From AOSE Methodologies to MAS Infrastructures: The SODA Case Study

Ambra Molesini[1]*, Enrico Denti[1], and Andrea Omicini[2]

[1] Alma Mater Studiorum—Università di Bologna
viale Risorgimento 2, 40136 Bologna, Italy
ambra.molesini@unibo.it, enrico.denti@unibo.it
[2] Alma Mater Studiorum—Università di Bologna a Cesena
via Venezia 52, 47023 Cesena, Italy
andrea.omicini@unibo.it

Abstract Research on agent-oriented software engineering (AOSE) methodologies and multi-agent system (MAS) infrastructures has developed in the last years along two opposite paths: while AOSE methodologies have essentially undergone a top-down evolution pushed by contributions from heterogeneous fields like human sciences, MAS infrastructures have mostly followed a bottom-up path growing from existing and widespread (typically object-oriented) technologies. This dichotomy has produced a conceptual gap between the proposed AOSE methodologies and the agent infrastructures actually available, as well as a technical gap in the MAS engineering practice, where methodologies are often built *ad hoc* out of MAS infrastructures, languages and tools.
This paper proposes a new method for filling the gap between methodologies and infrastructures based on the definition and study of the meta-models of both AOSE methodologies and MAS infrastructures. By allowing structural representation of abstractions to be captured along with their mutual relations, meta-models make it possible to map design-time abstractions from AOSE methodologies upon run-time abstractions from MAS technologies, thus promoting a more coherent and effective practice in MAS engineering. In order to validate our method, we take an AOSE methodology, SODA, and show how it can be mapped upon three different MAS infrastructures using meta-models as mapping guidelines.

1 Introduction

Traditional software engineering (SE) approaches and metaphors fall short when applied to areas of growing relevance such as electronic commerce, enterprise resource planning, and mobile computing: such areas, in fact, generally call for open architectures that may evolve dynamically over time so as to accommodate new components and meet new requirements. This is probably one of the main reasons why the *agent* metaphor and the agent-oriented paradigm are gaining momentum in these areas. At the same time, such a rapid paradigm shift dropped

* Corresponding author.

technology behind: while in the past new abstractions used to come from programming languages, and were later included in software engineering practice, now it is often the case that technologies adopted for MAS (multi-agent system) development and deployment do not support the novel abstractions adopted in the AOSE (agent-oriented software engineering) analysis and design phases.

Such a gap mainly depends on AOSE methodologies and MAS infrastructures having evolved along two parallel, yet somehow inverse, paths: a top-down evolution for AOSE methodologies, a bottom-up path for multi-agent infrastructures. In fact, on the one side, abstractions and metaphors (models and structures) from human organisations have been used to analyse, model and design software systems, leading to methodologies like Gaia [1,2], Tropos [3,4], PASSI [5,6] and SODA [7,8]. There, modelling *agent societies* means to identify the global rules that should drive the expected MAS evolution, and the roles that agents should play. On the other side, MAS infrastructures have typically evolved out from existing (mainly, object-oriented) programming languages and development environments, "stretching" the agent paradigm on top of more traditional paradigms and technologies [9]. For instance, infrastructures such as TuCSoN [10,11], TOTA [12,13] and CArtAgO [14,15] introduce specific agent-oriented abstractions (tuple centres, co-fields, artifacts) to constructively constrain the design and final architecture of MAS: yet, the imprint of the object-oriented paradigm is still visible—for instance, in agents taking the form of Java threads. The above gap can lead to inconsistencies between the design and the implementation of a system, as the agent-based concepts and metaphors adopted in the analysis and design phases can not match the development tools used for system implementation and deployment, which are often in the stage of academic prototypes.

In this context, this paper is aimed at highlighting some guidelines for correctly mapping the abstractions adopted by an AOSE methodology onto the abstractions supported by MAS infrastructures: we assume SODA as a case study, and discuss how its design abstractions could be mapped onto three MAS infrastructures—TuCSoN, CArtAgO and TOTA. Accordingly, we first analyse and compare the meta-models of the SODA methodology and of the chosen infrastructures, then exploit them to express both the structural representation of the elements constituting the actual system, and their relationships [16].

Accordingly, the paper is structured as follows. Section 2 sketches a possible classification of AOSE methodologies based on their relations with MAS technologies, and highlights the main advantages of their meta-model representation. Then, Section 3 discusses the meta-models of the SODA methodology and of the selected infrastructures, whereas Section 4 presents the mapping of SODA abstractions onto such infrastructures and discusses the key guidelines. Related work, conclusions and future works are reported in Section 5 and Section 6.

2 AOSE Methodologies: Technologies & Meta-models

MAS are a powerful paradigm for the implementation of complex computational systems: the aim of AOSE is to effectively support the path from (an agent-

oriented) design to (an agent-based) deployment of the system. This is why methodologies (and respective notations) have become central in AOSE research as a key tool in the MAS analysis, design and development process.

