Towards organizational interoperability through artifacts

Fabio T. Muramatsu, Tomas M. Vitorello, Anarosa A. F. Brandão

Computing Engineering and Digital Systems Department
University of Sao Paulo – Brazil
fabio.muramatsu@usp.br, tomas.vitorello@usp.br
anarosa.brandao@usp.br

Abstract. Achieving interoperability among open and distributed systems is an issue addressed by several research communities, such as the ones related to service-oriented, cloud and bigdata computing, among others, mostly by proposing standards and protocols. The multiagent systems (MAS) research community is also interested in it, since Organization-Centered MAS (OC-MAS) are suitable for developing open systems for distributed and heterogeneous environments. Nevertheless, OC-MAS are still dependent on organizational infrastructures to execute properly, which means that to interoperate with several OC-MAS, agents must be able to run on different organizational infrastructures. This is still an issue under investigation in the MAS community. In this paper, we describe a first step to provide an artifact-based solution to achieve interoperability among OC-MASs by using an existing normative programming language to translate organizational models and following the multiagent programming approach from JaCaMo platform.

1 Introduction

The adoption of an organization-centered approach to develop multiagent systems (MAS) is based on the definition of a set of constraints that a group of agents adopts to achieve their social purposes easily [16]. This set of constraints is usually described as organizational models, such as MOISE+ [14, 12], AGR [7] and Opera [6], among others. By having an underlying organization model, the MAS may assure some level of efficiency and efficacy [8] *apud* [12], since the model establishes ways to coordinate and control the agents' behavior. This characteristic makes such approach suitable for developing open systems for distributed and heterogeneous environments. In fact, by this very nature, any agent that agrees to adopt the organization constraints is allowed to enter it. These systems are called Organization-centered MAS (OC-MAS).

The implementation of OC-MAS uses to be dependent on organizational infrastructures (OI) tailored to the OC-MAS underlying model. For instance, S-MOISE+ [15] and MADKit [9] are OI for MOISE+ and AGR, respectively. Therefore, in order to enter an open system developed as an OC-MAS, an agent should run on its associate OI. In this scenario, to interoperate with several OC-MAS, agents must be able to run on different OI, which is still a problem under investigation. Considering that OC-MAS are suitable for developing open systems for distributed and heterogeneous

adfa, p. 1, 2011.

environments such as the ones which support Ambient Intelligence, Smart Cities and Internet of Things, addressing such problem may accelerate their adoption in such a scale. At the best of our knowledge, Coutinho et al [5] is the only work that addresses such problem by providing a model-driven approach to define an integrated organizational model that can be mapped to existing ones and used as a common model to provide interoperability among different organizational models.

Recently Hübner and colleagues proposed ORA4MAS [10], an approach that transfers to the environment the responsibility of the OI, implementing it through the use of artifacts [17]. In such approach, agents must be able to interact with organizational artifacts in order to decide about the adoption of the organization constraints. An implementation of that was given considering MOISE+ and CArtAgO [18], which is an environment-oriented framework to support the development of MAS based on artifacts. Moreover, CArtAgO was integrated with several agent-programming platforms [19], including Jason [2].

Considering that CArtAgO and MOISE+ had already been integrated with Jason [19][13], respectively, Boissier et al recently propose JaCaMo [1], a platform that integrates Jason, CArtAgO and MOISE+ by combining agent-oriented, environment-oriented and organization-oriented programming issues. Also, JaCaMo adopts a normative programming language (NPL) [11] to describe the organization constraints related to coordination and control originally present in Moise+, through the inclusion of a normative engine in organizational artifacts. Nevertheless, the organizational artifacts implementation relies on the NPL interpreter instead of the OI itself, as shown in **Fig. 1**.

In this paper, we investigate if the NPL adopted by JaCaMo could be used to describe other organization models as a first step to address interoperability among organizational models using an environment-oriented approach. This is so because the organizational artifacts responsible for coordinating and controlling the organization constraints just need a translation of the organization specification to the NPL to run.

