison of methodologies. Rather, we stick to the viewpoint of engineering MAS environment, and try to provide the reader with a clear overview of how existing AOSE methodologies deal with it. So, our classification

should not be taken as a statement of why a certain methodology is better or worse than another, but just as a measure of how much it supports MAS engineers in dealing with the environment.

Accordingly, though AOSE methodologies are quite heterogeneous, in this paper it is obviously useful to classify them according to the extent by which they tackle environment in MAS:

strong-env *Strong environment viewpoint*: this kind of methodologies allow MAS engineers to both model and *design* the environment at every stage of the methodology.

weak-env *Weak environment viewpoint*: methodologies belonging to this category take into account only the *modelling* of the environment.

no-env *No environment viewpoint*: in short, these methodologies do not consider environment at all, at least explicitly.

The difference between strong-env and weak-env methodologies is crucial here: in the former case, MAS engineers can fully design the environment through suitable dedicated abstractions provided by the methodology; in the latter case, environment structure and behaviour are taken as given a priori, and MAS engineers can only model them. In other terms, weak-env methodologies mostly deal with a notion of MAS environment as the external environment, while strong-env methodologies typically recognise the existence of an agent environment within a MAS that can be suitably modelled and designed.

3.2 Strong-Env Methodologies

Methodologies in the *strong-env* category promote MAS environment as a *first-class abstraction* [45] in the modelling and design of MAS since the early phases of the development process. In this category we find ADELFE, SODA and OperA+Environment.

3.2.1 ADELFE

ADELFE [3][35] has the Rational Unified Process (RUP) [23] tailored to take into account specificities of the design of *adaptive* MASs. In particular, the environment is studied since the WorkDefinition 2 (WD2, Final Requirements) stage, where MAS environment is characterised in terms of the entities (passive or active) that interact with the MAS. An *active entity* can behave autonomously and is able to dynamically interact with the system, instead a *passive entity* can be considered as a resource exploited by the system, which cannot change in an autonomous way (these will later become *objects* of the environment). In the subsequent stage, MAS context is studied through the interactions between the entities and the system. Finally, the environment is described in the terms of *accessibility*, *continuity*, *determinism* and *dynamism*.

In the WD3 (Analysis) stage, active entities are split in two sets: *cooperative* entities (that will become agents), and the *active entities* that autonomously evolve without having a goal. The latter are autonomous resource or active objects, and will remain simple objects in the environment. Afterwards, in WD3 interactions between entities (passive and active) are designed by means of standard UML [41] sequences or collaboration diagrams, as advised by RUP. Then,

the design of "environment abstractions"(passive and active entities) is done according to the RUP guidelines, determining packages, classes and using traditional design patterns. This implicitly defines the behaviour of the environment abstractions. Also, the topology of environment is implicitly defined by means of an agent's internal module called "representation module", which enables agents to create their own representation of the environment it perceives, and also by means of the study of MAS context in WD2.

3.2.2 SODA

SODA [25][26] is a methodology that focuses on interaction-related issues, like the engineering of societies and environments for MASs. In SODA the environment is studied since the requirement analysis step, where it is first characterised in terms of the "*legacy systems*" with which the MAS is required to interact. The abstract entity *relation* is there used to model the relationship among requirements and legacy systems. In the analysis step, the environment is modelled by means of *functions*, reactive activities that aimed at supporting other active activities (called *tasks*). Functions can either come from the legacy-system entities, or be designed ex-novo starting from the requirements. The structure of the environment is also modelled in terms of *topologies*, which are topological constraints over the environment. In particular, topologies could be also derived from functions, and could influence the achievement of the MAS tasks. In the analysis step the interactions among tasks and functions (and among functions themselves) are captured by means of *dependencies* that represent any sort of relationship among the abstract entities.

Then, in the architectural design stage, functions are assigned to *resources* (which are the abstractions providing functions by means of *operations*), topologies generate *workspaces* [37], and dependencies generate *interactions*. Workspaces are conceptual loci in the environment. An operation is characterised by a name and a set of parameters; *roles* (entities responsible for the achievement of tasks) interact with resources, allocated in the workspaces perceived by roles, by invoking operations and observing events generated from them. Interactions are defined as rules, which aim at enabling and bounding both the behaviour of the abstract entities (roles and resources) and the space of interactions.

Finally in the detailed design, resources are assigned to *artifacts* [29]. In addition also the interactions are mapped onto artifacts. As a result, artifacts construct and bound the space of interactions among agents and between agents and other artifacts, and allow agent societies to be engineered. Workspaces are the same defined in analysis phase, but in this phase a workspace can be defined as an open set of artifacts and agents: artifacts can be dynamically added to or removed from workspaces, agents can dynamically enter (join) or exit workspaces.

3.2.3 OperA + Environment

The OperA+Environment approach [9] adopts the analysis from OperA [14], a methodology that uses organisation structures and provides for open agent systems. The authors have refined the OperA with the introduction of the environment model and of the design phase.

	Environment Abstractions	*Behaviour of Abstractions*	*Interaction*	*Topology Abstractions*
ADELFE	passive and active objects	implicitly designed	explicitly designed	implicitly defined
SODA	legacy-systems, functions, resources and artifacts	explicitly designed	explicitly designed	workspaces
OperA + Environment	resources, services and coordination facilities	explicitly designed	explicitly designed	not supported

Figure 2: Strong environment viewpoint in AOSE methodologies

The environment model in the analysis phase specifies the *resources* that are available for the agents, like databases, etc. The model also specifies the available *services*. These can be like white or yellow pages, but also the Mathematica package that supports all kinds of calculations. In the model also specifies the way the system can interact with the environment.

