Modeling and Validating Norms[*]

Viviane Torres da Silva   Christiano Braga

Computer Science Department, Universidade Federal Fluminense (UFF)
Rua Passos da Pátria 156, Bloco E, 24210-240, Niterói, Brazil
{viviane.silva, cbraga}@ic.uff.br

Abstract. Norms describe the permissions, prohibitions and obligations of agents in multi-agent systems in order to regulate their behavior. In this paper we propose a normative modeling language that makes possible the modeling of norms motivating the modeling of such norms together with the non-normative part of the system. In addition, we also propose a mechanism to validate the norms at design time, i.e., to check if the norms respect the constraints defined by the language and also their possible conflicts.

Keywords: norm, modeling, validation, conflict, metamodel.

1 Introduction

Norms are used to regulate the behavior of the agents in open multi-agent systems (MAS) by describing their permissions, prohibitions and obligations. The definition of norms is an important part of the specification of a system and should be treated as an important task of MAS design. Methodologies such as Gaia [29][29], MaSE [5], SODA [20] and PASSI [3] [10] propose the specification of organization rules (or norms) during the analysis phase and recognize the need to associate these rules with design elements. However, there are still few modeling languages that support the modeling of norms together with the modeling of the entities that compose a MAS.

It is important to consider norms while designing a MAS since:

(i) *Norms refer to actions, agents and roles that compose a system.* They specify the actions that agents playing roles in the system are obliged, permitted or prohibited to execute. Therefore, redesigning the system, for instance, by excluding a role, may affect the norms. On the other hand, the definition of a new norm will only be possible if the actions, agents and roles being mentioned in the norm are being considered in the system design.

(ii) *Norms' conflicts can cause the redesign of a system.* Two norms are in conflict, for instance, if one gives a permission and another a prohibition to an agent to execute the same action in the same time frame. When it occurs, it is necessary to rewrite one of the norms in order to eliminate the conflict. While rewriting the norm, it may be desired, or even necessary, to redesign the system.

[*] The present work has been partially funded by the Spanish project "Agreement Technologies" (CONSOLIDER CSD2007-0022) and by the Brazilian research council CNPq under project no 550865/2007-1.

The two main goals of this paper are: (i) to support norm modeling during the design phase of a MAS and (ii) to define a technique to check possible conflicts between two defined norms at design time. We propose a normative modeling language, called NormML, that can be used during the design phase of a MAS to model the corresponding norms and an invariant-based technique to check for *well-formedness* of the norms and conflicts between two norms. Such invariants are defined over the *metamodel* of NormML.

The novelty of our approach is twofold: first, the modeling language itself, to model norms and second a validation technique that, when supported by a tool (Section 3.3), can automatically check conflicts between norms at design-time. None of the proposed methodologies or modeling languages for MAS is able to represent the three norm kinds (permission, obligation and prohibition) and to check their conflicts.

This paper is organized as follows. Section 2 provides some background material and Section 3 introduces our normative modeling language and tool used to automatically check for conflicts and query the norms model. In Section 4 we present related work. Section 5 concludes the paper with final remarks and discusses future work.

2 Background

NormML is a modeling language to specify norms that constraint the behavior of agents in MAS. Our modeling language was designed with the perception that *norm specification in MAS design and security policy specification in role-based access control (RBAC)* [10] *design are closely coupled issues*. RBAC security policies specify the *permissions* that a *user* has under a given *role*, while trying to access system *resources*. In MAS we specify the *norms* that regulate the *behavior* (or actions) of an *agent* playing a given *role*.

In this section we briefly provide background material for the rest of this paper. In Section 2.1 we introduce the necessary norm-related terminology that will be used throughout the paper. Section 2.2 introduces basic notions of models and metamodels, necessary to understand the design of NormML. In Section 2.3 we introduce Secure UML [1], a UML-based [18] modeling language for RBAC, which we extend with normative-related concepts. Such an extension gives rise to NormML.

2.1 Norms

A norm can be used to regulate the interaction between two agents—those norms are called dialogical norms [10]—and to regulate the access to resources, the entering and leaving of agents in organizations and environments, and the permissions to play roles.

A norm describes an action that is being permitted, obligated or prohibited, the entity whose behavior is being regulated (an agent, a role or an agent playing a given role) and a set of conditions to activate and deactivate the norm.

2.2 Models and metamodels

A modeling language provides a vocabulary (concepts and relations) for creating *models*. Such vocabulary is described by the *metamodel* of the modeling language which elements formalize the language concepts and their relationships. A metamodel may include invariants that specify additional properties that the models must fulfill as instances of the metamodel. Such invariants may specify the *well-formedness* conditions of a model with respect to its metamodel and the *consistency* conditions between metamodel concepts.

When UML is chosen as metalanguage, a metamodel is represented by a class diagram and its invariants are written in OCL (Object Constraint Language) [17]. This is the choice followed in this paper.