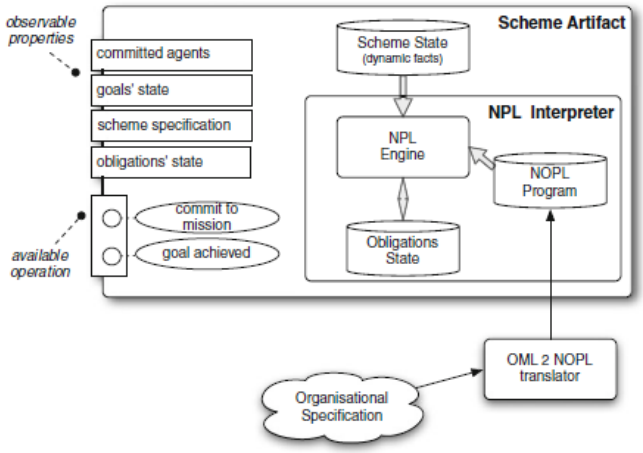

Fig. 1. General view of the organizational scheme artifact in Moise+ (source [11])

The paper is organized as follows: section 2 presents the organizational models we refer on the paper and section 3 describes the background knowledge for understanding the paper idea. Section 4 shows the translation of two organizational models to the NPL. Section 5 discusses about possible modification in existing artifacts followed by the conclusion and further work suggestions on the issue in section 6.

2 Organizational Models

2.1 MOISE+

MOISE+ is a model to support organization-centered MAS [12] modeling while providing a conceptual framework and syntax for designing their underlying organization specifications. It distinguishes three organizational dimensions to explain how a MAS organization can be designed: the Structural Specification, the Functional Specification and the Deontic Specification. The Structural Specification addresses the static aspects of an organization and involves roles, links and groups as main concepts. These concepts are used to describe the individual, the social and the collective levels of an organization, respectively. The individual level is formed by the organization's roles while the social level specifies the links (relations between roles) defined in order to constrain the agent action after accepting to play a role. Possible links are authority, communication and acquaintance. Finally, the collective level specifies how different roles can take part in groups.

The Functional Specification describes how organizational goals should be achieved, stating how these goals are decomposed in plans and distributed to the agents through missions. Finally, the Deontic Specification is responsible for linking the Functional and Structural Specifications by relating roles and missions through permissions and obligations.

A comparative analysis based on six organizational models was performed in [4], aiming at identifying which modeling dimensions were present in each of them. In the case of MOISE+, three dimensions were identified, the functional, structural and normative ones, in conformance to the aforementioned specifications.

2.2 AGR

The AGR (Agent/Group/Role) model [7] was developed aiming at providing a simple and concise way to describe multiagent organizations. As its acronym suggests, it is based on three primitives: agents, groups and roles. Agents are proactive, autonomous and communicative entities that populate these organizations by assuming certain roles within groups. Groups can be defined as sets of agents that have some similarities. Lastly, roles are abstract representations of agent functionalities in a group.

In this model, a group is described by a group structure. Such structure contains all the characteristics that define a group, for example its name and the roles agents are allowed to play in it. Thus, it is possible to conceive groups as instances of group structures.

In AGR, two types of constraints between roles are defined, namely correspondence and dependence. If there is a correspondence constraint between two roles A and B, it means that an agent who plays A will automatically play B. In its turn, a dependence constraint between A and B means that playing A is a prerequisite to play B.

Finally, it is stated in the description of this model that it is possible to define interaction relationships between roles, in order to constrain the communication performed by agents. However, no detailed explanation about the nature of these interactions is provided, leaving it as an open aspect for implementation.

In the case of AGR, two dimensions were identified in the analysis presented in [4]. First, the notions of groups and roles compound the structural dimension. Furthermore, the definition of interactions between agents forms the dialogical dimension.

2.3 OPERA

OperA is a model devised to describe open MAS using formal logical semantics and aims to ensure interaction and collaboration among its members while maintaining autonomy between society design and agent design [6]. The structure of the model is split into three separate sub models: the Organizational Model, the Interaction Model and the Social Model.