In the design phase the Infrastructure Model makes use of the design models from two existing methodologies: SODA [27] and Gaia [50]. This model consists of a resource model, a service model, and a model of coordination facilities. Each resource model contains a specification of the resource (e.g. document, database, or library) in terms of various qualities and quantities, access permissions, and admissible actions. The service model defines which services (e.g. any activity that processes information) can be delivered to a certain service request. The services can be defined as abstract operations in terms of input and output functionality. For each service, the model specifies the permissions needed to use the service by an agent playing a certain role. It also specifies the quality of service, such as the maximal delay in providing the service, the format of messages it can process, and the protocols it can support. Finally, *coordination facilities* are mechanisms that can be used by the agents to coordinate their activities. Examples are synchronization, tuple spaces, subscribe notify design pattern, or various types of protocols.

3.2.4 Strong-env Methodologies at a Glance

Table 2 summarises how the methodologies support the key elements of strong environment viewpoint, and classifies them along four dimensions: environment abstractions, behaviour of abstractions, interaction, and topology abstractions.

3.3 Weak-Env Methodologies

Methodologies in this category take into account the environment as an entity which is given a priori, which MAS engineers can only model. Environment modelling may occur at different stages of methodologies. In this category we find Gaia, PASSI, MESSAGE, INGENIAS, Prometheus, and ROADMAP.

3.3.1 Gaia

Gaia [51][52] drives the designer in the development of MAS by means of the identification of several organisational models and the interaction among them. In Gaia, MAS environment is studied since the analysis phase, in particular the

environmental model is intended to make explicit the features of the environment in which the MAS will be immersed. The identification and modelling of the environment involves determining all the entities and resources that the MAS can exploit. Gaia suggests treating the environment in terms of *abstract computational resources* (such as variables or tuples) made available to agents for sensing, effecting or consuming. Following such identification, the environmental model can be viewed as a list of resources characterised by the type of the actions that the agents can perform on it. In the *preliminary role model* roles are related to environment by means of *permissions*. Permissions discipline how roles can access to environmental resources and possibly change or consume them. In order to represent permissions, Gaia adopts the same notation used for environmental resources. However, the attributes associated with resources no longer represent what can be done with such resources (i.e., reading, writing, or consuming) from an environmental perspective, but rather what the agents playing the role are be allowed to do (or not to do) to accomplish the role's goal(s).

In the architectural design, the preliminary role model is completed by the addition of roles generated by the architectural choice. For each new role, permissions associated to environmental resources are defined. In the detailed design, agents are associated with roles, so that agents are related to environmental resources.

3.3.2 PASSI

PASSI (Process for Agent Societies Specification and Implementation, [7][8]) is a step-by-step methodology for the design and development of multi-agent societies. In PASSI, the environment is studied since the Agent Society Model, in particular in the Ontology Description Phase. In the Domain Ontology Description, agents know the environment through the abstractions of *Concepts* (categories, entities of the domain), *Predicates* (describing the state of the instance of concepts that actually occur in the environment), and *Actions* (performed in the domain), along with their mutual relationship. In the Communication Ontology Description the agent interactions are represented: in each communication, it is important to introduce the proper data structures (selected from elements in Domain Ontology Description) within each agent in order to store the exchanged data. In the PASSI meta-model the concept of *Resource* also appears, which represents a tangible entity of the environment with which agents can interact, and which is a part of agents' environment awareness. This concept only relates with agents, and no relation exists with ontology elements.

In the Role Description Phase PASSI introduces a relationship called *Resource Dependency*: a role depends on another for the availability of an entity (indicated by a *resource name*). However, it is unclear whether this resource is the same that appears in the meta-model.

3.3.3 MESSAGE

MESSAGE [5, 17] has adopted the Rational Unified Process (RUP) as a generic software engineering project lifecycle framework. MESSAGE adopts the abstraction of Resource as a Concrete Element in the system. Resource is used to represent non-autonomous entities such as databases or external programs

used by Agents. Standard object-oriented concepts are adequate for modelling Resources.

Then, in the analysis phase, several views take into account the concept of resource. In particular the Organisation view shows Concrete Entities (Agents, Organisations, Roles, Resources) in the system and its environment, as well as coarse-grained relationships between them (aggregation, power, and acquaintance relationships). An acquaintance relationship indicates the existence of at least one Interaction involving the entities concerned. In addition, the Domain view shows the domain specific concepts and relations that are relevant for the system under development. The Agent/Role view specifies for each agent/role what resources it controls.

The purpose of the design phase is to define computational entities that represent the MAS appearing at the analysis level. Analysis entities are thus translated into subsystem, interface, classes, operation signatures, algorithms, objects, object diagrams, and other computational concepts.

3.3.4 INGENIAS

INGENIAS [20][34] provides a notation for modelling MAS, and a well-defined collection of activities to guide the development process of MAS from tasks to code generation. INGENIAS uses the concept of *viewpoint* as MESSAGE. Each viewpoint in INGENIAS is constructed following two sets of activities structured into activity diagrams: one set aims at elaborating the view at the analysis level, whereas the other focuses on the design. In particular the *Environment Viewpoint* defines the entities with which the MAS interacts, which could be *Resources* (such as CPU, File Descriptors or memory), *Other Agents* (from existing organisations), and *Applications* (expressing the perception and action of the agents, producing the events that can be observed).

In the Development Process INGENIAS suggests to identify in first place all the software systems (usually non-agent based) that will coexist with the MAS. By using the environmental model, it is possible to consider dependencies—in terms of agents' perceptions and actions—with legacy and proprietary systems from the beginning.

3.3.5 Prometheus

Prometheus [32][33] is intended to be a practical methodology: it aims at being complete and detailed, and to be usable by industrial software developer. In Prometheus the environment is modelled since the System Specification phase. In particular, the environment is defined by describing the percepts available to the system, the actions that it will be able to perform, and any external data that are available as well as any external bodies of code.

In the Architectural Design phase the environment appears in the design of the overall structure of the systems: the overview diagram captures the agent types, the boundaries of the system, and its interface in terms of actions and perceptions, but also in terms of data and code that are external to the system.

In the Detailed Design phase the overview diagram is used to develop internals of agents and interaction protocols.