2.3 Secure UML

Secure UML provides a language for modelling *Roles*, *Permissions*, *Actions*, *Resources*, and *Authorization Constraints*, along with the relationships between permissions and roles, actions and permissions, resources and actions, and constraints and permissions. The actions described in the language can be either *Atomic* or *Composite*. The atomic actions are intended to map directly onto actual operations of the modeled system (delete, update, read, create and execute). The composite actions are used to hierarchically group atomic ones.

SecureUML leaves open what the protected resources are and which actions they offer to clients. ComponentUML [1] is a simple language for modeling component-based systems that provides provides a subset of UML class models: entities can be related by associations and may have attributes and methods. Therefore, *Entity*, *Attribute*, *Method*, *Association* and *AssociationEnd* are the possible protected resources. Figure 1 illustrates the metamodel of SecureUML+ComponentUML[†]. By using such SecureUML+ComponentUML[‡] it is possible, for instance, to specify the permissions a user playing a given role must have to execute a method (or to update an attribute) of a resource. In order to do so, it is necessary to instantiate the metaclasses *User*, *Role*, *Permission*, *ActionExecute*, *Method* (or *ActionUpdate*) and *Attribute*.

3 NormML: A Normative Modeling Language

NormML is a UML-based modeling language for the specification of norms in MAS. The choice for UML as metalanguage allows for an easy integration of NormML with UML-based MAS modeling languages such as AUML[19], AML[4] and MAS-

[†] The metamodel of SecureUML+ComponentUML (from now referred as SecureUML metamodel) is available at http://www.ic.uff.br/~viviane.silva/normML/secureUML.pdf

[‡] The metamodel of SecureUML+ComponentUML (from now referred as SecureUML metamodel) is available at http://www.ic.uff.br/~viviane.silva/normML/secureUML.pdf

ML[25]. Moreover, metamodel-based validation techniques may be applied to norms specified in NormML.

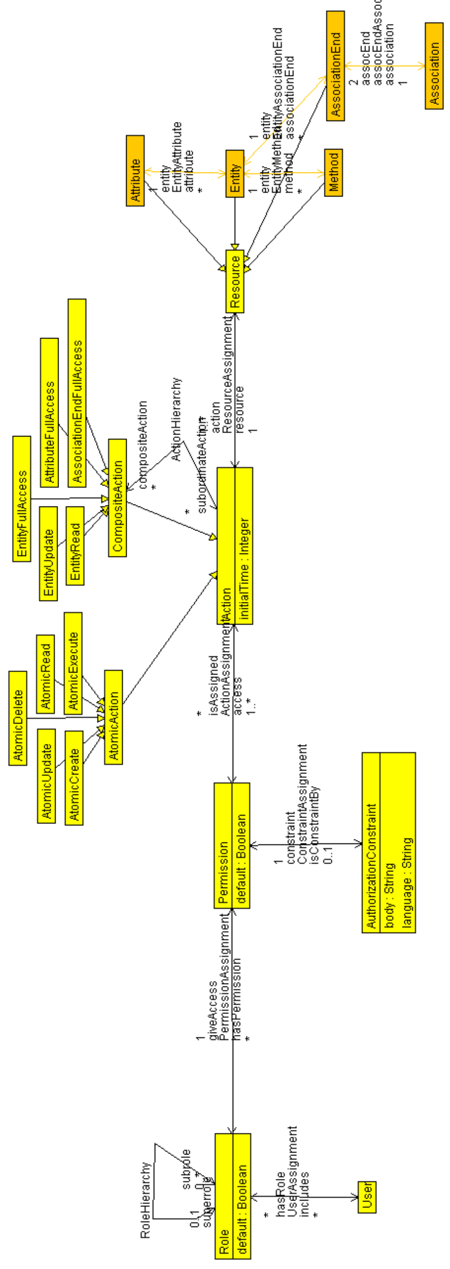

Figure 1. SecureUML+ComponenteUML metamodel

As mentioned in Section 2, NormML extends SecureUML modeling language. The NormML metamodel extends the SecureUML metamodel with the following basic elements: *Norm*, *Agent* and *AgentAction.* The NormML metamodel also includes a set of *invariants* that guarantees the well-formedness of a norm and several *operations* that are used to identify conflicts between two given norms.

3.1 The NormML Metamodel

The NormML metamodel extends the Secure UML metamodel in order to view norms as security policies, as mentioned in Section 2. While in Secure UML it is possible to define *permissions* a *user* has, i.e., the constraints that a user, in a given role, must fulfill to perform actions over the system resources, in NormML is possible to define the *norms* (obligations, permissions or prohibitions) an *entity* must obey, i.e., it is possible to describe the set of actions an agent, a role or an agent playing a role is obliged, permitted or prohibited to execute, conditioned by the execution of other actions. Figure 2 presents the NormML metamodel. (Some of SecureUML metaclasses are not presented for readability purposes.) A norm corresponds to an instance of the NormML metamodel, i.e., it is defined by instantiating several metaclasses and their relationships from the NormML metamodel. A norm may be either a permission (by instantiating the metaclass *NormPermission)*, a prohibition (by instantiating the metaclass *NormProhibition)* or an obligation (by instantiating the metaclass *NormObligation)*.