The Organizational Model is a description of the system itself, that is to say, it describes how the system is organized. This description is further segmented into four components: social, communication, interaction and normative structures. The Social Structure specifies the existing roles, as well as the goals and relations associated with them. The Communication Structure defines the ontology and communication language used in the system. The Interaction Structure contains the possible system states (interaction scenes) and describes the allowed transitions between them. Finally, the Normative Structure describes norms imposed upon the roles and interaction scenes' norms.

The two remaining sub models regulate behavior within the MAS. The Social Model manages agents' enactment of roles whereas the Interaction Model serves to adjust role-enacting agents' actions during interaction scenes.

The analysis done in [4] identified five different modeling dimensions in OperA model. They are the structural dimension, encompassed by the social structure and social model; the functional dimension, represented by the interaction structure and the interaction model; the dialogical and ontological dimensions, both housed within the communication structure; and the normative dimension, associated with the normative structure.

3 Artifacts and related issues

In this section we briefly describe issues that are used along the proposed step towards achieving interoperability among different OC-MAS.

3.1 Artifacts

When dealing with the field of AI, it is common to make analogies to human societies to understand and conceive how systems should work [20]. For instance, in the case of MAS, it seems natural to compare working people to agents, given their similarities in behavior. However, it is very clear that people, when working with others, make use of many tools that facilitate coordination and cooperation. This concept, when ported to the MAS domain, gives origin to the idea of working environments and environment-oriented programming.

An artifact is the main component of a working environment. It can be defined as a passive entity, which can be managed by agents in order to perform a function [17]. Basically, it consists of an interface that agents use either to send commands or to receive information. An agent requests the artifact's functionality by triggering an operation defined in it, which is similar to a method in an object. The artifact, in its turn, communicates with the agents by updating its observable properties (analogous to attributes in an object) or by sending signals. The CArtAgO framework [18] implements artifact-based working environments through an environment-oriented approach using Java, given its similarities with an object in the object-oriented programming.

One of the main advantages in this approach, especially when dealing with multi-agent systems, is that agents can now rely on the environment to get resources and tools to promote their activities. For instance, because artifacts are easily accessible by any agent in a workspace, they can be of great importance when coordinated teamwork is necessary [18]. Therefore, it is possible to infer that making use of artifacts to implement an organizational model may bring some advantages when compared to the traditional service-based OI, as discussed in the next section.

3.2 ORA4MAS

Traditionally, the implementation of OI is based on an architecture composed of services and special agents, located in a layer inaccessible to ordinary agents and dependent on its underlying organizational model [3][10]. Although this approach successfully achieves its goals of running an OI, it has the pitfall to be too strict and inflexible, since agents are bound to the OI as they are required to previously know the OI in which they are running. As such, agents remain incapable of running in different OIs.

ORA4MAS (Organizational Artifacts for Multiagent Systems) [10] was proposed as a solution to the limits imposed by the previously mentioned approach. Its goal is to make organizations more flexible by implementing it in a layer accessible to agents, exploiting the concept of working environments discussed in section 3.1. In this new approach, an OI is composed of a set of organizational artifacts and agents responsible to deal with all aspects of the organizational model they implement. It is important to stress that ORA4MAS solution is to move the required knowledge of the OI from the agents to the organizational artifacts.

The role of artifacts in this context is to provide operations and information regarding the organization to any agent that participates in it. For instance, if an agent wants

to adopt a role, he must trigger the correspondent operation on the artifact. Moreover, he can easily get information about the organization state (for example, the available roles) by inspecting the artifact's observable properties. Organizational agents are proposed in ORA4MAS to deal with aspects that require reasoning. For instance, whenever an agent triggers a forbidden operation, one of two actions can be taken. If the violation brings a result that must be reverted, a regimentation mechanism is applied directly in the artifact, blocking that operation. However, when this is not the case (the violation brings no harm to the system), the artifact only communicates the occurrence to the organizational agent, who in turn decides on what action to take.

3.3 JACAMO

JaCaMo is a platform for programming MAS. However, it innovates and differentiates itself from other such platforms as it combines agent, organization and environment programming into a single framework.

