On the Checking of Indirect Normative Conflicts

Jean de Oliveira Zahn, Viviane Torres da Silva
Computer Science Department
Universidade Federal Fluminense - UFF
Niterói, Brazil
{jzahn, viviane.silva}@ic.uff.br

Abstract— In open multi-agent systems, norms are being used to regulate the behavior of the autonomous, heterogeneous and independently designed agents. Norms describe the behavior that can be performed, that must be performed, and that cannot be performed in the system. One of the main challenges on developing normative systems is that norms may conflict with each other. Two norms are in conflict when the fulfillment of one norm violates the other and vice-versa. The majority works that deal with the checking of normative conflicts are not able to detect conflicts that depend on how the entities are related and how the actions are connected. They are only able to detect conflicts when the two norms regulate the same behavior executed by the same entity. In this paper, we present an approach able to check for conflicts between norms that regulate the execution of different, but related action. We describe four relationships that relate the actions of a domain and present an algorithm for the checking of indirect conflicts, i.e., conflicts between norms that do not govern the behavior of the same entity and/or that do not regulate the same action.

Keywords—norm; conflict; relationship; actions

I. INTRODUCTION

In open multi-agent systems, norms have been used to regulate the behavior of autonomous and heterogeneous entities by stating permissions, prohibitions and obligations. Due to the numerous norms that may be necessary to govern the entities of a given system, the identification of conflicts among such norms is one of the main challenges in the area.

A normative conflict arises when the fulfillment of one norm implies on the violation of the other. Several studies propose the identification of normative conflicts and the resolution of such conflicts. However, in the majority, only simple and direct conflicts are detected, i.e., conflicts between norms that govern the same behavior executed by the same entity. These approaches do not detect indirect conflict, i.e., conflicts between norms that govern different but related behavior executed by different but related entities.

The detection of indirect conflicts is only possible when the conflict checker considers the characteristics of the application domain. However, in this paper we do not concern about how the characteristics of the domain is being provided. In this paper, we extend our preliminary work presented in [9] on the identification of the relationships between entities and actions and on the definition of a normative conflict checker algorithm that consider those relationships. The main extensions presented in this paper are: (i) in the previous version, only norms defining obligations and prohibitions were considered. In this new version, we also analyzed conflicts including permissions; (ii) due to the inclusion of permission, the two relationships defined in the previous version were extended to consider conflicts among prohibitions and permissions; (iii) two new actions’ relationships were defined and the possible conflicts between permissions, prohibitions and obligations that may rise due to such relationships were analyzed; and at least but not last (iv) we present in detail the normative conflict checker algorithm that consider the relationships mentioned in the paper.

The remainder of this paper is divided in 5 sections. Section 2 presents the background material about the definition of norms and the relationships among the entities of the multi-agent system. Section 3 describes four relationships used to link actions and Section 4 presents the proposed conflict checker algorithm. Section 5 describes some related work and, finally, Section 6 states some conclusions and future work.

II. BACKGROUND

In this section, we present the background material need to understand our approach.

A. Norm Definition

According to [2] a norm prohibits, permits or obliges an entity to execute an action in a given context during a certain period of time. Several normative specifications, modelling languages, methodologies and organizational models define norms in similar ways. In all of them, a norm is associated with a deontic concept, an entity and an action (or state) that is being regulated.

Definition (Norm): A norm \( n \) is a tuple of the form \( \{\text{deoC}, c, e, a, ac, dc, s\} \) where \( \text{deoC} \) is a deontic concept from the set \{obligation, prohibition or permission\}, \( c \in C \) is the context where the norm is defined, \( e \in E \) is the entity whose behavior is being regulated, \( a \in A \) is the action being regulated, \( ac \in Cd \) indicates the condition that activates the norm, \( dc \in Cd \) is the condition that deactivates the norm and \( s \) indicates the state of the norm from the set \{fulfilled, violated, none\}. None indicates that the norm has not been fulfilled or violated yet.

The context of a norm indicates the scope where the norm is defined. A norm must be fulfilled only when the entity is executing in such context. Outside its context the norm is not
valid. In this paper we consider that a norm can be defined in the context of an organization or of an environment that is the habitat of the entities.