A norm may constraint the behavior of *Agents* by restricting the behavior of any given agent playing a given *Role*, or by restricting the behavior of a specific agent while playing a role. This is captured by the *Agent<->Role relationship*.

NormML inherits four resource kinds from SecureUML: *Attribute*, *Method*, *Entity* and *AssociationEnd.* It extends the set of resources with agent's actions and roles' actions represented by the metaclass *AgentAction*. Thus, it is possible to describe norms to control the access to attributes, methods, objects and association ends, and also to control the execution of the actions of agents and roles.

Each resource kind has a set of actions that can be used to control access to a resource. For instance, attributes are associated with the actions *read*, *update* and *full access (read+update)*. In the case of actions of agents and roles (*AgentAction* metaclass), the behavior that applies to it is the *execution* of the action.

Furthermore, NormML allows for the specification of the time period that a norm is *active*, which is represented by the metaclass *NormConstraint*. If a norm is conditioned by a *Before* clause, it means that the norm is active before the execution of the action(s) described in the *Before* clause. If a norm is conditioned by an *After* clause, it means that the norm is active only after the execution of the action(s) described in the *After* clause. In the case of a *Between* clause, the norm is only active during the period delimited by two groups of actions.

In order to illustrate the use of NormML to model the norms of a MAS, consider norms *N1*, *N2* and *N3* in Table 1. Figure 3, Figure 4 and Figure 5 illustrates the norm diagrams of *N1*, *N2* and *N3*, respectively.

N1: Seller is *obliged* to give the good to the buyer *after* the given buyer paid for it.

<u>*N2:*</u> Seller is *permitted* to update the price of a good *before* a buyer pays for it.
<u>*N3:*</u> Buyer is *prohibited* to return a good he/she has bought.