This multiple approach is possible due to integration of several technologies into JaCaMo: (i) Jason [2], a platform for programming MAS using an agent-oriented approach; (ii) CArtAgO for an environment-oriented approach; and (iii) MOISE+, for an organizational-oriented approach. In addition, ORA4MAS provided the means to integrate MOISE+ organizational model through organizational artifacts provided by CArtAgO framework.

During runtime, agents in JaCaMo have direct access to the environment, by means of manipulating artifacts in the same way as described in CArtAgO (section 3.1). Concurrently, their actions are actively monitored by an organization, modelled in accordance to Moise+ and implemented in the environment as proposes ORA4MAS. An important feature in JaCaMo, which will be explored in this paper, is the presence of an NPL interpreter in the organizational artifacts to efficiently run the organization. As expected, this requires the organization description to be translated into NPL before it is loaded by the organizational artifacts.

3.4 Normative Programming Language

A Normative Programing Language (NPL) is, as the name implies, a programming language based on norms. In general, a norm is a statement that describes an expected pattern of behavior and the consequences of disregarding it. In addition, the language also utilizes facts, which are statements of information, and inference rules.

Usually these norms can be one of three types: obligation, permission or prohibition and are enforced by either sanctions or regimentations [11]. An obligation constitutes a behavior that must be complied; permission refers to an allowed behavior and, of course, a prohibition is a disallowed behavior.

In the case a violation to a norm occurs, the appropriate enforcement strategy should activate. Regimentations are strategies to prevent a norm infringement in the first place; therefore, no actions that would result in regiment infraction are possible. Meanwhile sanctions are punitive strategies that become effective after an infraction, so actions that fall under a sanctioned norm are indeed possible. Regimented norms

are primarily designed as means to preserve the system from otherwise harmful actions. Sanctioned norms, on the other hand, should encourage desired behavior without effectively compromising autonomy of the concerning party.

```
np       ::= "np" atom "{" ( rule | norm )* "}"
rule     ::= atom [ ":-" formula ] "."
norm     ::= "norm" id ":" formula "->" ( fail | obl ) "."

fail     ::= "fail(" atom ")"
obl      ::= "obligation(" (var | id) "," atom "," formula "," time ")"

formula ::= atom | "not" formula | atom ( "&" | "|") formula
time     ::= "(" ( "now" |
               number ( "second" | "minute" | ...))
               "(" [ ( "+" | "-" ) time ]
```

Fig. 2: NPL syntax (source [11]) - non-terminals *atom*, *id*, *var* and *number* corresponds, respectively, to *predicates*, *identifiers*, *variables*, and *numbers*, as used in Prolog.

In conclusion, the NPL serves to regulate behavior and, specifically in the case of MAS, it may be used to regulate agent behavior. Nevertheless, to make this possible, it is necessary [11]: (i) an interpreter capable of running the NPL; and (ii) the translation of the organizational specification into a normative specification.

A successful implementation of MOISE+ by means of translating it to a simplified NPL containing only two constructs, *obligation* and *regimentation*, is described in [11]. This approach used the aforementioned ORA4MAS platform and had the NPL engine embedded in the organizational artifacts. A program described in the NPL consists of (i) a set of facts and inference rules and (ii) a set of norms. Norms have unique identifiers, activation condition and consequence. Consequence could be of types *fail* or *obligation*. **Fig. 2** shows the syntax of the NPL, where *np* is a program in NPL. Two organizational models are described in section 4 using the NPL. Considering the implementation of JaCaMo and the fact that organizational constraints are embedded in an NPL program, having such a description is an important step towards achieving interoperability among organizational models during runtime. This is so because the NPL could be used as a common language to describe the organizational models.

4 Translating Organizational Models to the NPL

4.1 AGR

In order to describe the AGR model in terms of normative language, it is necessary first to obtain facts and rules that are inherent to the model, and then define the correspondent norms in terms of them.

Facts.

The following facts describe the structural dimension of AGR. Some of the notations used here are suggested in [7].