A norm can be defined to regulate the behavior of an agent itself, of an organization (or group of agents) – meaning that all agents playing roles in such organization must fulfill the norm –, or of a role – meaning that all agents playing such role must fulfill the norm.

The activation and deactivation conditions can state an event that can be a date, the execution of an action, the fulfillment of a norm, etc. In this paper, we will focus on the specification of a date.

B. Entities Relationships

The five relationships used to relate the entities that are being considered in this paper were defined following [9]. In that paper, we describe eight relationships: four relationships between entities (as presented below) and four relationships between actions (that are detailed in Section 3).

Inhabit: it relates an entity to the environment that it inhabits. If a norm applies in the scope of an environment, such norm applies to all entities that inhabit such environment.

Play: it relates an entity to the roles that it can play. If a norm applies to a role, it applies to all agents (or organization) playing such role.

PlayIn: it relates an entity to the organization where the entity is playing role. If a norm applies to an organization, it applies to all entities playing roles in such organization.

Ownership: it describes the roles defined in the scope of an organization. If a norm applies to an organization, it applies to all roles being played in such organization.

III. ACTIONS RELATIONSHIPS

In this section, we describe four kinds of relationships that can be used to link actions. In order to exemplify such relationships, let’s consider the simple examples below:

E.g.1: (refinement relationship) to walk and to drive are actions that specialize to move. When the agent is walking or is driving we can say that it is moving.

\[
\text{supeaction, subaction, refinement} \\
\text{(to move, to walk, refinement)} \\
\text{(to move, to drive, refinement)}
\]

E.g.2: (composition relationship) to govern the multi-agent system is an action that implies the execution of three other actions: to find out violations, to find out fulfillsments and to apply sanctions.

\[
\text{wholeAction, partAction, composition} \\
\text{(to govern, to findViolations, composition)} \\
\text{(to govern, to findFulfilments, composition)} \\
\text{(to govern, to applySanctions, composition)}
\]

E.g.3: (orthogonal relationship) to walk and to drive are orthogonal actions that cannot be executed simultaneously by the same or related entities.

\[
\text{action, action, orthogonal} \\
\text{(to walk, to drive, orthogonal)}
\]

E.g.4: (dependency relationship) to find out violations is a precondition to apply the sanctions. Therefore, we may say that these two actions are related by the dependency relationship as follows:

\[(\text{dependent, client, dependency}) \\
\text{(to applySanctions, to findOutViolations, dependency)}\]

A. Action Refinement

When the refinement relationship is defined between two actions, there is an action called subaction that refines another called superaction (that is an abstract action). The execution of the subaction achieves the goal of executing the superaction, and may also achieve other goals. If there are more than one subactions for a given superaction, the execution of any subaction achieves the goal of executing the superaction.

Obligation: If the norm applied to the superaction is an obligation, it means that the entity, whose behavior is being regulated by the norm, is obliged to execute the superaction.

Fulfillment and violation: If the superaction has more than one subaction and knowing that the states achieved by the superaction are a subset of the states achieved by any subaction, when one of the subactions is executed (in the period during while the norm is active), the entity fulfills its obligation. In order to illustrate such case, let’s consider that there is a norm obligating an entity to move. If it walks or if it drives, it will fulfill the norm.

Conflicts: A conflict between the obligation applied to the superaction and the norms applied to the subactions will arise only if all the subactions are being prohibited.

Prohibition: If the norm applied to the superaction is a prohibition, it means that the entity, whose behavior is being regulated by the norm, is prohibited to execute the superaction and achieve any of its states.

Fulfillment and violation: If the superaction has more than one subaction and knowing that the states achieved by the superaction are a subset of the states achieved by any of its subactions, if the entity executes any subaction (in the period during while the norm is active), it will be violating its prohibition. For instance, let’s assume that there is a norm prohibiting an entity to move. If it walks or if it drives it will be violating the norm.

Conflicts: The entity whose behavior is being regulated by the prohibition applied to the superaction should not execute any of the subactions in order to avoid the violation of the prohibition.