Among the current methodologies, some are rooted in artificial intelligence (AI), others emerge as an extension of object-oriented (OO) methodologies, further try to merge the two approaches in some original way; finally, others are not directly derived by previous approaches. So, an important classification criterion is to distinguish methodologies that are neutral with respect to the implementation technologies (*technology-neutral methodologies*, or simply *neutral* ones in the following), from those that are bound to specific infrastructures (*technology-biased methodologies*, or simply *biased* ones in the following)—usually, due to the choice of developing a CASE tool for supporting rapid prototyping and code generation [17,18]. The first category includes methodologies like Gaia [1,2], MESSAGE [19,20], INGENIAS [21], and SODA [7,8]: they all aim at guiding the designer from the requirement analysis phase down to the design phase, yet with no assumptions on the implementation and deployment phases—probably, because of the lack of a recognised standard language and infrastructure that could natively support agent-oriented concepts. Among biased methodologies, Tropos [3,4] and PASSI [5,6] are both tied to the JADE [22,23] infrastructure, and come with a set of development support tools.

Since there is currently no widely-acknowledged standard infrastructure for MAS implementation, it is unclear whether committing to an infrastructure at methodological level may be better or worse than opting for technological neutrality. In fact, even though neutral methodologies suffer from a deeper gap with respect to the underlying technology, their models are general enough to be potentially implemented over any infrastructure by just providing suitable guidelines for mapping methodology abstractions onto infrastructure ones.

Apart from the technology neutrality matter, however, all AOSE methodologies introduce some basic abstractions (agents, roles, behaviour, ontology, ...) organised in a set of independent – but strongly correlated – models and phases. The relationships between such entities and the models can then be expressed by means of a *meta-model* [24,25], which becomes the key tool to compare methodologies with each other, identify families of (related) methodologies, and check the consistency of a methodology when planning extensions or modifications. So, a well-defined meta-model should address several different methodological aspects—for instance, the process to be followed, the work products to be generated, who is responsible for each process phase (analysts, designers, ...), etc.

Meta-models are also an important guide for integrating different methodologies avoiding several errors [24], such as assuming that differences of concern exist when none exists, or assuming similarity of concern because of a common use of terms—despite a different semantics. In the same way, infrastructure meta-models associate each methodology concept to some suitable infrastructural abstraction. So, we expect that studying and comparing methodologies with infrastructures in terms of meta-models makes it possible to provide guidelines for mapping the design model of a methodology onto its implementation.

3 Meta-models for the Case Study

In the following we first introduce and analyse the meta-models of of the SODA methodology (Subsection 3.1) and of the three selected infrastructures—TuCSoN (Subsection 3.2), CArtAgO (Subsection 3.3), and TOTA (Subsection 3.4).

Then (Section 4) we discuss the guidelines for mapping the SODA concepts and metaphors onto such infrastructures, based on their meta-models.

3.1 SODA

SODA (Societies in Open and Distributed Agent spaces) [7,8,26,27] is an agent-oriented methodology for the analysis and design of agent-based systems. Since the original version [7], SODA has always focused on inter-agent issues, like the engineering of agent societies and the environment for MAS: in this perspective, it has recently been reformulated according to the A&A meta-model [28,29,30], where artifacts take the form of computational devices that populate the agent environment, and provide some kind of function or service used by agents [28]. Agents are used to model individual activities, while artifacts shape the MAS environment [29]. More generally, artifacts make it easier to enrich the MAS design with social and organisational structures, as well as with complex security models: roles, permissions, policies, commitments, and the like can be represented explicitly as first-class entities, and encapsulated within suitable artifacts. Many sorts of artifacts are supported by SODA, even if in the meta-model we refer to them simply as "artifact" without specify their typology. In particular, artifacts used to mediate between individual agents and the MAS are called *individual artifacts*, whereas *social artifacts* build up agent societies, and *environmental artifacts* mediate between the MAS and an external resource [29].

SODA is organised in two phases, each structured in two sub-phases: the *Analysis phase*, which is composed of the Requirements Analysis and the Analysis steps, and the *Design phase*, which is composed of the Architectural Design and the Detailed Design steps. The meta-model that represents the abstract entities adopted by SODA is depicted in Figure 1.

Requirement Analysis. Several abstract entities are introduced for requirement modelling (see Figure 1 "requirement 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, "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 the definition of *dependencies* (interactions, constraints, etc.) among the abstract entities. The