- `member(x,g):` represents that agent `x` is member of group `g`;
- `plays(x,r,g):` agent `x` plays role `r` in group `g`;
- `GStruct(g,gs):` group `g` is described by group structure `gs`;
- `roleIn(r,gs):` `r` is a role defined in group structure `gs`;
- `correspondence(role1, gs1, role2, gs2):` there is a correspondence constraint between `role1` defined in `gs1`, and `role2` defined in `gs2`, meaning that an agent who plays `role1` will be obligated to play `role2`;
- `dependence(role1, gs1, role2, gs2):` there is a dependence constraint between `role1` in `gs1` and `role2` in `gs2`, meaning that playing `role2` is a prerequisite for assuming `role1`.

The next facts relate to the dialogical dimension of AGR.

- `interact(role1,role2):` `role1` has an interaction relationship with `role2`, meaning that an agent playing `role1` can send messages to another agent playing `role2`. Thus, an interaction is a directed relation;
- `msg(x,y,content):` agent `x` has sent to agent `y` a message `content`.

Rules.

Next, inference rules that help describe the state of the organization will be presented. Both rules shown here relate to the structural dimension of AGR.

The first rule tells whether a certain role is defined in a group instantiated in the organization. Remember that roles are described directly in a group structure.

```
rdefined(g,r) :- Gstruct(g,gs) & roleIn(r,gs)
```

The second one informs if two agents are members of the same group, i.e. whether there is a group g in which both agents participate.

```
samegroup(x,y) :- member(x,g) & member(y,g)
```

Norms.

Finally, it is possible to define the set of norms that characterize this model, starting with the ones related to the structural dimension.

- Norm role_member: if an agent `x` plays a role `r` in a group `g`, he must be a member of this group.

```
role_member: plays(X,R,G)& ¬member(X,G)->
fail(role_member).
```

- Norm role_corresp: implements the correspondence constraint between roles.

```
role_corresp: plays(X,Role1,G1)& GStruct(G1,GS1)&
GStruct(G2,GS2)&
correspondence(Role1,GS1,Role2,GS2)->
obligation(X,role_corresp,plays(X,Role2,G2),now+t).
```

— Norm role_dep: implements the dependence constraint between roles

```
role_dep: plays(X,Role1,G1)& ¬plays(X,Role2,G2)&
GStruct(G1,GS1)& GStruct(G2,GS2)&
dependence(Role1,GS1,Role2,GS2)->
fail(role_dep).
```

The next norms are related to the dialogical dimension of AGR.

— Norm group_comm: two agents can communicate with each other only if they are members of at least one group in common.

```
group_comm: msg(X,Y,C)& ¬samegroup(X,T)->
fail(group_comm)
```

— Norm inter_comm: two agents may communicate only if there is an interaction defined between them.

```
inter_comm: msg(X,Y,C) & ¬interact(X,Y)->
fail(inter_comm)
```

4.2 OPERA

The following facts and rules are related to the structural dimension of the OperA model.

Facts.

— `plays(x,r)`: indicates that agent `x` plays (enacts) role `r`;
— `objective(obj,r)`: indicates that objective `obj` is an objective of role `r`;
— `sub-objective(sobj,obj)`: indicates that objective `sobj` is a sub-objective of objective `obj`;
— `contains_sub-objectives(obj)`: indicates that objective `obj` has sub-objectives;
— `right(rt,r)`: indicates that `rt` is a right of role `r`;
— `completed(obj)`: indicates that requirements of objective `obj` have been fulfilled.

Rules.

This rule indicates whether an objective has been achieved. An objective is achieved if all of its sub-objectives are achieved or if, in the case it has no sub-objectives, it has been completed.

```
achieved(obj):-[contains_sub-objectives(obj)
&[achieved(sobj1) &achieved(sobj2) & … &
achieved(sobjN)]]|[¬contains_sub-objectives(obj) &
completed( obj)]
```

In sequence, the facts, rules and norms related to the OPERA functional dimension are presented.

Facts.