	Environment Abstractions	*Topology Abstractions*
Gaia	abstract computational resources	not supported
PASSI	resource, concepts, predicates and actions	not supported
MESSAGE	resources	not supported
INGENIAS	resources, other agents and applications	not supported
Prometheus	actions, perceptions, external data and code	not supported
ROADMAP	static objects and objects	zones, zones schema, constraints

Figure 3: Weak Environment Viewpoint in AOSE methodologies

3.3.6 ROADMAP

The ROADMAP [22] methodology extends the first version of Gaia with several features: support for requirements gathering, explicit models to describe the domain knowledge and the execution environment, levels of abstraction during the analysis phase, to allow iterative decomposition of the system, explicit models and representations of social aspects and individual agent characteristics, from the analysis phase to the final implementation, runtime reflection, modelling mechanisms to reason and change the social aspects and individual agent characteristics at runtime. The environment model is proposed to provide a holistic description of the system environment and provides both the environment abstractions and the modelling of the topology. By formally describing the environment, the authors create a knowledge foundation on which environment changes are handled consistently. The environment model is derived from the use-case model. The model contains a tree hierarchy of *zones* in the environment, and a set of *zone schema* to describe each zone in the hierarchy. A zone schema includes a text description of the zone, and the following attributes: *static objects*, *objects*, *constraints*, *sources of uncertainty* and assumptions made about the zone. Static objects are entities in the environment whose existences are known to the agents, with which the agents do not interact explicitly. Objects are similar entities with which agents interact. Sources of uncertainty in the environment are identified and analysed. The zone hierarchy uses OO-like inheritance and aggregation to relate zones and various objects inside zones.

3.3.7 Weak-env Methodologies at a Glance

Table 3 summarises the environment-related abstractions adopted by weak-env methodologies. This table, unlike Table 2, only presents environment and topology abstractions, while other columns are omitted: the behaviour of the abstractions is omitted because here MAS environment is only modelled and not designed, while interaction is omitted because it is explicitly designed in every methodology.

3.4 No-Env Methodologies

Methodologies in this category do not deal with the concept of MAS environment. Tropos and MaSE can be taken as the representatives of this category.

3.4.1 Tropos

Tropos [4][18]is a requirements-driven methodology and adopts the abstractions offered by i*, a modelling framework proposing concepts such as actor (actors can be agents, positions or roles), as well as social dependencies among actors, including goal, softgoal, task and resource dependencies. These concepts are used in all software development phases of Tropos, from the early requirements analysis down to the actual implementation. Resources are always involved in dependencies among actors. A resource is not an abstraction that models the environment, instead it could be thought of as an item for knowledge exchange among actors: more precisely, a resource represents a physical or an informational entity that one actor may want and another could deliver.

3.4.2 MaSE

MaSE [11][49] guides a designer through the software lifecycle from a prose specification to an implemented agent system. MaSE is independent of a particular MAS architecture, agent architecture, programming language, or message-passing system. The abstractions used by MaSE are goals, tasks, roles and agents. Each goal is associated to the role that achieves it. Any mention of separate machines or other forms of distribution requires one role for each "side" of the distributed relationship. Interfacing with an external source is the same. One role may interface with the source while another may be required to bridge the gap back to the system. This is also true for any database, file interface, or user interface in the system. A user interface implies a role by itself, and should be separated from other roles as if it were a distinct data source. Then role are assigned to agents—so in the end everything is an agent in MaSE.

3.4.3 No-env Methodologies at a Glance

In this subsection we have shortly reviewed two different examples of methodologies that do not explicitly consider the environment, yet. Typically in this kind of methodologies the environment is not totally forgot, instead it is typically represented by means of agents–so, not as a first-class entity. A common viewpoint that we share [45] is that using agents for modelling everything means somehow to pervert the nature of the agent abstraction itself. For instance, the agent abstraction is not the most suitable one for modelling resources, for which crucial agent concepts like autonomy and proactiveness simply do not apply.

4 Introducing Environment in AOSE Methodologies through Artifacts

In this section we present a viable approach to introducing the notion of environment within existing AOSE methodologies. First of all, we identify some environment-related abstractions among the existing AOSE methodologies and

MAS infrastructures that could be general enough to fit most methodologies (Subsection 4.1). Among the available alternatives, in this paper we experiment with the abstractions introduced by the A&A meta-model [36] – namely, *artifacts* and *workspaces*–, which are adopted by the SODA methodology, and provided by the CArtAgO infrastructure. While this choice is obviously not the only possible, in the remainder of this section we discuss how AOSE methodologies can be extended with artifacts and workspaces to generally deal with the engineering of environment in MAS. We also suggest that this approach is potentially applicable to every AOSE methodology, since it does not requires many changes: the only modifications required concern the design of the interaction protocols that tie the agents to the environmental entities, and obviously also the insertion of the proposed environmental models into all the phases of the methodology to be extended

In the discussion that follows, we keep no-env and weak-env methodologies distinct: whereas they both do not have a complete handling of environment, they have anyway a different approach toward it. So, in the next subsections we first show how to introduce environment in no-env methodology (Subsection 4.2), then discuss how to transform a weak-env methodology in a strong-env one (Subsection 4.3).

4.1 Environment and Topology Abstractions

When dealing with the introduction of the environment in a methodology, the first issue to consider is the choice of the most suitable environment abstraction. The notion of environment abstraction as introduced in Section 2 is a high-level notion: while it is a good way for discussing about the environment in general, it might be practical to exploit a more concrete notion at the methodological level. In fact, since it is an abstraction over the real entities used by infrastructures, there is no clear definition of its concrete structure. Instead, it is useful to answer questions such as "how to model the environment abstractions?", and "how to design the environment abstractions?".

In order to choose a proper concrete environment abstraction it might be useful to take inspiration from the abstractions already used by both existing infrastructures for MAS and AOSE methodologies. The entities provided by infrastructures seem to be too specific because they are typically conceived for a particular application domain, so they are not easy to adapt to general modelling—as required by a general-purpose methodology. On the other hand, AOSE methodologies—especially in the methodologies in the weak-env group—use very different notions of environment abstractions, belonging to different cognitive levels: from lower level ones such as *external data and code*, *ontologies* and *objects*, to higher level ones such as *resources*, *abstract computational resources*, and *artifacts*.