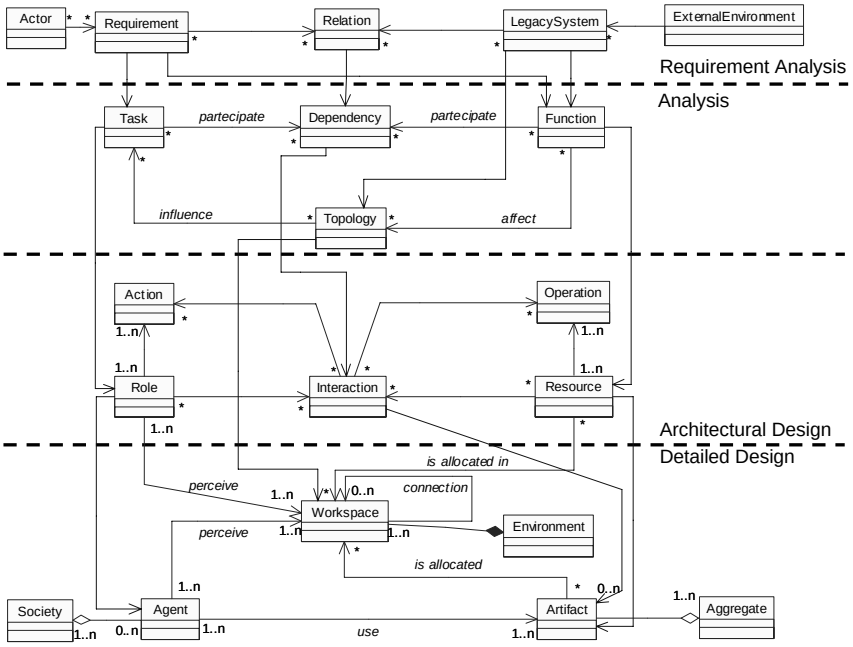

Figure 1. SODA Meta-model

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, "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 *artifact aggregates* (see Figure 1 "detailed design" part). Agents are intended here as autonomous entities able to play several roles, while societies are defined as the abstractions responsible for a collection of agents. The resources identified in the previous step are now mapped onto suitable artifacts (intended as entities providing some services), while aggregates are defined as the abstractions responsible for a collection of artifacts. 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, as well as agents can dynamically enter (join) or exit workspaces.

3.2 TuCSoN

TuCSoN (Tuple Centres Spread Over Networks) [10,11] is an infrastructure providing services for the communication and coordination of distributed / concurrent independent agents: its meta-model is depicted in Figure 2.

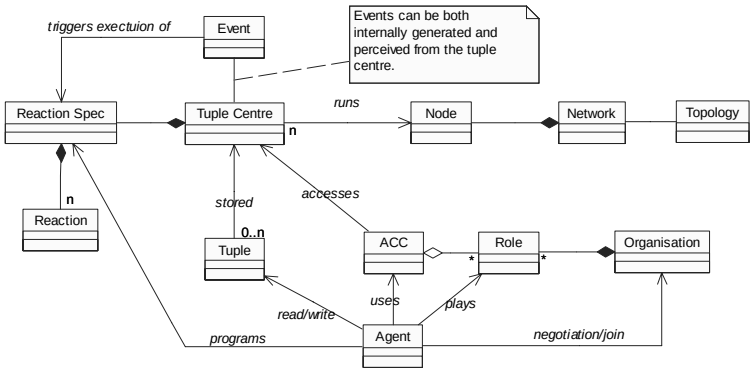

Figure 2. TuCSoN Meta-model

In detail, TuCSoN supports agent communication and coordination via *tuple centres* coordination media [31]: these are shared & reactive information spaces, distributed over the infrastructure *nodes*. In turn, this inducts a *topology* over the *network*. Agents access tuple centres associatively, by writing (`out`), reading (`rd`, `rdp`), and consuming (`in`, `inp`) *tuples* – i.e., ordered collections of heterogeneous information chunks – via the above coordination primitives.

A tuple centre is a tuple space enhanced with the notion of behaviour specification. More precisely, a tuple centre is a coordination abstraction perceived by the interacting entities as a standard tuple space [32], but whose behaviour in response to events can be defined so as to embed the coordination laws. So, defining a new behaviour for a tuple centre basically amounts at specifying state transitions in terms of *reactions* to *events* [10]. In particular, reactions are specified in TuCSoN via the ReSpecT (Reaction Specification Language) language [30]: a reaction is defined as a set of non-blocking operations [10], and has a success/failure transactional semantics: a successful reaction may atomically produce effects on the tuple centre state, a failed reaction yields no result at all. Typically, a tuple centre contains a set of reactions (*reaction spec* in Figure 2), each tied to a specific event: the same event could trigger multiple, different reactions. Tuple centres are connected each other through *link* operations, having the same form and a similar semantics as TuCSoN coordination primitives, but invoked by successful reactions rather than by agents [30].

The Agent Coordination Context (ACC), introduced in [33] as the conceptual place where to set the boundary between the agent and the environment,

encapsulates the interface enabling agent actions and perceptions inside the environment. More precisely, an ACC (*i*) works as a model for the agent environment, by describing the environment where an agent can interact, and (*ii*) enables and rules the interactions between the agent and the environment, by defining the space of the admissible agent interactions. The ACC dynamics is characterised by two basic steps: *negotiation* and *use*. In fact, an ACC is meant to be first negotiated by the agents with the MAS infrastructure, in order to start a working session inside an *organisation*. To this end, the agent specifies which *roles* to activate: if the agent request is compatible with the (current) organisation rules, a new ACC is created, configured according to the characteristics of the specified roles, and is released to the agent for active playing inside the organisation. The agent can then use the ACC to interact with other agents in the organisation, and with the organisation environment, by performing the actions and activating the perceptions made possible by the ACC.

3.3 CArtAgO

The abstract architecture of CArtAgO (Common Artifact for Agents Open environment) [14,15] is composed of three main elements (see Figure 3): (*i*) *agent bodies* – as the entities that make it possible to situate agents inside the working environment; (*ii*) *artifacts* – as the basic building blocks to structure the working environment; and (*iii*) *workspaces* – as the logical containers of artifacts, aimed at defining the topology of the working environment.