Permission: If the norm applied to the superaction is a permission, it means that the entity, whose behavior is being regulated by the norm, is permitted to execute the superaction and achieve its states.

Fulfillment and violation: By knowing that the states achieved by the superaction are only a subset of the states achieved by any subaction, the permission for executing the superaction is not granted by the subactions. We can say that the entity is partially permitted for executing the subactions since it is permitted for achieving only the states related to the
execution of the superaction. Therefore, if the entity is permitted to move it may or not be permitted to drive and to walk.

Conflicts: A conflict between the permission applied to the superaction and the norms applied to the subactions will arise only if all the subactions are being prohibited.

B. Action Composition

If the composition relationship is defined between two actions, it means that there is an action called part that is part of the action called whole and that the whole action is an abstract action. The states achieved by executing the whole action are the union of the states achieved by executing all its parts. Therefore, in order to achieve the goals of executing the whole action it is necessary to execute all its parts.

Obligation: If the norm applied to the whole action is an obligation, it means that the entity is obliged to execute the whole action and achieve its states.

Fulfillment and violation: If the whole action has more than one part and knowing that the states achieved by each part are a subset of the states achieved by the whole, the entity is obliged to execute all its parts (in the period during while the norm is active) in order to fulfill the obligation applied to the whole action. If one of the parts is not executed, the norm will be violated.

Conflict: If there is a norm prohibiting the execution of any part action, a conflict will arise between such norm and the norm applied to the whole action.

Prohibition: If the norm applied to the whole action is a prohibition, it means that the entity is prohibited to execute the whole action and achieve any of its states.

Fulfillment and violation: If the whole action has more than one part, the agent will fulfill the prohibition if it does not execute one of the parts (in the period during while the norm is active). The agent is only violating the prohibition if it executes all the parts. For instance, if there is a norm prohibiting an entity of governing a MAS, the act of find out violations or fulfilsment or of applying sanctions does not violate the norm.

Conflict: Since the violation of a prohibition applied to the whole action will only occur if the entity executes all its parts, conflicts will only arise if there are norms obligating the entity to execute all the parts actions.

Permission: If the norm applied to the whole action is a permission, it means that the entity is permitted to execute the whole action and achieve its states.

Fulfillment and violation: If the whole action has more than one part and knowing that the states achieved by each part are a subset of the states achieved by the whole, the entity is also permitted to execute all its parts (in the period during while the norm is active).

Conflicts: Knowing that the permission for executing the whole action is propagated to the part actions, a conflict will arise if a norm prohibits the execution of any part action.

C. Action Orthogonal

If an orthogonal relationship is defined between two actions, it means that both actions cannot be executed at the same time by the same or related entities.

Obligation: As stated before, if a norm applied to an action is an obligation, it means that the entity must execute such action in order to fulfill the norm.

Fulfillment and Violation: If there are two obligations whose activation period intersects, that regulate related entities (in the same context) and are applied to orthogonal actions, it means that the fulfillment of one norm will violate the other, and vice-versa.

Conflict: The conflict between two norms applied to orthogonal actions will occur if one norm obligates the execution of an action and the other obligates or permits the execution of the other actions.

Prohibition: If there are two prohibitions applied to orthogonal actions, it means that the entities (or the related entities) cannot execute those actions.

Fulfillment and violation: In case of orthogonal actions, the fulfillment of a prohibition does not imply in the violation of the other. Both norms can be fulfilled at the same time. For instance, if an entity is prohibited to walk and to drive at the same time, the fulfillment of the first norm does not imply on the violation of the other.

Conflict: There is no conflict between two norms that prohibit the execution of orthogonal actions.

Permission: If there are two permissions applied to orthogonal actions, it means that the entities (or the related entities) are free to chose to execute or not these actions.

Fulfillment and Violation: The fulfillment of a permission does not imply on the fulfillment or violation of the other. If one orthogonal action is executed, the entity will not be able to execute the other action even though being permitted.

Conflict: There is no conflict between two norms that permit the execution of orthogonal actions.

D. Action Dependency

If a dependency relationship is defined between two actions, it means that there is an action that must be executed before another action. If the client action is not executed, the dependent action cannot be executed.