This problem, associated with the typical openness requirements for MAS, leads to see the agent interaction space as spanning over different levels. There, agents cannot interact with the other components (agents and environment abstractions) of the MAS in an uniform way: they communicate with each other via high-level languages, while often using lower-level languages – one for each different kind of environment abstraction – as a way to interact with the environment. From the general software engineering point of view this situation is not acceptable because it produces an unnecessary complication in the engi-

neering of both the agent internal structure and the interaction protocols. The proliferation of environment abstractions seems to be more a drawback than a strength because it does promote a clear and common view of environment engineering. In addition it also seems an obstacle in the path toward the creation of a general purpose AOSE methodology.

Seemingly, one of the most promising environment abstraction is the notion of *artifact*—as highlighted in Subsection 2.1. In fact, artifacts are generally defined as both conceptual and runtime entities that mediate agent activities [43], providing some kind of function or service that agents can fruitfully exploit to achieve their individual or social objectives. In particular, in the Agents & Artifacts (A&A) meta-model [28][30] recently proposed for MAS engineering, artifacts—along with agents—are adopted as the basic building blocks to engineer complex software systems. Artifacts are the basic abstractions to represent passive, function-oriented building blocks, which are constructed and used by agents, either individually or cooperatively, during their working activities. Many sorts of artifact can populate a MAS: in particular, artifacts are used to mediate between individual agents and the MAS (individual artifacts), to build up agent societies (social artifacts), and to mediate between a MAS and an external resource (environmental artifacts) [29]. In addition, artifacts have been fruitfully introduced in a methodology (SODA) already, thus showing a possible path for introducing artifacts within a methodology.

The second issue concerns the choice of the most suitable abstraction for modelling MAS topology. As for the environment abstraction, there is the need of a general-purpose abstraction able to capture different deployment contexts. However, differently from the environment abstraction case, the topology abstractions provided by both MAS infrastructures and AOSE methodologies are few, and the level of abstraction provided by them is very low. Here, *workspaces* – introduced by CArtAgO in the A&A meta-model and adopted also by SODA – seem a suitable choice since they are enough general for abstracting every topology, and are also closely tied with the concept of artifact.

4.2 From No-Env to Weak-Env Methodologies

The introduction of the environment in a methodology implies a significant modification of the first phases of the methodology, and as a consequence a partial modification of the other phases.

Requirement Specification

The first step is a thorough analysis of the requirements specification with the following goals—see Figure 4-top:

(i) Detecting the presence of legacy systems that the system should interact with. Legacy systems are computational systems that have already been installed and are working, and that the designer of a new system should exploit without the possibility of changing them. These systems obviously will become part of the environment of MAS, since they often provide several services to the system to be developed.

(ii) Recognising those requirements concerning function-oriented and passive entities, which should not become agents for they have no goals and au-

tonomy. However in some cases it is not so easy to understand when an entity is not an agent, since the concept of "function-orientation" is not always tied with the concept of passivity.

However it is widely accepted that a pheromone is part of the environment because it is not an autonomous active entity.

So, recognising when a requirement deals with non-agent entities is a non-trivial job as well, because typically a requirement deals both with several (future) agents and non-agent entities, and such entities are closely tied. As a remark, here such function-oriented entities are supposed to be already implemented or anyway provided by someone, so they are only modelled. The design of these entities is one of the topics of next subsection.

(iii) Individuating topological constraints over the structure of the deployment context. Typically in the early phase of requirement analysis, such constraints represent physical ties over the environment, but sometimes some constraints could come from requirements.

It would be useful to keep track of the relations between requirements and legacy systems / function-oriented entities in order to create those links that will become the interaction protocols between agents and artifacts. The protocols will be properly designed later in the design phase. In addition, also the relations between requirements and constraints could be examined in order to facilitate the definition of workspaces.

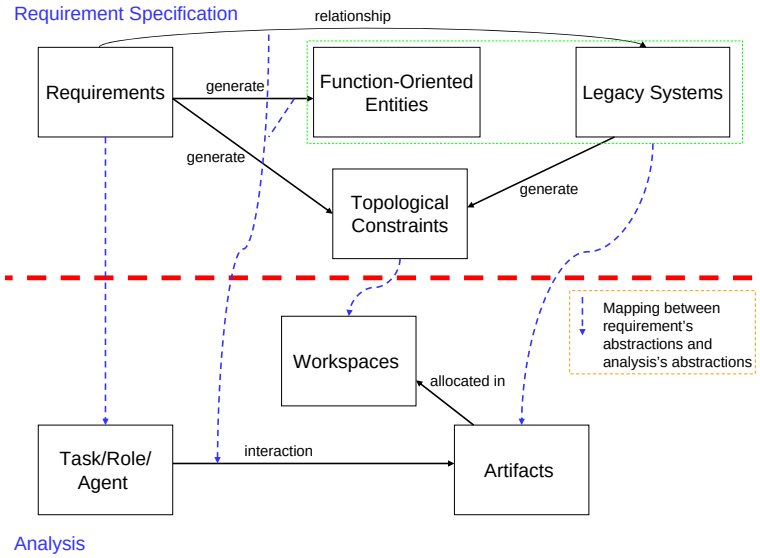

Figure 4: Relations between requirement's abstractions (top) and analysis's abstractions (bottom)

Analysis

In the analysis phase, a model of environment composed of artifacts and workspaces is built: both the legacy systems and the function-oriented entities are modelled as artifacts (see Figure 4). According to [36], this means specifying for each of them the *usage interface* (the operations provided by the artifact), the *operating instructions* (how to access the artifact), and the *function description* (what services are provided by the artifact). Then the relationships previously identified, between requirements and the function-oriented entities / legacy systems, are now refined in more concrete relations between the system's abstract entities (adopted in the other original models of the methodology) and artifacts (see Figure 4). Finally, the topological constraints generate the workspaces for structuring the environment. Sometimes, intersection and nesting of workspaces are here necessary in order to create articulated topologies coming from very complex constraints. Moreover, artifacts are allocated to the most suitable workspace(s) according to the specific topological constraints.