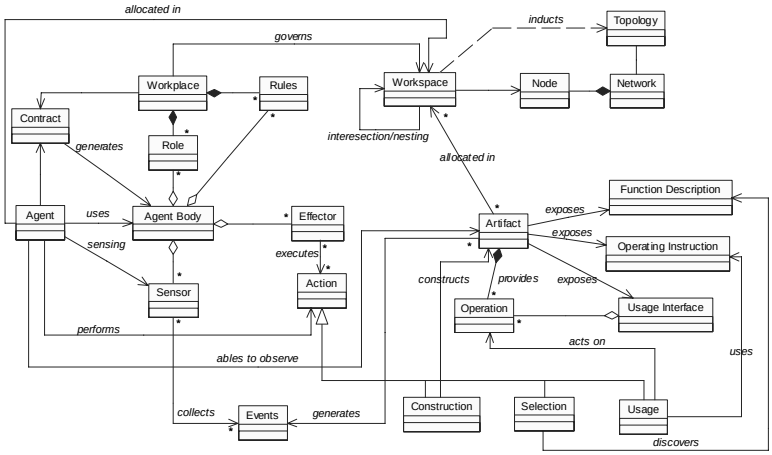

Figure 3. CArtAgO Meta-model

Agent bodies. The agent body contains *effectors* to perform *actions* upon the working environment, and a dynamic set of *sensors* to collect *events* from the working environment. Agents interact with their working environment by

"piloting" their bodies: they execute actions to *construct*, *select* and *use* artifacts, and perceive the *observable events* generated by artifacts.

Artifacts. Artifacts are the basic bricks managed by CArtAgO: agents use artifacts by triggering the execution of *operations* listed in the artifact *usage interface*. The execution of an operation typically causes the update of the internal state of an artifact, and the generation of one or more observable events: these are then collected by the agent sensors and perceived by means of explicit sensing actions. In order to support a rational exploitation of artifacts by intelligent agents, each artifact is equipped with a *function description*, i.e. an explicit description of the functionalities it provides, and *operating instructions*, i.e. an explicit description of how to use the artifact to get its function.

Workspaces. Artifacts are logically located in workspaces, which define the *topology* of the working environment. A workspace is an open set of artifacts and agents: artifacts can be dynamically added to or removed from workspaces by agents, agents can dynamically enter (join) or exit workspaces. Workspaces make it possible to structure agents and artifacts organisation & interaction: in particular, workspaces can function as scopes for event generation and perception, as well as for artifact access and use. Articulated topologies can be created via workspace intersection and nesting: in particular, intersection is supported by allowing the same artifacts and agents to belong to different workspaces.

In addition, CArtAgO also introduces the concept of *workplace* as an organisational layer on top of workspaces. More precisely, a workplace is the set of *roles* and *organisational rules* being in force in a workspace: there, *contracts* define the norms and policies that rule agent access to artifacts and allow the generation of agent bodies. So, for instance, an agent may or may not be granted permission to use some artifacts or to execute some specific operations on selected artifacts depending on the role(s) that the agent is playing inside the workplace [34].

3.4 TOTA

TOTA (Tuples On The Air)[12,13] is a middleware for multi-agent coordination, in distributed computing scenarios. A meta-model of the infrastructure is presented in Figure 4. TOTA assumes the presence of a network of possibly mobile *nodes*, each running a *tuple space* [31]: each agent is supported by a local middleware and has only a local (one-hop) perception of its environment. Nodes are connected only by short-range *network* links, each holding references to a (limited) set of *neighbour* nodes: so, the *topology* of the network, as determined by the neighbourhood relations, may be highly dynamic.

In TOTA, tuples are not associated to a specific node (or to a specific data space) of the network: rather, they are "injected" in the network by an agent from some node, then autonomously propagate hop-by-hop, diffuse, and evolve according to specified propagation patterns. Thus, TOTA tuples form a sort of spatially-distributed data structure, that can be used to acquire contextual information about the environment and to support the mechanisms required for stigmergic interaction [35]. More precisely, TOTA distributed tuples T=(C,P,M)

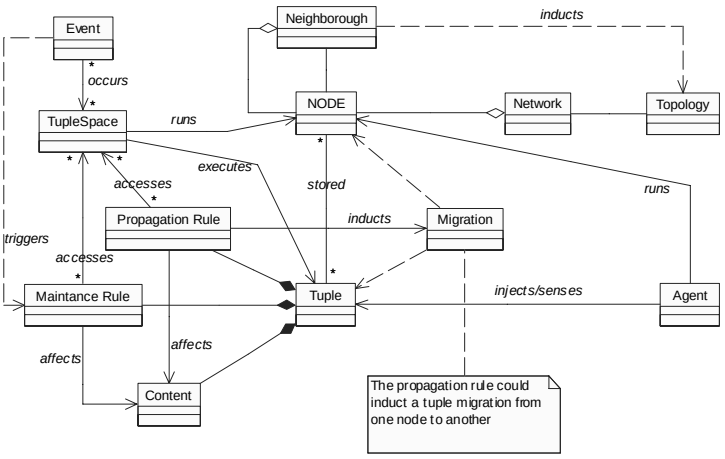

Figure 4. TOTA Meta-model

are characterised by a *content* C, a *propagation rule* P, and a *maintenance rule* M: the content C is an ordered set of typed fields representing the information carried by the tuple, the propagation rule P determines how the tuple propagates across the network (called "migration" in the Figure 4) and how the tuple content should change while the tuple is propagated; finally, the maintenance rule M determines how a tuple distributed structure should react to *events* occurring in the environment. Specifying the tuple propagation rule includes determining the "scope" of the tuple and how such propagation is affected by the presence or absence of other tuples in the system. In turn, events handled by the maintenance rule can range from simple time alarms, to changes in the network structure: the latter kind of events is of fundamental importance to preserve a coherent structure of the environment properties represented by tuple fields.