Table 1 - Norm example

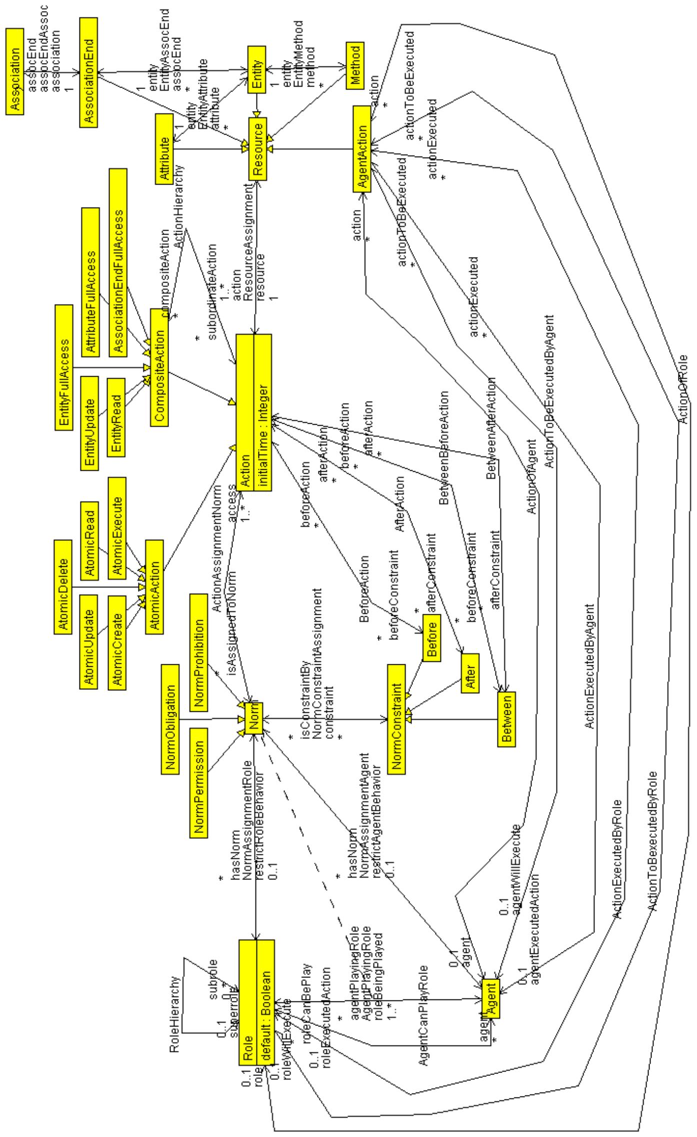

Figure 2. NormML **metamodel**

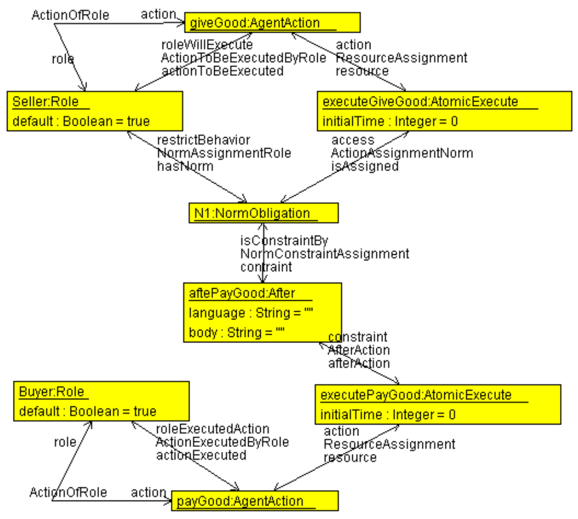

Figure 3. Norm N1 described by using NormML

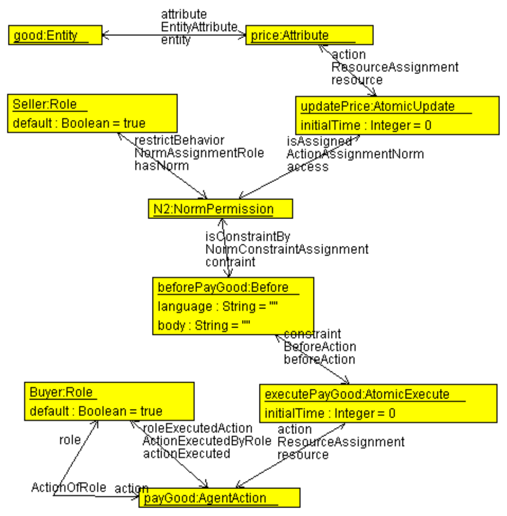

Figure 4. Norm N2 described by using NormML

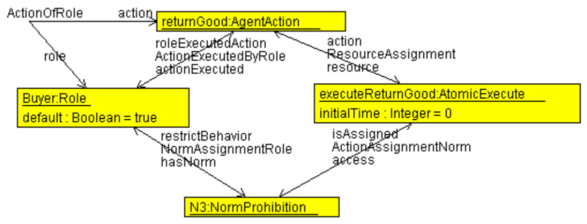

Figure 5. Norm N3 described by using NormML

3.2 Validating the Norms

The process of validating a norm encompasses two steps. First, the norm, as an instance of the NormML metamodel, is checked according to the invariants of the metamodel. The invariants check if the norm is well-formed according to the metamodel specification. The second step checks if any given two norms are in conflict.

Well-formed norms

Not all the norms that can be instantiated from the metamodel are well-formed. Examples of well-formed rules of the NormML metamodel are[§]:

<u>WFR1:</u> The resource AgentAction can only be linked with the atomic action called AtomicExecute. Any other atomic action does not apply to AgentAction.

```
context AgentAction
inv: AgentAction.allInstances-> forAll(aa|aa.action->
     select(a|not(a.oclIsTypeOf(AtomicExecution)))->isEmpty())
```

<u>WFR2:</u> The resource AgentAction cannot be constrained by Permission. Although the metaclass *Permission* is defined in the Secure UML metamodel to define the permissions a user has over resources, the resource *AgentAction* can only be used by *Norms* to restrict the actions of an agent.

```
context Permission
inv: Permission.allInstances->forAll(p|p.accesses->
   select(a|a.resource.oclIsTypeOf(AgentAction))->isEmpty())
```

<u>WFR3:</u> A norm that regulates the execution of a given action cannot be conditioned by the execution of the same action by the same agent. An agent cannot be obliged, permitted or prohibited to execute an action conditioned to the execution of such action. This rule uses four operations in order to guarantee that the action being regulated by the norm is not in the set of actions of the *Before*, *After* or *Between* constraints.

[§] Some of the well-formed rules of the NormML metamodel are available in http://www.ic.uff.br/~viviane.silva/normML/normML.pdf. We are using OCL [17] to describe the well-formed rules and also the operations to check conflicts.

```
context Norm
inv: self.GetAgentExecutedActionInBeforeConstraint->
        union(self.GetAgentExecutedActionInAfterConstraint)->
        union(self.GetAgentExecutedActionInBetweenConstraint)->
        excludes(self.GetAgentExecutedActionRestrictedByNorm)
```

Checking for Conflicts

After verifying the well-formedness of the norms, it is important to check if there are conflicts between the norms. Two norms are in conflict, or are incompatible, if:

1. One states a permission and another one a prohibition to execute the same action and such norms are active during the same period of time or during periods of time that intersects. The conflict occurs because the agent is permitted and prohibited to execute an action at the same time. Example:

   N3a: Buyer is *prohibited* to return a good it has bought.

   N3b: Buyer is *permitted* to return a good it has bought *before* using it.

   The activation time of N3a and N3b intersects since N3a states an unlimited prohibition. Thus, these norms are in conflict.

2. One norm states an obligation and another one a prohibition over the same action and such norms are active during the same period of time or during periods of time that intersect. The conflict occurs because the agent is obliged and prohibited to execute an action at the same time. Example:

   N1a: Seller is *obliged* to give the good to buyer *after* the given buyer paid for it.