Design

In the design phase(s) of the methodology, the environment model does not change, only the interaction protocols are designed starting from the relations sketched out in the analysis phase. If the methodology schedules other phases such as fast prototyping or code generation, these should be adapted in order to support the generation of the new interaction protocols between agents and the entities in the environment. As interactions with the environment are uniformly seen as interactions between agents and artifact, all the above phases should apparently be no more complex than the design of MAS relying on standard agent communication protocols. In addition, if the methodology to be extended presents deployment models, these should be changed so as to support the structure of environment coming from workspaces. Simply, that model will need to be extended so that it can support the new "workspace" abstraction and the relations between workspaces.

An Example: Tropos from No-Env to Weak-Env

As an example methodology to ground our discussion we consider Tropos, which is a methodology where requirements are well structured and documented. Tropos could represent an exception in the proposed method, since it adopts a particular framework for the requirement analysis and needs a careful investigations. Requirements analysis in Tropos is split in two main phases: Early Requirements and Late Requirements analysis [4]. More precisely, during the first phase, the requirements engineer identifies the domain *stakeholders* and models them as social actors, who depend on one another for *goals* to be achieved, plans to be performed, and resources—remember that this concept does not represent an environment abstraction—to be provided.

Following our method, first of all in the Early Requirement analysis the identification of legacy system is necessary. So while the requirements engineer identifies the domain stakeholders he/she should also analyse the system specification in order to discover those legacy systems the new MAS will interact with. In particular the requirements engineer should identify the relationships among the legacy systems and stakeholders (if any exist). If legacy systems are related

with actors' goals, then such relations need to be traced. In addition, the requirements engineer should also recognise if legacy systems are subject to some particular constraints over the environment structure. Legacy systems are then modelled as artifacts, and the topological constraints generate the workspaces. All the relationships among the entities should be traced.

In the Late Requirements analysis, the conceptual model previously identified is extended including a new actor representing the system, and a number of dependencies with other actors. These dependencies define all the functional and non-functional requirements of the system-to-be. Following our approach, this is the right stage for individuating the entities of *(ii)* and the other topological constraints. In particular, the requirements engineer should check carefully both the functional and non-functional requirements searching those function-oriented entities that either provide services to actors or support them in the achievement of their goals. Topological constraints typically come from non-functional requirements, so they should be deeper studied for recognising all the constraints over the environment structure. Subsequently, these entities become respectively artifacts and workspaces, and the links among artifacts and goal / actor should be traced.

The Architectural Design and the Detailed Design phases in Tropos focus on the system specification, according to the requirements resulting from the above phases. Architectural Design defines the system's global architecture in terms of sub-systems, interconnected through data and control flows. Sub-systems are represented as actors, and data/control interconnections are represented as dependencies. Following our approach, in this phase the interaction protocols among actors/agents and artifacts should be refined according to the specific architectural structure adopted. In particular the choice of a specific architecture could lead to refine the actor model and as a consequence new and/or more refined relations among actors and artifacts could be discovered. In a similar way, the choice of the architecture could lead to identify new topological constraints over the environment and as a consequence a new refined workspaces structure could be outlined.

The architectural design provides also a mapping of the system actors to a set of software agents, each characterised by specific capabilities. The Detailed Design phase aims at specifying agent capabilities and interactions. At this point, usually, the implementation platform has already been chosen and this can be taken into account in order to perform a detailed design that will map directly to the code. Following our approach, in this phase the interaction protocols among agents and artifacts should be designed.

Finally the Implementation activity in Tropos follows step by step, in a natural way, the detailed design specification on the basis of the established mapping between the implementation platform constructs and the detailed design notions. Following our approach in this activity it is necessary to implement the interaction protocol among artifacts and agents and to organise the environment structure according to the workspaces structure.

In all, then, our method makes the introduction of the environment in Tropos not so difficult. Similar considerations could be done also for MaSE, and possibly for any other no-env AOSE methodology.

4.3 From Weak-Env to Strong-Env Methodologies

Weak-env methodologies already deal with some kinds of environment abstractions. In order to simplify their treatment, and also to make methodologies homogeneous, as a first step we suggest the artifact-based wrapping of the different existing environment abstractions explained in Subsection 4.1. Then, in order to complete the analysis phase we can then go back to the requirements, and act in a similar way as discussed in the previous subsection. However, the identification of legacy systems is not necessary here, since they have yet been modelled according to the original environment model.

Requirement Specification

The recognition of function-oriented entities—as highlighted in point *(ii)* of requirement specification in the previous subsection—here is not simply aimed at modelling entities already implemented in the environment, but also at discovering new function-oriented entities that are necessary to the MAS but have not already been identified. Obviously, the relations of these new entities with other abstract entities should be soon devised out: first, by outlining the links between function-oriented entities and other requirements, then sketching out the more refined relations between function-oriented entities and the abstract entities adopted by the methodology—see Figure 4-top.

Subsequently, if the methodology does not deal with topological aspects, these should be introduced finding out the topological constraints from the requirements.

Analysis

In this phase the function-oriented entities are modelled as artifacts. In addition the discovery of new artifacts is also possible during this phase, when the requirements are well structured and modelled by means of appropriate abstract entities such as roles, tasks, goals and so on. In this case, it is easier to understand whether an active entity needs some kinds of services which are not present yet in order to achieve one or more objectives.

Finally, workspaces are generated from topological constraints, and allocating artifacts and agents inside them. As for artifacts, during the analysis phase new workspaces could be discovered so as to better structure the environment.

Design

In the design phase, the discovery of new artifacts is also possible. In this phase the choices about how to structure the system are made, and the detection of new services is not so difficult. As an example, let us suppose to use a group of roles/agents for the achievement of particular complex task/goal. Borrowing a group of roles/agents as a responsible of one task/goal leads to manage the problem of how to coordinate these entities so as to achieve the task/goal. In this case, the solution could be the adoption of a suitable coordination artifact [30, 44] for managing coordination among roles. This artifact could not be discovered in the previous phases because it comes from a specific design choice—whereas other choices could lead to other solutions, as for example the design of complex coordination protocols among agents/roles without using artifacts.