Obligation: If there is an obligation applied to a dependent action, it means that the entity is obliged to execute the action.

Fulfillment and violation: in order to fulfill the obligation the entity must be able to execute all client actions before executing the dependent action. If the entity is unable to execute one of the client/precondition action (i.e., if the entity is prohibited to execute one of the clients), the entity will violate the obligation. If there is a norm obliging an entity to apply sanctions, the same entity must be able to first find out the violations since such action is the client to apply sanctions.
Conflicts: If there is an obligation governing the behavior of an entity and applied to a dependent action, there must not be any prohibition governing the behavior of the same (or related) entity and applied to any client action at the same period of time and in the same context.

Prohibition: If there is a prohibition applied to a dependent, it means that the entity is prohibited to execute the action.

Fulfillment and violation: the execution or not of the clients of a dependent action being prohibited does not imply on the fulfillment or violation of such prohibition. The prohibition can be fulfilled independently of the precondition actions that may have been executed. For instance, even being prohibited to apply sanctions, the entity can find out violations.

Conflicts: There is no conflict between a norm that prohibits the execution of a dependent action and any other norm applied to the precondition actions.

Algorithm 1 Function: Verifying Context Relationship

Require: n₁ and n₂ as parameter
function contextsRelationship(n₁, n₂)
if ((n₁.c = n₂.c) && true) return true
else
if (n₁.cType = organization) and (n₂.cType = organization) then
if (n₁.c ⊆ n₂) or (n₂.c ⊆ n₁) then
return true
endif
endif
if (n₁.cType = environment) and (n₂.cType = environment) then
if (n₁.c ⊆ n₂) or (n₂.c ⊆ n₁) then
return true
endif
endif
if (n₁.cType = organization) and (n₂.cType = environment) then
if (n₁.c ⊆ n₂) then
return true
endif
endif
return false
endfunction

Fig. 1. Verifying context relationship.

Permission: If there is a permission applied to a dependent action, it means that the entity is permitted to execute action.

Fulfillment and violation: in order to be able to use its permission to execute the dependent action, the client actions must not be prohibited. The permission for executing the dependent action can only be used if the entity is able to execute its client actions.

Conflicts: If there is a permission governing the behavior of an entity and applied to a dependent action, there must not be any prohibition governing the behavior of the same (or related) entity and applied to any client action at the same period of time and in the same context.

IV. CONFLICT CHECKER

In this section, we present the algorithm for the checking of direct and indirect normative conflicts that was implemented following the conflict cases described in the previous section. Our approach is based on the rewriting of the norms, what it similar to the unification approach used in [6].

The main algorithm (algorithm 5) uses auxiliary functions that checks (i) if the contexts are related (algorithm 1); (ii) if entities are related (algorithm 2); (iii) if the periods during while the norms are active intersect (algorithm 3); and (iv) if the actions are equal or related (algorithm 4).

As stated before, algorithm 1 indicates if the contexts of the two norms are equal or related. In case both contexts are organizations, it checks if one is a suborganization of another. In case one context is an organization and another an environment, it checks if the organization inhabits the environment. In case both contexts are environments, it checks if one is a subenvironment of another.

Algorithm 2 Function: Verifying Entities Relationship

Require: n₁ and n₂ as parameter
function entitiesRelationship(n₁, n₂)
if (n₁.e = n₂.e) then
return true
else
if (checkEntitiesRelationship(n₁.e, n₂.e) = hierarchy) then
return true
endif
endif
if (checkEntitiesRelationship(n₁.e, n₂.e) = play) then
return true
endif
if (checkEntitiesRelationship(n₁.e, n₂.e) = ownership) then
return true
endif
return false
endfunction

Fig. 2. Verifying entities relationship.

Algorithm 2 starts by checking if the entities are the same. If not, it figures out if the entities are related by one of the following relationships, as described in [9]: hierarchy (that indicates the entities inhabiting the environment), play (that indicates the entities playing roles), playin (that represents the organizations where the entities are playing roles) and ownership (that states roles defined in organizations).