4 From SODA to Infrastructures

This section presents some guidelines for mapping SODA design-level abstractions onto the infrastructural abstractions of TuCSoN (Subsection 4.1), CArtAgO (Subsection 4.2) and TOTA (Subsection 4.3): the abstractions used in SODA analysis phase are left aside, as they would be too high-level with respect to infrastructures. Among the many MAS infrastructures available in literature, we choose TuCSoN and TOTA because interaction – and coordination, in particular – is at the core of both infrastructures, in the same way as in SODA. In addition, both infrastructures are not FIPA-compliant, and here we meant to explore such a sort of infrastructures. Finally we choose CArtAgO because it is the only infrastructure that natively supports the concepts of artifacts.

Such infrastructures are then compared so as to evaluate their support to SODA abstractions (Subsection 4.4).

4.1 TuCSoN & SODA

Since SODA is defined on top of the A&A meta-model, the first step is to define how agents and artifacts can be represented as TuCSoN abstractions.

Mapping the agent notion is straightforward, given TuCSoN native support for this concept: so there is a one-to-one mapping between SODA agent abstraction and TuCSoN one. However, the concept of agent action, explicitly considered in the SODA meta-model, is more or less reduced to the notion of coordination primitives, as performed by agents.

Mapping the artifact notion, instead, is less obvious, as SODA defines three different artifact types – social, individual and environmental artifacts – each requiring its own mapping. With respect to this issue, TuCSoN tuple centres can be seen as a special case of social artifacts: they mediate and govern agent interaction by encapsulating the laws of agent coordination. Such coordination laws, expressed in terms of reactions to interaction events, are well suited to map SODA interactions – i.e., the rules that enable and bound the entities' behaviour. In turn, the notion of individual artifact can be mapped onto the TuCSoN ACC concept, since its purpose is precisely to represent the interface of an agent towards the environment [36]. In fact, agents ask for an ACC specifying the roles to be activated: the ACC is then negotiated with the infrastructure as the agent joins the MAS organisation. If the negotiation is successful, the ACC is created and released to the agent, which, henceforth, exploits it to access the MAS services: the ACC redirects the agent invocations to the other artifacts in the environment. Finally, the environmental artifact is not natively supported by TuCSoN, so it must be developed if/when needed. Also, the notion of artifact operation is reduced here to the notion of tuple centre operation, and has not the generality required. Given that, link operations through tuple centres are the way in which TuCSoN "artifacts" are somehow composed.

Widening the view, the organisation concept provided by TuCSoN is well suited to represent SODA societies, in the same way as the TuCSoN role concept can well represent the SODA role notion. From the topological viewpoint, the SODA notion of workspace may be mapped onto the TuCSoN node concept, which, indeed, represents an open set of agents, tuple centres and ACCs; as a consequent step, TuCSoN network can be used to map SODA environments. On the other hand, workspace connection, as introduced by SODA, has no mapping in TuCSoN, so it should be developed ad hoc when needed.

4.2 CArtAgO & SODA

CArtAgO and SODA share the same root in the A&A meta-model: so, quite expectedly, CArtAgO abstractions can easily support all SODA concepts. In particular, CArtAgO provides the artifact notion as a first-class abstraction, which can be used and easily specialised in social and environmental artifacts according to the developer's needs. Therefore, unlike TuCSoN, the SODA notion of artifact operation is directly mapped onto the operation abstractions supported by CArtAgO; the same holds for SODA agent action. Moreover, individual artifacts

can be more specifically mapped on CArtAgO agent body abstraction, instead of using the generic artifact notion.

Composition of artifacts can also be easily realised, thanks to the *linkability* property [29] natively supported by CArtAgO artifacts to scale up with environment complexity. So, an artifact can be conceived and implemented as a composition of linked, possibly non-distributed, artifacts – or, conversely, a set of linked artifacts, scattered through a number of different physical locations, can be seen altogether as a single distributed artifact.

In addition, SODA organisational structure, which is defined in terms of roles and societies, can be easily translated on CArtAgO roles and workplaces. This makes it possible to capture SODA interaction concept in a straightforward way: in fact, interactions in SODA are aimed at enabling and constraining agent behaviour, which is precisely what the workplace rules and contract do – a CArtAgO agent may or may not have the permission to use some artifacts or to execute some specific operations on some specific artifacts depending on the role(s) that the agent itself is playing inside the workplace.

Finally, CArtAgO workspace concept can be directly used to map SODA workspace concept, in the same way as CArtAgO workspace nesting supports SODA workspace connection. The CArtAgO abstractions of node, network and topology can be used to represent the SODA environment, too.