Then, first of all the interaction protocols between agents and artifacts are designed starting from the relations sketched out in the analysis phase; moreover, the required artifacts are designed. The "external behaviour" of an artifact is defined by means of interaction protocols in which it is involved. The internal design could be made by following a traditional software engineering design, as for example the object oriented design, since an artifact looks like an object. Since specific guidelines for the internal artifact design have not been defined yet, the only possible choice is the adoption of traditional design techniques. However, some guidelines about the design of operating instructions are presented in [44].

If the methodology supports other phases such as fast prototyping or code generation, these should be adapted in order to support the generation of the new interaction protocols between the systems and artifacts, as well as also the code generation for the artifacts and for the workspaces. In addition, if the methodology presents deployment models, these should be changed for supporting the structure of environment as it comes from workspaces.

An Example: Gaia from Weak-Env to Strong-Env

For the sake of concreteness, we sketch here how to transform Gaia in a strong environment methodology. In Gaia the environmental model can be viewed as a list of resources, each denoted by a symbolic name, characterised by the type of actions that the agents can perform on it, and possibly associated with additional textual comments and descriptions [52]. Following our approach, first of all the resources already belonging to the environmental model should be wrapped by means of artifacts; then the requirements should be investigated looking for other function-oriented entities and for all the existing legacy systems. Both should become artifacts. In addition the requirements engineer should recognise if legacy systems are subject to some particular topological constraints over the environment structure. Such constraints eventually generate the workspaces. All the relationships among the role identified in the Preliminary Role Model and the artifacts should be traced during the construction of the Preliminary Interaction Model. Finally during the Analysis phase topological aspects could be deduced also from organisational rules, generating workspaces and allocating roles and artifacts inside them.

In the architectural design, the choice of the best system architecture is made, leading to the refinement of the role model—adding new specific roles coming from the choice of the architecture—and the interaction model. Following our approach, in this phase the choice of the particular architectural structure typically leads to the discovery of new artifacts needed to support the new role activities. In addition, the architecture choice leads to refine the interaction model and as a consequence new and / or more refined interaction protocols among roles and artifacts could be discovered. In a similar way the choice of the architecture could lead to identify new topological constraints over the environment and as a consequence a new refined workspaces structure could be delineated.

Finally in Gaia, in the detailed design stage, roles are assigned to agents, and artifacts should be internally designed according to a traditional software engineering design. In addition the interaction protocols among agents and artifacts should be designed.

Again, this example shows how artifacts and workspaces could be adopted inside a weak-env methodology to make it a strong-env one.

5 Related Work

Literature dealing with both environment and agent-oriented methodologies is not abundant. Works of that sort are usually more focussed on sociality features, so they typically present some comparison among methodologies (one example is in [19]) and simply point out whether a methodology supports the concept of environment or not. As an example in [40] a good framework for the evaluation of the methodologies is presented, accounting for a large number of agent features (organisations, roles, beliefs, desires and so on) as well as some criteria for the evaluation of the development process—however, environment is not considered at all.

At the same time, so little research has been devoted to the issue of how to introduce the notion of environment in a methodology—an AOSE methodology, in particular. At the best of our knowledge, there is only one work [12] investigating how to introduce an environmental model in a methodology. The methodology considered there is O-MaSE (Organisation-Based Multiagent System Engineering [10]), an evolution of MaSE dealing with the design of organisational MAS. Authors describe an approach to the modelling of MAS interactions with its environment: the key concepts in their approach are capabilities and the environment model. In particular, the environment is modelled as a set of objects/agents, and a set of relations between such objects/agents. Through a set of capabilities belonging to agents, agents have access to a set of operations that they may perform upon environment objects, whose effect are governed by environmental laws. This work is obviously very interesting in this context, and in the conclusions the authors argue about a possible integration of the concepts of their AEI (Agent-Environment Interaction) Model into existing methodologies: however, they do not specify *how* to introduce their key concepts into other methodologies, so their approach turns to be too strict in scope. Also, even in principle, such a process looks not so easy: in fact, the meta-model of AEI is largely tailored upon the MaSE meta-model, and some of the elements implicitly require the MaSE specific agent model. As a result, the introduction of environment according to AEI seems to be potentially invasive, since it would involve the creation of a new model for the environment and also affects the agent model: in fact, adopting the AEI meta-model would tie agents with the concept of "capabilities", thus requiring a substantial change to the original agent model of the methodology to be extended. Even more, AEI does not consider topology at all.

Another interesting work is the methodology proposed by Simonin and Gechter [39] that establishes the link between the representation of the problem, expressed as environmental constraints, and agent behaviours, which are regulation items of the environmental perturbations. The environment is modelled by means of the definition of its structure (the topology) and the laws that govern its dynamics. Then no environment abstractions are considered by the methodology for representing the environment, so it is not clear how and where the environment laws are enforced.

Finally, also the work on methodology fragmentation conducted by IEEE-

FIPA Methodology Technical Committee [16] is very interesting in our enviroment-AOSE perspective. In fact, the Committee has developed a method for assembling pieces out of the methodology processes starting from a meta-model of methodologies. An interesting fragment ([1]) is provided by the ADELFE methodology: there, in fact, the environment model is extracted from the methodology and a new fragment is create. However, the fragment uses the specific environment abstractions adopted by ADELFE, which appear not general enough to be widely applied to other methodologies. In addition the fragment is not yet well documented, and it is then not so clear what is actually needed for merging the fragment within a methodology like SODA.

6 Conclusions and Future Work

The key role of environment as a first-class abstraction in the engineering of MAS is today generally acknowledged in the MAS community. In this work we have highlighted the importance of both environment abstractions and topologies for the complete modelling of MAS environment.