   N1b: Seller is *prohibited* to give the good to buyer *before* the latter pays for it.

   The activation time of N1a and N1b do not intersect. These norms are not in conflict since the seller is not being obliged and prohibited to execute the same action during the same period of time.

3. One norm states a permission and another one an obligation over the same action and such norms are not active during the same period of time. A conflict may occur if an agent is obliged to execute an action that it is not permitted to. Example:

   N2a: Seller is *permitted* to update the price of a good *before* a buyer pays for it.

   N2b: Seller is *obliged* to update the price of a good *after* a buyer pays for it.

   The activation time of N2a and N2b do not intersect, thus these norms are in conflict.

In addition, we also consider that a conflict can be caused due to the relationship between an agent and the roles it is playing.

- *A norm applied to a role and another one applied to an agent may be in conflict:* A norm applied to a role restricts the behavior of all agents playing such role. Therefore, when searching for conflicts, it is important to check the incompatible norms that are applied to roles and also the ones applied to agents that are able to play such roles. Note that agents can play several roles.
- *A norm applied to a role and another one applied to an agent playing the role may be in conflict:* Since the norm applied to a role regulates the behavior of all agents applying such role, when searching for conflicts, it is important to check the incompatible norms that are applied to roles and to agents playing roles.
- *A norm applied to an agent and another one applied to the agent playing a role may be in conflict:* Since both norms will regulate the behavior of the same

agent, when searching for conflicts it is important to check the incompatible norms that are applied to agents and to agents playing roles.

Note that two norms applied to different roles are never in conflict even though the same agent can play both roles. Although an agent can play more than one role at the same time, an action is always executed in the context of one role. We understand that an agent must be able to obey each norm separately while playing the roles.

The operation *CheckConflict* illustrated below should be used to check conflicts between two norms. First, it checks if the norms are the same and, it they are not, if they apply to the same or related entities (as described above). Then, three important auxiliary operations[**] are used to check conflicts between an obligation and a prohibition, between an obligation and a permission and between a permission and an obligation.

```
context :: CheckConflict(norm1,norm2) : String
body if ( (norm1<>norm2)
then(
 if (CheckSameOrRelatedEntities(norm1,norm2)
 then(
  if (CheckConflictObligationProhibition(norm1, norm2) =“conflict” OR
      CheckConflictObligationPermission (norm1, norm2) =“conflict” OR
      CheckConflictPermissionObligation (norm1, norm2) =  “conflict”)
  then (“conflict”)
  else(
   if (CheckConflictObligationProhibition(norm1,norm2) = “conflictFree” AND
       CheckConflictObligationPermission(norm1,norm2) = “conflictFree” AND
      CheckConflictPermissionObligation (norm1, norm2) = “conflictFree”)
    then( “conflictFree”)
    else (“cannotBeVerified”)
  ))
 else (“conflictFree”))
else (“sameNorm”)
```

In order to exemplify one of the three auxiliary operations, let’s focus on the *CheckConflictObligationPermission* operation, since it is frequently forgotten by other authors. First, this operation checks if it is dealing with a permission and an obligation and if both norms regulate the same actions (*case 0* in operation *CheckConflictObligationPermission*). Second, it checks if the permission is not conditioned to any situation (*case 1*). In such case, there is not a conflict because the entity is always permitted to execute the action it is being obliged.

Then, it checks if the norms are constrained to the same set of constraints, i.e., if the actions that activate and deactivate the norms are the same (*case 2*). If it is the case that both norms are constrained by a *Before* clause, then there is not a conflict since the entity is being obligated to execute an action while it is permitted to. *Cases 2.2, 2.3* and *2.4* in operation *CheckConflictObligationPermission* are similar. However, if the obligation is conditioned by a *Between* clause and the permission to a *Before* or an *After* (*cases 2.5* and *2.6*) it is not possible to conclude during design time if these norms are in conflict. It will depend on the sequence of the executions of the actions that will activate the norms. On the other hand, if the permission is being constrained to a *Between* condition and the obligation by a *Before* or an *After* (*cases*

[**] The implementation of such operations can be found in http://maude.sip.ucm.es/~viviane/normML.txt

2.7 and *2.8*), both norms are in conflicts since the entity is being obliged to execute an action without permission.