4.3 TOTA & SODA

TOTA provides a native support to the agent concept, while the artifact concept is supported only in the case of social artifacts. So, SODA agents can be directly mapped onto TOTA agents, while social artifacts are mapped onto TOTA tuple spaces. Unlike tuple centres, tuple spaces provide only a fixed coordination service: so, they are unable to support the SODA interaction concept. However, SODA social rules can be mapped onto the maintenance rule and the propagation rule associated to TOTA distributed tuples, exploiting the fact that propagation rules determine how tuples propagate through the network, and maintenance rules determine how the tuple distributed structure reacts to environment events. Of course, this mapping is less straightforward than in TuCSoN (whose reactions map SODA interaction concept directly): indeed, a set of many tuples must be used to describe a single SODA interaction – each tuple representing one propagation and one maintenance rule. As a side effect of this one-to-many mapping, maintaining coherency is quite a hard task, and the rules/interaction mapping can often be very dispersive.

From the topology viewpoint, TOTA node concept maps SODA workspace concept: each node holds references to a limited set of neighbour nodes, and neighbourhood relations express the network topology. Such inter-node relations can be exploited also to provide an abstraction for mapping the SODA workspace connection concept. Finally, the TOTA network concept maps the SODA environment, too. All the others SODA concept are not natively supported by TOTA, and should therefore be developed in an *ad hoc* way when needed.

4.4 Discussion

Table 5 highlights the SODA abstractions that are supported natively from each of the three infrastructures. The agent and resource abstractions are both omitted – the first because it is explicitly supported by each infrastructure, the latter for the opposite reason.

Quite expectedly, CArtAgO provides the best support for SODA design abstractions, as they are both rooted in the A&A meta-model: in particular, both consider the environment as the key element, adopt artifacts as their basic building blocks for modelling the environment resources, and workspaces for structuring the environment. Moreover, both SODA and CArtAgO support the MAS organisational structure by explicitly enabling the specification of social rules.

TuCSoN and TOTA, instead, provide support for fewer SODA abstractions: so, the developer needs to implement by himself the abstractions which are not supported by the infrastructure natively. In particular, none of the two infrastructures supports environmental artifacts, while both support social artifacts: this is not surprising, since they take both inspiration from coordination models, where interaction is typically mediated by some coordination media [31] (like a tuple space or a tuple centre) that could be easily seen as a special case of social artifact. Individual artifacts, in their turn, find their counterpart only in TuCSoN – namely, in the ACC abstraction. Moreover, SODA interaction abstraction, which represents the rules that enable and shape the agent behaviour, can be expressed directly by TuCSoN reactions, and indirectly via TOTA maintenance and propagation rules. Finally, as far as the organisational structure of the MAS is concerned, TuCSoN provides explicit abstractions such as organisation, role and ACC; on the other hand, TOTA does not provide any support for this issue yet, so the developer must provide for managing the MAS organisations on his/her own.

SODA	TuCSoN	CArtAgO	TOTA
Role	*Role*	*Role*	-
Action	*Coordination Primitive*	*Action*	-
Interaction	*Reaction Specification Reaction*	*Rules Contract*	*Maintenance Rule Propagation Rule*
Operation	*Tuple Centre Operation*	*Operation*	-
(Social) Artifact	*Tuple Centre*	*Artifact*	*Tuple Space, Tuples*
(Individual) Artifact	*ACC*	*Agent Body*	-
(Environmental) Artifact	-	*Artifact*	-
Aggregate	*Linked Tuple Centres*	*Artifact*	-
Society	*Organisation*	*Workplace*	-
Workspace	*Node*	*Workspace*	*Node*
Workspace Connection	-	*Workspace Nesting*	*Neighborough*

Figure 5. Abstractions Mapping

5 Related Work

Model-Driven Architecture [37] (MDA) is another approach for filling the gap among methodologies and infrastructures: its basic idea is to define first a Platform Independent Model (PIM) and then iteratively make it more and more platform-specific by a series of transformations, whose endpoint is the Platform Specific Model (PSM). Current technologies, however, may not fully support MDA complex transformation rules: for instance, UML, which is one of MDA foundations, lacks the required precision and formalisation [38].

Further research efforts are being devoted to integrating MDA and AOSE [38,39]. In [38], for instance, an agent architecture based on the human cognitive model of planning, the Cognitive Agent Architecture (Cougaar), is integrated with MDA. The resulting Cougaar MDA defines the models to be used, how they should be prepared, and the relationships among them. The level of application composition is thus elevated from individual components to domain-level model specifications in order to generate software artifacts. The software artifacts generation is based on a meta-model: each component is mapped onto a UML structured component which is then converted into multiple artifacts—Cougaar/Java code, documentation, and test cases. In [39], Amor et al. show how the Model Driven Architecture (MDA) can be used to derive agent implementations from agent-oriented designs, independently from both the methodology and the concrete agent platform. Their goal is to study how to bridge the gap between methodologies and infrastructures, so as to cover the whole MAS lifecycle. Authors show how this problem can be naturally expressed in terms of MDA, and how MDA mechanisms can be used for defining the mappings. By applying the MDA ideas, the design model obtained from an agent-oriented methodology can be considered as a PIM, the target MAS agent platform as the PSM, and the mappings between the two can be given by the transformations defined for the selected agent platform. The target models need to be expressed in terms of their corresponding UML profiles, as indicated by MDA.