Algorithm 3 Function: Verifying Time Intersect

Require: n₁ and n₂ as parameter
function timeIntersect(n₁, n₂)
if (checkActionsRelationship(n₁.a, n₂.a) = dependency) then
return true
else
if (((n₁.ac <= n₂.ac) and (n₁.dc >= n₂.ac)) or
(n₁.ac >= n₂.ac) and (n₁.ac <= n₂.dc)) then
return true
endif
endif
return false
endfunction

Fig. 3. Verifying time intersect.

The third algorithm is responsible to check if the periods during while the norms are active intersect. If a norm is deactivated after the activation of another, the activation periods intersect.
Algorithm 4 checks if the actions identified in the two norms are equal or are related. If the actions are not equal, the algorithm checks if they are related by one of the relationships described in Section 3, as follows:

Algorithm 4 Function: Verifying Actions Relationship

Require: n1 and n2 as parameter

function actionsRelationship(n1, n2)
if (n1.a = n2.a) then
    return true
else
    if (checkActRelationship(n1, n2) = refinement and (n1.deoC = O and checkAllSubActions(n1, a, F)) or (n1.deoC = F and n2.deoC = (O or F)) or (n1.deoC = P and checkAllSubActions(n1, a, F))) then
        return true
    endif
    if (checkActRelationship(n1, n2) = composition and (n1.deoC = O and checkAnyPartActions(n1, a, F)) or (n1.deoC = F and checkAllActions(n1, a, O)) or (n1.deoC = P and checkAnyPartActions(n1, a, F))) then
        return true
    endif
    if (checkActRelationship(n1, n2) = orthogonal and (n1.deoC = O and n2.deoC = O) or (n1.deoC = F and n2.deoC = F) or (n1.deoC = P and n2.deoC = P)) then
        return true
    endif
    if (checkActRelationship(n1, n2) = dependency and (n1.deoC = O and checkAnyClientActions(n1, a, F)) or (n1.deoC = P and checkAnyClientActions(n1, a, F))) then
        return true
    endif
endif
endif
endif
endif
return false
endfunction

Fig. 4. Verifying actions relationship.

- **Refinement relationship**: (i) If the execution of the superaction is being prohibited and the subaction is being obliged, the algorithm concludes that there is a potential conflict. (ii) If the execution of the superaction is being obliged, it is necessary to check if there is at least one subaction that can be executed, i.e., if there is not a norm prohibiting the execution of such action. If all subactions cannot be executed, it means that there is a potential conflict. (iii) Similar to an obligation applied to the superaction, if the execution of the superaction is being permitted, it is necessary to check if there is at least one subaction that can be executed. If not, there is a potential conflict.

- **Composite relationship**: (i) If the execution of the whole action is being obliged and the part action is being prohibited, the algorithm concludes that there is a potential conflict. (ii) If the whole action is being prohibited, and the part action is being obliged, it is necessary to check if the other part actions are also being obliged or not. If all part actions are being obliged, it means that if the agent fulfills its obligations, it will violate the prohibition. Therefore, if all part actions are being obliged, the algorithm concludes that there is a potential conflict. (iii) If the execution of the whole action is being permitted and a part action is being prohibited, the algorithm concludes that there is a potential conflict. If the agent follows its permission it will need to execute the part action that is being prohibited.

- **Orthogonal relationship**: if both norms are obliging the execution of orthogonal actions, the algorithm concludes that there is a potential conflict. If one norm oblige and the other permits, it is also a case of potential conflict.

- **Dependency relationship**: if the client action is being prohibited and the dependent action is being obliged or permitted, the algorithm concludes that there is a potential conflict.

At last but not least, algorithm 5 is responsible to coordinate all other algorithms. It calls the others in sequence and informs if the norms are in conflict or not.

Algorithm 5 Conflict Checker Main

Require: The norm base (represented by norms)

Require: The n1 and n2 from the norm base

if not conflictsRelationship(n1, n2) then
    if entitiesRelationship(n1, n2) then
        if timeActionConflict(n1, n2) then
            if actionsRelationship(n1, n2) then
                return norms are in conflict!
            endif
        endif
    endif
endif
endif
return norms are not in conflict!