- `in_progress(s)`: indicates that scene `s` is in progress;
- `finished(s)`: indicates that scene `s` has finished;
- `start(s,x):` indicates that agent `x` has initiated scene `s`;
- `end(s,x):` indicates that agent `x` has terminated scene `s`;
- `scene_manager(x)`: indicates that agent `x` may initiate and terminate scenes;
- `from(t,s):` indicates that scene `s` transits from transition `t`;
- `to(s,t):` indicates that scene `s` transits to transition `t`;
- `and(t):` indicates that scene transition `t` is an `AND` operator;
- `or(t):` indicates that scene transition `t` is an `OR` operator;
- `xor(t)`: indicates that scene transition `t` is a `XOR` operator;
- `part(l,s):` landmark `l` is part of scene `s`;
- `order(l1,l2)`: landmark `l1` is ordered before landmark `l2`;
- `state_requirement(obj,l):` landmark `l` requires that objective `obj` is achieved.
- `state_negative_requirement(obj,l):` landmark `l` requires that objective `obj` is not achieved.
- `scene_requirement(l,s):` scene `s` requires that landmark `l` has been reached for before starting.

Rules.

Scene transitions are valid when the transition requirement is satisfied. The requirement depends on the transition type: AND transitions require that all scenes leading to the transition be finished, OR transitions require that at least one scene leading to the transition be finished and XOR transitions require that one, and only one, scene leading to the transition be finished.

```
Valid(t) :- [and(t)&[[to(s1,t)&finished(s1)]& …
&[to(sN,t)&finished(sN)]]]
|[or(t)&[[to(s1,t)&finished(s1)]|…|[to(sN,t)&finished(sN)
]]
|[xor(t)&[[to(s1,t)&finished(s1)]& ¬… &
¬[to(sN,t)&finished(sN)]]| …
|¬[to(s1,t)&finished(s1)]& ¬… & [to(sN,t)&finished(sN)]]]
```

A landmark is considered reached if all state requirements and state negative requirements are met, and all previous landmarks have also been reached.

```
reached(L) :- [state_requirement(obj1,L)&achieved(obj1)]
& … & [state_requirement(objN,L)&achieved(objN)] &
[state_negative_requirement(nobj1,L)&¬achieved(nobj1)] &
…&[state_requirement(nobjN,L)&¬achieved(nobjN)]
& Order(L0,L)&reached(L0) & … & Order(LN,L)&reached(LN)
```

Norms.

```
ended_without_permission: end(S,X) & ¬scene_manager(X) ->
fail(ended_without_permission)

started_without_permission: start(S,X) &
¬scene_manager(X) -> fail(started_without_permission)

started_at_inappropriate_time: start(S,X) & from(S,T) &
¬valid(T) -> fail(started_at_inappropriate_time)

and_transition: valid(T) & from(S,T) & and(T) &
scene_manager(X) ->
obligation(X,and_transition,start(S,X), now+Ts)

xor_transition: xor(T) & [ in_progress(S1) | finished(S1)
& from(T,S1] & [ in_progress(S2) | finished(S2) &
from(T,S2] -> fail(xor_transition)

started_without_requirements: start(S,X)
&scene_requirement(L,S) & ¬reached(L) ->
fail(started_without_requirements)
```

Finally, the model's explicit norms are described. Each of these explicit norms has a unique id and can either be active or inactive depending if the activation and termination conditions are met or not. They also have a maintenance condition that refers to the behavior regulated.

Norms.

```
obligation_norm_id: (activation condition) &
¬(termination condition) & (maintenance condition(R)) ->
obligation(X, obligation_norm_id, action, deadline)
```

```
permission_norm_id: (activation condition) &
¬(termination condition) & (maintenance condition(R)) &
¬right(RT,R) -> fail(permission_norm_id)

prohibition_norm_id: (activation condition) &
¬(termination condition) & (maintenance condition(R)) ->
fail(prohibition_norm_id)
```

The following assumptions were made in the OperA's translation to the normative language:

- Group notion is not present in the OperettA framework and therefore was not included in this description.
- Links existing between roles have also not been included here as they can be expressed through explicit norms.
- An action constitutes of any behavior that causes an observable change in the system (such as choosing to enact a role, achieving an objective or initiating/terminating a scene).
- The requirements for a scene to be initiated are represented by a list of landmarks.