If the norms are not conditioned by the same set of conditions, then it is only possible to affirm that they are in conflict (i) in the case one of the norms is conditioned to an *After*[††] and the other to a *Before*[‡‡] (*cases 3.1* and *3.2*) and (ii) in the case the permission is conditioned to a *Between* and the obligation to a *Before* (*case 3,3*). In both cases the agent is being obliged to execute a norm that it is not permitted to.

```
context :: CheckConflictObligationPermission(norm1,norm2) : String
body
if ((norm1.oclIsTypeOf(NormObligation) and norm2.oclIsTypeOf(NormPermission))
 or (norm1.oclIsTypeOf(NormPermission) and norm2.oclIsTypeOf(NormObligation)))
then ( **case 0,check if norms applies to same action**
 if (norm1.accesses=norm2.accesses
 then (  **case 1**
   if ((norm1.oclIsTypeOf(NormPermission) and
        norm1.ActionsInConstraintOfNorm()->isEmpty()() ) or
       (norm2.oclIsTypeOf(NormPermission) and
        norm2.ActionsInConstraintOfNorm()->isEmpty()()))
   then ( "conflictFree" )
   else ( **case 2**
     if (CheckSameSetOfConstraint(norm1,norm2)
     then ( **case 2.1**
       if (CheckBeforeBeforeNorms(norm1,norm2))
       then ( "conflicFree")
       else ( **case 2.2**
         if (CheckAfterAftertNorms(norm1,norm2))
         then ( "conflictFree" )
         else ( **case 2.3**
           if (CheckBetweenBetweentNorms(norm1,norm2))
           then ( "conflictFree" )
           else ( **case 2.4**
             if (CheckAfterBeforeAfterBeforeNorms(norm1,norm2))
             then ( "conflictFree" )
             else ( **case 2.5**
            if(CheckBetweenObligationBeforePermissionNorms(norm1,norm2))
               then ( "cannotBeVerified" )
               else ( **case 2.6**
                 if (CheckBetweenObligationAfterPermissionNorms(norm1,norm2))
                 then ( "cannotBeVerified" )
                 else ( **case 2.7**
                   if (CheckBetweenPermissionBeforeObligation(norm1,norm2))
                   then ( "conflict" )
                   else ( **case 2.8**
                     if (CheckBetweenPermissionAfterObligationNorms
                                                            (norm1,norm2))
                     then ( "conflict" )
                     else ( "cannoBeVerified" )
       ))))))) )
     else ( **case 3**
       **case 3.1**
       if (CheckBeforePermissionAfterObligationNorms(norm1,norm2))
       then ( "conflict" )
```

[††] Note that we are considering that the *After* condition specifies that the norm is only valid when all the actions identified in the condition are executed.

[‡‡] Note that we are considering that the *Before* condition specifies that the norm is only valid while none of the actions described in such condition is executed.

```
        else  ( **case 3.2**
          if (CheckAfterPermissionBeforeObligationNorms(norm1,norm2))
          then ( "conflict" )
          else  ( **case 3.3**
            if (CheckBetweenPermissionBeforeObligationNorms(norm1,norm2))
            then ( "conflict" )
            else  ( "cannotBeVerified" )
    )))) )
   else ( "conflictFree"  ))
  else ( "conflictFree" )
```

3.3 The Use of MOVA to Model, Validate and Query the Norms

MOVA (Modeling and Validation group) tool [6] was used as a modeling tool (i) to describe the NormML metamodel, (ii) to create the normative models, (iii) to check the well-formedness of the norms and their conflicts, and also (iv) to inspect the normative models. MOVA allows for the creation of class diagrams, the definition of a set of invariants and operations over such diagrams and checking if object diagrams respect the invariants defined in class diagrams. We used MOVA to define the NormML *metamodel* as a class diagram and to describe the *well-formed rules* of the metamodel as invariants of the class diagram. The *normative models* were then described as object diagrams and checked if they comply with these invariants.

By using MOVA it is also possible to query the object diagram (i.e., to define queries over such models) that can use operations defined in the class diagram. Such mechanism was used not only to *check the conflicts* between the modeled norms by using the operations that investigate the possible conflicts but also to *explore the normative models* themselves. Such investigation is fundamental when dealing with large-scale MAS that typically define a large number of norms. After describing hundreds norms it is almost impossible to find out, without the helping of a tool, all the norms applied to a role, for instance.

4 RELATED WORK

In this section we briefly describe how some methodologies and modeling languages deal with the modeling of the system norms. In addition we also present and compare works that have also proposed approaches to deal with norm conflicts.

4.1 Methodologies

Methodologies such as MESSAGE [3], Tropos [2] and Prometheus [22] do not address the problem of identifying and explicitly modeling norms or organizational rules. However, others such as Gaia, SODA, MaSE and PASSI state the importance of modeling organization rules during the analysis and design phases.

Gaia affirms that the explicit identification of such rules in the analysis phase is very important for the correct understanding of the characteristics that the organization-to-be must express and for the subsequent definition of the system structure by the designer. Although they have proposed a formal language to model the norms, they have not described any mechanism to validate the norms in order to find out conflicts and to verify if the elements being referred to by the norms are elements being modeled.

In [7] the authors propose the integration of organizational rules into the MaSE methodology by extending its analysis and design phases. The rules are modeled in the analysis phase, while in the design phase, the organization tasks related to the implementation and enforcement of those rules are described. Like Gaia, MaSE defines a formal language for describing norms but have not proposed how to find out norms' conflicts or how to check consistency between the elements described in the norms and the elements being modeled.

SODA states the need for modeling social rules as agents' interactions in the analysis phase and defines social models expressive enough to model the society interaction rules in the design phase. However, as opposed to Gaia and MaSE, this methodology neither presents a guideline to define such rules nor describes in details the characteristics of the proposed social models.