Since both methods imply the use of UML, its practical application requires that the selected AOSE methodology adopts UML or AUML as its notation: if this is not the case, like for many AOSE methodologies, an additional transformation from the methodology own notation to UML is necessary. As a result, the overall application of this approach involves many transformations for each mapping, and requires a PSM to be defined for each infrastructure.

6 Conclusions and Future Work

In this paper we adopted the SODA methodology as a running example for mapping the methodological concepts onto infrastructural abstractions in the case of three main agent infrastructures—TuCSoN, CArtAgO, and TOTA. To this end, we first studied the agent-oriented methodologies from the point of view of the connection with the implementation technologies, and classified them into *technology-neutral* and *technology-biased* methodologies.

Starting from neutral methodologies, that currently seem more appealing because of their independence from the underlying non-standard technologies, we then exploited meta-modelling as a tool to formalise the inner structure and the composing relationships both for the methodology and the selected infrastructures. Accordingly, we developed and comparatively analysed the meta-models of the SODA methodology and of CArtAgO, TuCSoN, and TOTA infrastructures, with double purpose of *(a)* providing guidelines for bridging the design and implementation phases, and *(b)* evaluating the quality of the mapping of SODA concepts onto infrastructural abstractions in terms of naturalness, clearness, and directness of the mapping.

Of course, this research is still in its early stage, so a lot of work remains to do: the next steps will be devoted to develop meta-models for other MAS infrastructures such as MARS [40], RETSINA [41] and JADE [22,23], and to study how to map SODA concepts onto these infrastructures. In order to test our method, we also plan to make the same experiments taking as a base another neutral methodology, among Gaia [2], MESSAGE [19] or INGENIAS [21].

7 Acknowledgements

This work has been supported by the *MEnSA* project (*Methodologies for the Engineering of complex software Systems: Agent-based approach*) funded by the Italian Ministry of University and Research (MIUR) in the context of the National Research 'PRIN 2006' call.

References

1. Zambonelli, F., Jennings, N., Wooldridge, M.: Multiagent systems as computational organizations: the Gaia methodology. [42] chapter VI 136–171
2. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) **12** (2003) 317–370
3. Giorgini, P., Kolp, M., Mylopoulos, J., Castro, J.: Tropos: A requirements-driven methodology for agent-oriented software. [42] chapter II 20–45
4. Tropos: Home page. (`http://www.troposproject.org/`)
5. Cossentino, M.: From requirements to code with the PASSI methodology. [42] chapter IV 79–106
6. Cossentino, M., Sabatucci, L., Chella, A.: Patterns reuse in the PASSI methodology. In Omicini, A., Petta, P., Pitt, J., eds.: Engineering Societies in the Agents World IV. Volume 3071 of LNAI., Springer-Verlag (2004) 294–310 4th Inter. Workshop (ESAW 2003), London, UK, 29–31 October2003.
7. Omicini, A.: SODA: Societies and infrastructures in the analysis and design of agent-based systems. In Ciancarini, P., Wooldridge, M.J., eds.: Agent-Oriented Software Engineering. Volume 1957 of LNCS. Springer (2001) 185–193 1st Inter. Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.
8. SODA: Home page. (`http://soda.alice.unibo.it`)

9. Omicini, A., Rimassa, G.: Towards seamless agent middleware. In: IEEE 13th Inter. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2004), 2nd Inter. Workshop "Theory and Practice of Open Computational Systems" (TAPOCS 2004), Modena, Italy, IEEE CS (2004) 417–422
10. Omicini, A., Zambonelli, F.: Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems **2** (1999) 251–269
11. TuCSoN: Home page at SourceForge. (`http://tucson.sourceforge.net`)
12. Mamei, M., Zambonelli, F.: Programming stigmergic coordination with the TOTA middleware. In Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M., eds.: Proceedings of AAMAS 2005, ACM Press (2005) 415–422
13. Mamei, M., Zambonelli, F.: Programming modular robots with the tota middleware. In Nakashima, H., Wellman, M.P., Weiss, G., Stone, P., eds.: 5th Inter. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), ACM (2006) 485–487
14. Ricci, A., Viroli, M., Omicini, A.: CArtAgO: A framework for prototyping artifact-based environments in MAS. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: Environments for MultiAgent Systems. Volume 4389 of LNAI. Springer (2007) 67–86 3rd Inter. Workshop (E4MAS 2006). Selected Revised and Invited Paper.
15. CArtAgO: Home page. (`http://www.alice.unibo.it:16080/projects/cartago/`)
16. Molesini, A., Denti, E., Omicini, A.: MAS meta-models on test: UML vs. OPM in the SODA case study. In Pěchouček, M., Petta, P., Varga, L.Z., eds.: Multi-Agent Systems and Applications IV. Volume 3690 of LNAI., Springer (2005) 163–172 4th Inter. Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05), Budapest, Hungary, 15–17 September 2005, Proceedings.
17. PASSI: Toolkit web page. (`http://sourceforge.net/projects/ptk`)
18. TAOM4E: Home page. (`http://sra.itc.it/tools/taom4e/`)
19. Garijo, F.J., Gòmez-Sanz, J.J., Massonet, P.: The MESSAGE methodoly for agent-oriented analysis and design. [42] chapter VIII 203–235
20. Caire, G., Coulier, W., Garijo, F.J., Gomez, J., Pavòn, J., Leal, F., Chainho, P., Kearney, P.E., Stark, J., Evans, R., Massonet, P.: Agent oriented analysis using Message/UML. In Wooldridge, M., Weiss, G., Ciancarini, P., eds.: Agent-Oriented Software Engineering II. Volume 2222 of LNCS. Springer (2002) 119–135
21. Pavòn, J., Gòmez-Sanz, J.J., Fuentes, R.: The INGENIAS methodology and tools. [42] chapter IX 236–276
22. JADE: Home page. `http://sharon.cselt.it/projects/jade/` (2000)
23. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a fipa-compliant agent framework. Softw., Pract. Exper. **31** (2001) 103–128
24. Bernon, C., Cossentino, M., Gleizes, M.P., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. In Odell, J., Giorgini, P., Müller, J.P., eds.: Agent-Oriented Software Engineering V. Volume 3382 of LNCS., Springer (2004) 62–77 5th International Workshop, AOSE 2004. Revised Selected Papers.
25. van Hillegersberg, J., Kumar, K., Welke, R.J.: Using metamodeling to analyze the fit of object-oriented methods to languages. In: 31st Hawaii Inter. Conference on System Sciences (HICSS 1998). Volume 5:Modeling Technologies and Intelligent Systems., Kohala Coast, HI, USA, IEEE Computer Society (1998) 323–332
26. Molesini, A., Omicini, A., Ricci, A., Denti, E.: Zooming multi-agent systems. In Müller, J.P., Zambonelli, F., eds.: Agent-Oriented Software Engineering VI. Volume 3950 of LNCS. Springer (2006) 81–93 6th Inter. Workshop (AOSE 2005), Utrecht, The Netherlands, 25–26 July 2005. Revised and Invited Papers.