5 Discussion

The results of the work done in [11] suggest that the adoption of the proposed normative programming language and its artifact-based interpreter is a viable solution to the interoperability problem. The key idea is to investigate the possibility of translating different organizational models other than MOISE+ to the NPL, that was left open in [11], and analyze whether the implemented infrastructure for the interpreter fits with the translation.

Our initial work consisted in understanding the main aspects of two organizational models, namely AGR and OperA, and describing them in terms of facts, rules and norms. The results presented in section 4 shows that this translation is feasible, at least for these two models.

However, it is necessary to check whether the artifact-based interpreter for MOISE+, as implemented in JaCaMo, can be used with our translation in order to effectively run the organization. This interpreter consists of two artifacts, one for each organizational dimension in Moise+ (structural and functional - see session 2.1). There is no dedicated artifact dealing with the normative dimension, since each organizational artifact runs a normative engine and therefore is responsible for a piece of the normative dimension. In other words, each of the two aforementioned artifacts deal with the set of norms related to their respective dimension.

As pointed in section 2.2, AGR allows the description of organizations with structural and dialogical dimensions, only. Thus, making an analogy to Moise+, we believe that one artifact dealing with the dialogical dimension should be necessary, while the one responsible for the functional dimension would not be used.

As for OperA, analyzed in section 2.3, the model incorporates both the functional and structural dimensions, but also dialogical, normative and ontological ones. Therefore, for this model, an additional artifact may be required to deal with dialogical dimension; in addition, the existing structural and functional artifacts may suffer modifications to incorporate elements not present in other organizational models. Moreover, since the ontology is directly translated into NPL facts, there is no need of an artifact dedicated to the ontological dimension. Finally, we presume that there is no need for a normative artifact for the similar reasons as in Moise+. Currently we are implementing the adaptations for the existing organizational artifacts in order to confirm such assumptions.

6 Conclusion and Future work

In this paper, we analyzed the use of a NPL to describe organization specifications from two organizational models: AGR and Opera. The analysis showed that the NPL proposed by Hübner et al in [11] could, in fact, be used as is to describe organization specifications from the aforementioned models. Nevertheless, the possibility of including artifacts to deal with dialogical issues must be investigated to realize the concrete implementation of these models through NPL. This is the first step in achieving interoperability through an environment-oriented approach. Further work will follow this way by analyzing organizational artifacts implementation provided by JaCaMo concerning their conformity with other models' specifications related to the organizational dimensions they are responsible for. After, decision about including new artifacts and/or modifying existing ones must be taken and implemented.

7 Acknowledgements

Fabio T. Muramatsu is partially supported by grant #013/17948-7, São Paulo Research Foundation (FAPESP). Tomas A. Vitorello is partially supported by grant 013/17973-1, São Paulo Research Foundation (FAPESP). Anarosa A. F. Brandão is partially supported by grant #010/2640-5, São Paulo Research Foundation (FAPESP).

8 References

1. Boissier, O.; Bordini, R.; Hübner, J.; Ricci, A.; Santi, A. Multi-agent oriented programming with JaCaMo, Science of Computer Programming 78 (2013) 747–761.
2. Bordini, R.; Hübner, J.; Wooldridge, M.. Programming Multi-Agent Systems in AgentSpeak using Jason, Wiley, 2007.
3. Coutinho, L.R. ; Brandão, Anarosa A.F. ; Sichman, J.S. ; Boissier, O. . Organizational Interoperability in Open Multiagents Systems - An Approach based on Metamodels and Ontologies. In: Proc.of the 2nd Workshop on Ontologies and Metamodels in Software Engineering WOMSDE 2007, 2007.