However, we have observed that in the AOSE field a small number of methodologies actually deal with environment as a first-class abstractions, while some others provide MAS engineers with only one model of the environment, and a few others do not consider environment as a first-class abstraction at all, yet. Furthermore, even in the case of methodologies actually modelling MAS environment, such feature is often somehow hidden or not-well documented. In our survey, we have tried to understand in depth how each methodology handles MAS environment, and which sorts of environment abstractions it adopts. The first result of our work is then a classification of methodologies in three different categories: strong-environment viewpoint (methodologies that consider environment as a first-class abstraction), weak-environment viewpoint (methodologies that only model environment) and no-environment viewpoint (methodologies that do not consider environment).

As a further contribution, we have then proposed a possible method for introducing the treatment of environment in no-env methodologies—thus transforming them in weak-env methodologies; afterwards, we have shown how to transform weak-env methodologies in strong-env methodologies. To this end, we have shown how to exploit artifacts as a general-purpose environment abstraction, and workspaces as an abstraction for modelling topologies. These abstractions have already been fruitfully used in a strong-env methodology.

Further investigations are obviously necessary in order to fully demonstrate the general applicability of our method. Accordingly, we plan to actually apply our approach to different methodologies, and to make it as detailed as possible in order to make its application easy and immediate. In particular we plan to create some specific fragments for treating MAS environment in each phase of the development process. This should simplify the introduction of the environment in the no-env methodologies, as well as the improvement of the weak-env methodologies. In addition, this could promote the definition of an effective pattern for the creation of new methodologies dealing with MAS environment as a first-class abstraction.

References

[1] ADELFE. http://www.pa.icar.cnr.it/~cossentino/FIPAmeth/docs/adelfe_july05.pdf.

[2] Stefania Bandini, Sara Manzoni, and Carla Simone. Supporting the application of situated cellular agents in non-uniform spaces. *Future Generation Computer Systems*, 21(4):627–631, 2005.

[3] Carol Bernon, Valérie Camps, Marie-Pierre Gleizes, and Gauthier Picard. Engineering adaptive multi-agent systems: The ADELFE methodology. In Henderson-Sellers and Giorgini [19], chapter VII, pages 172–202.

[4] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Tropos: An agent-oriented software development methodology. *Autonomous Agent and Multi-Agent Systems*, 8(3):203–236, May 2004.

[5] Giovanni Caire, Wim Coulier, Francisco J. Garijo, Jorge Gomez, Juan Pavòn, Francisco Leal, Paulo Chainho, Paul E. Kearney, Jamie Stark, Richard Evans, and Philippe Massonet. Agent oriented analysis using Message/UML. In Michael Wooldridge, Gerhard Weiss, and Paolo Ciancarini, editors, *Agent-Oriented Software Engineering II*, volume 2222 of *LNCS*, pages 119–135. Springer, 2002. 2nd International Workshop (AOSE 2001), Montreal, Canada, 29 May 2001. Revised Papers and Invited Contributions.

[6] Cristiano Castelfranchi and W. Lewis Johnson, editors. *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, Bologna, Italy, 15–19 July 2002. ACM Press.

[7] Massimo Cossentino. From requirements to code with the PASSI methodology. In Henderson-Sellers and Giorgini [19], chapter IV, pages 79–106.

[8] Massimo Cossentino, Luca Sabatucci, and Antonio Chella. Patterns reuse in the PASSI methodology. In Andrea Omicini, Paolo Petta, and Jeremy Pitt, editors, *Engineering Societies in the Agents World IV*, volume 3071 of *LNAI*, pages 294–310. Springer-Verlag, June 2004. 4th International Workshop (ESAW 2003), London, UK, 29–31 October 2003. Revised Selected and Invited Papers.

[9] Mehdi Dastani, Joris Hulstijn, Frank Dignum, and John-Jules Ch. Meyer. Issues in multiagent system development. In Castelfranchi and Johnson [6], pages 922–929.

[10] Scott A. DeLoach. Engineering organization-based multiagent systems. In Alessandro F. Garcia, Ricardo Choren, Carlos José Pereira de Lucena, Paolo Giorgini, Tom Holvoet, and Alexander B. Romanovsky, editors, *Software Engineering for Multi-Agent Systems IV, Research Issues and Practical Applications*, volume 3914 of *LNCS*, pages 109–125. Springer, 2006. 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2005), St. Louis, Missouri, USA, 15-16 May 2005. Revised Selected Papers.

[11] Scott A. DeLoach and Madhukar Kumar. Multi-agent systems engineering: An overview and case study. In Henderson-Sellers and Giorgini [19], chapter XI, pages 317–340.

[12] Scott A. DeLoach and Jorge L. Valenzuela. An agent-environment interaction model. In Lin Padgham and Franco Zambonelli, editors, *Agent-Oriented Software Engineering VII*, volume 4405 of *LNCS*. Springer, 2007. 7th International Workshop (AOSE 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

[13] Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors. *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, The Netherlands, 25–29 July 2005. ACM Press.

[14] Virginia Dignum. *A Model for Organizational Interaction, based on Agents, founded in Logic*. PhD thesis, University of Utrecht, 2003.

[15] Marc Esteva, Bruno Rosell, Juan A. Rodríguez-Aguilar, and Josep Lluís Arcos. AMELI: An agent-based middleware for electronic institutions. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 236–243, New York, USA, 19–23 July 2004. IEEE Computer Society.

[16] FIPA Methodologies. Home page. `http://www.pa.icar.cnr.it/~cossentino/FIPAmeth/`.

[17] Francisco J. Garijo, Jorge J. Gòmez-Sanz, and Philippe Massonet. The MESSAGE methodoly for agent-oriented analysis and design. In Henderson-Sellers and Giorgini [19], chapter VIII, pages 203–235.

[18] Paolo Giorgini, Manuel Kolp, John Mylopoulos, and Jaelson Castro. Tropos: A requirements-driven methodology for agent-oriented software. In Henderson-Sellers and Giorgini [19], chapter II, pages 20–45.

[19] Brian Henderson-Sellers and Paolo Giorgini, editors. *Agent Oriented Methodologies*. Idea Group Publishing, Hershey, PA, USA, June 2005.