27. Molesini, A., Omicini, A., Denti, E., Ricci, A.: SODA: A roadmap to artefacts. In Dikenelli, O., Gleizes, M.P., Ricci, A., eds.: Engineering Societies in the Agents World VI. Volume 3963 of LNAI. Springer (2006) 49–62 6th Inter. Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 October 2005. Revised Paper.
28. Omicini, A., Ricci, A., Viroli, M.: Coordination artifacts as first-class abstractions for MAS engineering: State of the research. In Garcia, A.F., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A., eds.: Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications. Volume 3914 of LNAI. Springer (2006) 71–90 Invited Paper.
29. Omicini, A., Ricci, A., Viroli, M.: *Agens Faber*: Toward a theory of artefacts for MAS. Electronic Notes in Theoretical Computer Sciences **150** (2006) 21–36 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.
30. Omicini, A.: Formal ReSpecT in the A&A perspective. Electronic Notes in Theoretical Computer Sciences **175** (2007) 97–117 5th Inter. Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006. Post-proceedings.
31. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems **7** (1985) 80–112
32. Gelernter, D., Carriero, N.: Coordination languages and their significance. Communications of the ACM **35** (1992) 97–107
33. Omicini, A.: Towards a notion of agent coordination context. In Marinescu, D.C., Lee, C., eds.: Process Coordination and Ubiquitous Computing. CRC Press, Boca Raton, FL, USA (2002) 187–200
34. Ricci, A., Viroli, M., Omicini, A.: CArtAgO: An infrastructure for engineering computational environments in MAS. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: 3rd Inter. Workshop "Environments for Multi-Agent Systems" (E4MAS 2006). (2006) 102–119
35. Parunak, H.V.D.: "go to the ant": Engineering principles from natural agent systems. Annals of Operation Research **75** (1997) 69–101
36. Viroli, M., Omicini, A., Ricci, A.: Engineering MAS environment with artifacts. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: 2nd Inter. Workshop "Environments for Multi-Agent Systems" (E4MAS 2005), AAMAS 2005, Utrecht, NL (2005) 62–77
37. OMG: Home page. (`www.omg.org/mda/`)
38. Gracanin, D., Singh, H.L., Bohner, S.A., Hinchey, M.G.: Model-driven architecture for agent-based systems. In Hinchey, M.G., Rash, J.L., Truszkowski, W., Rouff, C., eds.: Formal Approaches to Agent-Based Systems. Volume 3228 of LNCS., Springer (2005) 249–261 3rd Inter. Workshop, FAABS 2004. Revised Selected Papers.
39. Amor, M., Fuentes, L., Vallecillo, A.: Bridging the gap between agent-oriented design and implementation using mda. In Odell, J., Giorgini, P., Müller, J.P., eds.: Agent-Oriented Software Engineering V. Volume 3382 of LNCS., Springer (2004) 93–108 5th Inter. Workshop, AOSE 2004. Revised Selected Paper.
40. Cabri, G., Leonardi, L., Zambonelli, F.: MARS: A programmable coordination architecture for mobile agents. IEEE Internet Computing **4** (2000) 26–35
41. Sycara, K.P., Paolucci, M., Velsen, M.V., Giampapa, J.A.: The RETSINA MAS infrastructure. Autonomous Agents and Multi-Agent Systems **7** (2003) 29–48
42. Henderson-Sellers, B., Giorgini, P., eds.: Agent Oriented Methodologies. Idea Group Publishing, Hershey, PA, USA (2005)