4. Coutinho, L., Sichman, J., Boissier, O., Modelling Dimensions for Agent Organizations. In Dignum, V. (Ed.). Handbook of research on multi-agent systems: semantics and dynamics of organizational models. Hershey: IGI Global, 2009. Cap.II, pp.18-50.
5. Coutinho, L.R.; Brandão, Anarosa A.F.; Sichman, J.S.; Hubner, J.F.; Boissier, O.. A Model-based Architecture for Organizational Interoperability in Open Multiagent Systems. In: Padget et al. (Org.). Coordination, Organizations, Institutions and Norms in Agent Systems V. Berlin: Springer, 2010, LNCS v. 6069, p. 102-113.
6. Dignum. V. A Model for Organizational Interaction: based on Agents, founded in Logic. SIKS Dissertation Series 2004-1. SIKS, 2004. PhD Thesis.
7. Ferber, J., Gutknecht, O., Michel, F., From Agents to Organizations: An Organizational View of Multi-agent Systems. In P. Giorgini, J.P. Müller, J. Odell (Eds.): AOSE 2003, LNCS 2935, pp. 214–230, 2004. Springer-Verlag Berlin Heidelberg 2004.
8. Gasser, L.. Organizations in multi-agent systems. In Pre-Proc. of the 10th European Worshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'2001), Annecy, 2001.
9. Gutknecht, O. and Ferber, J. Madkit: a Generic Multi-Agent Platform (short paper). Autonomous Agents (AGENTS 2000), Barcelona, ACM Press, pp. 78-79, 2000.
10. Hübner, J., Boissier, O., Kitio, R., Ricci, A., Instrumenting multi-agent organisations with organisational artifacts and agents. In Auton Agent Multi-Agent Syst (2010) 20:369–400.
11. Hübner, J.F., Boissier, O., Bordini, R.H., A normative programming language for multi-agent organizations. In Annals of Mathematics and Artificial Intelligence, v. 62, n. 1-2, pp. 27-53. Springer Science+Business Media B.V. 2011.
12. Hübner, J. F.; Sichman, J. S.; Boissier, O. *Using the MOISE+ for a cooperative framework of MAS reorganization.* In: Proc. 17th Brazilian Symposium on Artificial Intelligence (SBIA'04), 2004. A. Bazzan and S. Labidi eds. LNAI, vol. 3171, p 506-515, 2004. Springer-Verlag.
13. Hübner, J.; Bordini, R.; Picard, G.. Using Jason and MOISE+ to develop a team of cowboys. In: Programming Multi-Agent Systems, LNCS. Volume 5442, 2009, pp 238-242.
14. Hübner, J.; Sichman, J.; Boissier, O.. A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In: Proceedings of the 16th Brazilian Symposium on Artificial Intelligence (SBIA'02). Berlin: Springer, 2002. v. 2507. p. 118-128.
15. Hubner, J; Sichman, J.; Boissier, O.. S-MOISE+: A middleware for developing Organised Multi-Agent Systems. In. Intl Workshops ANIREM 2005 and, OOOP 2005, Revised Selected Papers, Boissier, et al (Eds), LNCS, 3913, 2006, pp 64-78.
16. Lemaıtre, C. and Excelente, C.B. Multi-agent organization approach. In Garijo and Lemaıtre, Eds, Proceedings of II Iberoamerican Workshop on DAI and MAS, 1998.
17. Omicini, A., Ricci, A., Viroli, M.. Artifacts in the A&A meta-model for multi-agent systems, Auton Agent Multi-Agent Syst (2008) 17:432–456.
18. Ricci, A., Viroli, M., Omicini, A., CArtAgO: A Framework for Prototyping Artifact-Based Environments in MAS. In D. Weyns, H.V.D. Parunak, and F. Michel (Eds.): E4MAS 2006, LNAI 4389, pp. 67–86, 2007. Springer-Verlag Berlin Heidelberg 2007.
19. Ricci, A.; Piunti, M.; Acay, L.D.; Bordini, R.; Hubner, J.; Dastani, M. Integrating Heterogeneous Agent Programming Platforms within Artifact-Based Environments, Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), Padgham, Parkes, Müller and Parsons (eds.), May, 12-16. , 2008, Estoril – Portugal, pp. 225 – 232.
20. Wooldridge, M. An introduction to Multiagent Systems, Wiley & Sons, 2nd edition, 2009.