[20] INGENIAS. Home page. `http://grasia.fdi.ucm.es/ingenias/metamodel/`.

[21] JADE-board. `http://sharon.cselt.it/projects/jade/`, 2000.

[22] Thomas Juan, Adrian R. Pearce, and Leon Sterling. ROADMAP: extending the Gaia methodology for complex open systems. In Castelfranchi and Johnson [6], pages 3–10.

[23] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 3rd edition, December 2003.

[24] Marco Mamei and Franco Zambonelli. Programming stigmergic coordination with the TOTA middleware. In Dignum et al. [13], pages 415–422.

[25] 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.

[26] 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.

[27] 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.

[28] Andrea Omicini. Formal ReSpecT in the A&A perspective. In Carlos Canal and Mirko Viroli, editors, *5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06)*, pages 93–115, CONCUR 2006, Bonn, Germany, 31 August 2006. University of Málaga, Spain. Proceedings.

[29] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3):21–36, 29 May 2006. 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.

[30] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Coordination artifacts as first-class abstractions for MAS engineering: State of the research. In Alessandro F. Garcia, Ricardo Choren, Carlos Lucena, Paolo Giorgini, Tom Holvoet, and Alexander Romanovsky, editors, *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications*, volume 3914 of *LNAI*, pages 71–90. Springer, April 2006. Invited Paper.

[31] Andrea Omicini and Franco Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, September 1999.

[32] Lin Padgham and Michael Winikof. Prometheus: A methodology for developing intelligent agents. In Fausto Giunchiglia, James Odell, and Gerhard Weiss, editors, *Agent-Oriented Software Engineering III*, volume 2585 of *LNCS*, pages 174–185. Springer, 2003. 3rd International Workshop (AOSE 2002), Bologna, Italy, 15 July 2002. Revised Papers and Invited Contributions.

[33] Lin Padgham and Michael Winikoff. Prometheous: A practical agent oriented methodology. In Henderson-Sellers and Giorgini [19], chapter V, pages 107–135.

[34] Juan Pavòn, Jorge J. Gòmez-Sanz, and Rubén Fuentes. The INGENIAS methodology and tools. In Henderson-Sellers and Giorgini [19], chapter IX, pages 236–276.

[35] Gauthier Picard, Carole Bernon, and Marie-Pierre Gleizes. Cooperative agent model within ADELFE framework: An application to a timetabling problem. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 3, pages 1506–1507, New York, USA, 19–23 July 2004. ACM Press.

[36] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Programming MAS with artifacts. In Rafael P. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3862 of *LNAI*, pages 206–221. Springer, March 2006. 3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26 July 2005. Revised and Invited Papers.

[37] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO: A framework for prototyping artifact-based environments in MAS. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for MultiAgent Systems*, volume 4389 of *LNAI*, pages 67–86. Springer, February 2007. 3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

[38] John A. Sauter, Robert S. Matthews, H. Van Dyke Parunak, and Sven Brueckner. Performance of digital pheromones for swarming vehicle control. In Dignum et al. [13], pages 903–910.

[39] Olivier Simonin and Franck Gechter. An environment-based methodology to design reactive multi-agent systems for problem solving. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for Multi-Agent Systems II*, volume 3830 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2006. 2nd International Workshop (E4MAS 2005), Utrecht, The Netherlands, July 25, 2005, Selected Revised and Invited Papers.

[40] Arnon Sturm and Onn Shehory. A framework for evaluating agent-oriented methodologies. In Paolo Giorgini, Brian Henderson-Sellers, and Michael Winikoff, editors, *Agent-Oriented Information Systems*, volume 3030 of *LNCS*, pages 94–109. Spinger, 24 June 2004. 5th International Bi-Conference Workshop, AOIS 2003, Melbourne, Australia, July 14, 2003 and Chicago, USA, October 13, 2003, Revised Selected Papers.

[41] UML. Home page. `http://www.uml.org/`.

[42] Mirko Viroli, Tom Holvoet, Alessandro Ricci, Kurt Schelfthout, and Franco Zambonelli. Infrastructures for the environment of multiagent systems. In *Autonomous Agents and Multi-Agent Systems* [46], pages 49–60.

[43] Mirko Viroli, Andrea Omicini, and Alessandro Ricci. Engineering MAS environment with artifacts. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *2nd International Workshop "Environments for Multi-*

Agent Systems" (E4MAS 2005), pages 62–77, AAMAS 2005, Utrecht, The Netherlands, 26 July 2005.

[44] Mirko Viroli, Alessandro Ricci, and Andrea Omicini. Operating instructions for intelligent agent coordination. *The Knowledge Engineering Review*, 21(1):49–69, March 2006.

[45] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first-class abstraction in multi-agent systems. In *Autonomous Agents and Multi-Agent Systems* [46], pages 5–30.

[46] Danny Weyns and H. Van Dyke Parunak. Special issue on environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):1–116, February 2007.

[47] Danny Weyns, Kurt Schelfthout, Tom Holvoet, and Tom Lefever. Decentralized control of E'GV transportation systems. In Dignum et al. [13], pages 67–74.

[48] Danny Weyns, Giuseppe Vizzari, and Tom Holvet. Environments for situated multi-agent systems: Beyond infrastructure. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for Multi-Agent Systems II*, volume 3830 of *Lecture Notes in Computer Science*, pages 1–17. Springer, February 2006. Second International Workshop, E4MAS 2005, Utrecht, The Netherlands, July 25, 2005, Selected Revised and Invited Paper.

[49] Mark F. Wood and Scott A. DeLoach. An overview of the multiagent systems engineering methodology. In Paolo Ciancarini and Michael J. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 207–221. Springer-Verlag, 2001. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.

[50] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, September 2000.

[51] Franco Zambonelli, Nicholas Jennings, and Michael Wooldridge. Multiagent systems as computational organizations: the Gaia methodology. In Henderson-Sellers and Giorgini [19], chapter VI, pages 136–171.

[52] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, July 2003.