Fig. 5. Conflict Checker Main.

In order to exemplify the algorithm, let’s consider the following norms:

**Norm 1**: In a workshop, the attendees are obliged to make silence during the presentation of speakers.

N1: (obligation, workshop, attendee, makeSilence, talkStarted, talkFinished, _)

**Norm 2**: In a given section of a workshop, the chair is obliged to tell the speaker that (s)he has 5 minutes to finish the talk.

N2: (obligation, section, chair, tell(5Minutes), talkStarted, talkFinished, _)

Let’s assume that the application domain states that workshops are composed of sections and that the section chair is an (kind of) attendee. In addition, it describes that to make silence is the opposite of to tell.

(workshop, section, hierarchy)
(attendee, chair, hierarchy)
(makeSilence, tell, orthogonal)

In the first step of algorithm 5, the contexts of the norms are checked and algorithm 1 concludes that they are related. Since norm 1 is applied in the context of workshops and workshops are composed of sections, N1 can be rewritten to:


The second step of algorithm 5 calls algorithm 2 to check if the entities included in the norms are related. Algorithm 2 concludes that they are related by the hierarchy relationship and N1a is written to:

\[ N1b: \text{obligation, section, chair, makeSilence, talkStartEd, talkFinished, } \]

The third step of algorithm 5 concludes that the time during with the norms are active is exactly the same. The fourth step calls algorithm 4 that checks the relationship between the actions and the deontic concepts of the norms are analyzed. Since the actions being governed by the norms are orthogonal and the deontic concepts are obligation, the algorithm concludes that N1b and N2 are in conflict.

V. RELATED WORKS

There are several works on the checking of normative conflicts and on the resolution of those conflicts. However, the majority focuses on the checking of simple conflicts. In [6], [10], [11] and [12] the authors presents approaches for the checking of normative conflicts/inconsistencies that do only consider norms applied to the same action. Indirect conflicts are thus not detected.

Indirect conflicts are detected in works such as [1], [3], [4], [5] and [10]. The approaches in [3] and [4] take into account the normative position when checking for conflicts. Normative position describes activities that are propagated to other activities. The approach presented in [3] considers that multiple, concurrent and related activities are executed by agents and present a conflict checker that takes that into account. In [4] the authors consider the composition relationship between activities, i.e., an activity can be composed of several sub-activities.

In [6] and [10] the normative conflict checker considers indirect conflicts by taking into account the domain specific relationships among actions. Two relationships among actions are defined (composition and delegation) and they also use unification to find out the norms that overlap. We claim that such approaches are incomplete since they consider only the relationships among actions and ignore the relationships among the entities. The work presented in [1] focuses on conflicts between norms defined in different contexts. Similar to such approach, the works presented in [7, 8] are able to detect conflict among different laws in different jurisdictions.

In [9] the author presented an algorithm to detect conflicts between norms. The algorithm focuses on detecting conflicts between a prohibition and an obligation that do not govern the behavior of the same entity, but entities that are somehow related.

VI. CONCLUSION AND FUTURE WORK

This work presents the identification of normative conflicts that can only be found when considering the system characteristics. In [9] the author detailed the possible relationships that can be used to relate the entities of the systems. In this paper, we focus on presenting some relationships that can be used to relate the actions executed by those entities. To consider those relationships is fundamental on the identification of the interdependencies between the norms and on the checking of conflicts.

One important limitation of the current work is that it is limited to analyze the relationships among actions. A norm can be used to regulate the execution of an action but also the achievement of a state. Therefore, we are now working on describing relationships between states and defining the connections between states and actions in order to be able to check for conflicts among norms that regulate the execution of an action and the achievement of a state. Furthermore, other action relationships can be defined and the algorithm should be extended in order to contemplate such new relationships.

In addition, we are in the process of implementing the algorithm by using Jess, a rule engine for Java platform (http://herzberg.ca.sandia.gov/). We are developing an expert system able to detect conflicts between two norms by considering the system characteristics, to indicate to the user why the norms are in conflict, providing conflict resolutions.

REFERENCES