In the role description phase of the PASSI methodology, it is possible to introduce social rules (or organization rules) in the UML class diagrams used to model the agents, their roles and actions. The rules may be expressed in OCL or other formal, or semi-formal manner depending on one's needs. The two main drawbacks of this approach to model norms are: (i) there is not a method to verify if the elements being described in the norms are modeled in the system diagrams; and (ii) they do not propose any mechanism to check if the norms have conflicts.

4.2 Modeling Languages

Both AUML and AML recognize the need for modeling norms but have not defined any modeling technique to describe them. AML states that roles are used to define a normative behavioral repertoire of entities but has not proposed the modeling of norms. Thus, it is not possible to point out the permissions, obligations and prohibitions of an agent playing a role.

In AOR [28] the use of deontic logic to describe norms is still under investigation. Although it is possible to describe rights (or permissions) and prohibitions, it is still not possible to describe obligation. In addition, there is not any mechanism to detect norms conflicts, even though it is possible to describe them.

MAS-ML originally proposed the modeling of duties (or obligations) and rights as actions associated with roles. However, it is not possible to model more complex norms such as the ones conditioned to an event or to check their conflicts.

4.3 Other Approaches that deal with Norm Conflicts

There are several works that introduce approaches to check conflicts between norms and to solve such conflicts [9][13][21][23][26]. Since we have not presented any suggestion to the resolution of conflicts, we compare our approach with the ones that can find out the conflicts.

In [23] the authors identify three forms of conflict/inconsistency called total-total, total-partial and intersection. The approach we propose to validate the set of norms and detect conflicts can capture these three forms of conflict/inconsistency. In [9] several aspects of some types of conflicts and the problems they arise are discussed. In particular, the authors discuss the difference between deontic inconsistencies, which occur when actions are simultaneously prohibited and permitted, and deontic conflicts, which occur when actions are simultaneously prohibited and obliged. In our approach we present solutions to deal with these two types of conflicts.

The model presented in [14], called NoA, is able to detect conflicts between norms at runtime and propose resolutions to those conflicts. They state that by allowing conflicts it has partial benefits in the engineering of multi-agent systems. Thus, the main difference between their approach and ours is that our mechanism must be used to check norms at design time. In our point of view, at least the norms defined by the design must be conflict-free before the execution of the system.

Differently from us, in [12] the authors present an approach to detect conflict based on the time a norm is activated. In our approach we have not associated a norm with an activation time but with the execution of a set of actions that activates the norm. In [26] the authors present an approach to detect conflicts between related norms, i.e., norms applied to the same agent/role, restricting the same actions and whose activation periods overlap. The mechanism used to detect conflicts proposed in our paper is based on the approach presented in [26]. We extend such approach to consider conflicts between norms that state permissions and obligations—the authors in [26] only consider permissions and prohibitions or obligations and prohibitions—and to deal with activation time that is related to the execution of actions—the activation time proposed in [26] is related to values associated with attributes.

5 CONCLUSIONS AND FUTURE WORK

We have presented NormML, a normative modeling language that builds on role-based access control concepts. By using NormML it is possible to identify roles, agents and actions of a system while modeling its norms. Since NormML is based on UML, the integration of such language with any multi-agent system modeling language also based in UML, such as AUML, AML and MAS-ML, is facilitated. The roles, agents and actions identified while modeling the norms must be modeled in the agent-oriented models provided by such modeling languages.

We have defined a set of invariants and operations that makes possible the validation of the norms by verifying their well-formedness and by checking the possible conflicts between norms. We have defined three main operations to detect conflicts between an obligation and a permission, an obligation and a prohibition, and a permission and a prohibition.

We are in the process of extending the language to describe temporal restrictions and also sanctions. In order to be able to define the NormML metamodel, we have based such definition in the normative grammar proposed in [24]. This grammar extends the normative language proposed by Garcia-Camino et al. [10] with the notion of non-dialogical actions proposed by Vazquez-Salceda et al. [27] and with the definition of sanctions and relationships between norms stated by Lopez y Lopez et al. in [11][16]. However, the current version of NormML does not contemplate the definition of sanctions or temporal conditions.

It is also our intension to define a sequence diagram for NormML to describe the sequence of the executed actions. By using such diagram it will be possible to check conflicts that depend on the sequence of the executed actions (as mentioned in Section 3.2) and it will also be possible to identify the norms that are active and the ones that were violated.

REFERENCES

[1] Basin, D. A., Doser, J. and Lodderstedt, T. 2006. Model driven security: From UML models to access control infrastructures. ACMTrans. on Soft. Eng. and Met.15(1)pp.39-91

[2] Bresciani, P, Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J. 2004. Tropos: An Agent-Oriented Software Development Methodology. In JAAMAS, 8, pp. 203-236.

[3] Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., Massonet, P. 2002. Agent Oriented Analysis Using Message/UML. In Agent-Oriented Software Engineering II, LNCS 2222, pp. 119-135.

[4] Caronervenka, R., Trenccaronansky, I., Calisti M. and Greenwood, D. 2005. AML: Agent Modeling Language Toward Industry-Grade Agent-Based Modeling. InAgent-Oriented Software Engineering V, LNCS 3382, pp. 31-46.

[5] Ciancarini, P., Omicini, A., and Zambonelli, F. 2000. Multiagent System Engineering: the Coordination Viewpoint. In Intelligent Agents VI., LNAI 1767, pp. 250-259.

[6] Clavel, M., Egea, M., Silva, V. 2007. The MOVA Tool: A Rewriting-Based UML Modeling, Measuring, and Validation Tool. In Proc. Workshop de Demonstraciones de Herramientas de las Jornadas de Ingeniería del Software y Bases de Datos, pp. 393-394.

[7] Cossentino, M. 2005. From requirements to code with the PASSI methodology. In Agent-oriented Methods, Idea group, pp. 79-106.

[8] DeLoach, S. 2002. Modeling Organizational Rules in the Multiagent System Engineering Methodology", in Proc. Canadian Conf.on Artificial Intelligence, LNAI 2338, pp. 1-15.

[9] Elhag, A; Breuker, J.; Brouwer P. 2000. On the formal analysis of normative conflicts. Information and Communication Technology Law, 9(3), pp. 2007-217.

[10] Ferraiolo, D. F., Kuhn, D. R., and Chandramouli, R. 2007. Role-Based Access Control. Artech House Publishers, 2nd Edition.

[11] García-Camino, A., Noriega, P. and Rodríguez-Aguilar, J. 2005. Implementing Norms in Electronic Institutions. In Proc. of Autonomous Agents and MAS, ACMPress,pp.667-673.

[12] García-Camino, A., Noriega, P. and Rodríguez-Aguilar, J. 2007. An algorithm for conflict resolution in regulated compound activities. In Engineering Societies in the Agents World VII, LNCS 4457, Spriger-Verlag, pp.193-208.

[13] Kagal, L; Finin, T. (2007). Modeling conversation policies using permissions and obligations. Journal of Autonomous Agents and Multagent Systems, 14(2), pp. 187-206.

[14] Kollingbaum, M; Norman, T.; Preece, A; Sleeman, D. 2007. Norm Conflicts and Inconsistencies in Virtual Organisations. In Coordination, Organizations, Institutions, and Norms in Agent Systems II, LNCS, 4386.

[15] López, F. 2003. Social Power and Norms: Impact on agent behavior. PhD thesis, Univ. of Southampton, Faculty of Eng. and Applied Science, Depart. Electronics and Computer Science.

[16] López, F. Luck, M. and d'Inverno, M. 2002. Constraining autonomy through norms. In Proceedings of Autonomous Agents and Multi-Agent Systems, ACM Press, pp. 674-681

[17] Object Management group, OCL Specification, OMG. Available in http://www.omg.org/docs/ptc/03-10-14.pdf.

[18] Object Management group, UML 2.0, OMG. Available in http://www.uml.org/.

[19] Odell, J., Parunak, H., and Bauer, B. 2000. Extending UML for Agents. In Proc. Agent-Oriented Information Systems Workshop at National Conf. of AI, pp. 3-17.

[20] Omicini, A. 2002. SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems. In Agent-Oriented Software Engineering, LNCS1957, pp. 311-326.

[21] Oren, N.; Luck, M; Miles, S., Norman, T. 2008. An argumentation-inspired heuristic for resolving normative conflict. In Proceedings of the 5th Workshop on Coordination, Organizations, Institutions and Norms in Agent Systems.

[22] Padgham, L. and Winikoff, M. 2002. Prometheus: A Methodology for Developing Intelligent Agents. In Proc.Agent-Oriented Software Engineering Workshop, pp. 174-185.

[23] Ross, A. 1958. On Law and Justice. Stevens & Sons.

[24] Silva, V. 2008. From the Specification to the Implementation of Norms: An Automatic Approach to Generate Rules from Norms to Govern the Behaviour of Agents. JAAMAS, Special Issue on Norms in Muli-Agent Systems, volume 17, number1, pp. 113-155.

[25] Silva, V.; Lucena, C. 2004. From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language. JAAMAS, 9(1-2), Kluwer, pp. 145-189.

[26] Vasconcelos, W., Kollingbaum, M., Norman, T. 2007. Resolving Conflict and Inconsistency in Norm-Regulated Virtual Organizations. In Proc. AAMAS.

[27] Vázquez-Salceda, J., Aldewereld, H., Dignum, F. 2004. Implementing Norms in Multiagent Systems. In LNAI 3187, pp. 313-327.

[28] Wagner, G. 2003. The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behaviour. In Information Systems, vol. 28(5).

[29] Zambonelli, F., Jennings, N., Wooldridge, M. 2003. Developing Multiagent Systems: The Gaia Methodology. ACM Trans. on Soft. Eng. and Methodology, Vol., no 3, pp. 317-370.