On the Checking of Indirect Normative Conflicts

Jean de Oliveira Zahn, Viviane Torres da Silva

Computer Science Department – Universidade Federal Fluminense (UFF)
24.210-240 – Niterói – RJ – 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 fulfilment 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.*

1. 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 fulfilment 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. It is fundamental to figure out the domain-dependent relationships among the entities and the actions identified in the norms when checking for conflicts.

In this paper, we extend our preliminary work presented in [Silva 2013] 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.

2. Background

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

2.1 Norm Definition

According to [Figueiredo et al. 2011] 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 {deoC, c, e, a, ac, dc, s} where deoC is a deontic concept from the set {obligation, prohibition or permission}, c ∈ C is the context where the norm is defined, e ∈ E is the entity whose behavior is being regulated, a ∈ A is the action being regulated, ac ∈ Cd indicates the condition that activates the norm, dc ∈ 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 fulfil the norm –, or of a role – meaning that all agents playing such role must fulfil the norm.

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

2.2 Entities Relationship

The four relationships used to relate the entities that are being considered in this paper were defined following [Silva 2013]. In that paper, we describe seven relationships: five relationships between entities (as presented below) and two 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.*

3. Actions Relationship

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 four very 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.

```
(superaction, subaction, refinement)
(to move, to walk, refinement)
(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 fulfilments and to apply sanctions.

```
(wholeAction, partAction, composition)
(to govern, to findViolations, composition)
(to govern, to findFulfilments, composition)
(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.

```
(action, action, orthogonal)
(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:

```
(dependent, client, dependency)
(to applySanctions, to findOutViolations, dependency)
```

3.1. 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 fulfil 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.

Fulfilment 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.

3.2. 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. If there is a norm obligating an entity of *governing* a MAS, in order to fulfill such norm the entity needs *to find out the violations* and the *fulfillments* and *to apply the sanctions*.

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.

Fulfilment and violation: If the *whole action* has more than one *part*, the agent will fulfil 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 *fulfilments* 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 *part 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). Following the example of *governing* a MAS, if there is a norm permitting an entity of *governing*, it is also permitting the entity *to find out violations* and *fulfilments* and *to apply sanctions*.

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*.

3.3. 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. For instance, when an agent is obliged *to walk* and also obliged *to drive* at the same time, if it fulfills one norm it will violated the other since it cannot execute both actions at the same time.

If there is one obligation and one permission applied to orthogonal actions regulating the behavior of related entities (in the same context) in period that intersects,

we may say that there is a potential conflict. For instance, when an agent is obliged *to walk* and permitted *to drive* at the same time, if it *drives*, it will violate the obligation.

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 related entities) are free to choose 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. However, 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.

3.4. 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 such 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 action, it means that the entity is prohibited to execute such 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*. Such execution does not imply on the fulfillment or violation of the prohibition.

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.

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

Fulfillment and violation: in order to be able to uses 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.

4. 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]. If the scope of a norm includes the scope of another norm, the more general norm can be rewritten to comply with the more specific norm.

The main algorithm (algorithm 5) uses auxiliary functions that checks (i) if the contexts of two norms are equal or related (algorithm 1); (ii) if the entities whose behavior is governed by the norms are the same or are related (algorithm 2); (iii) if the periods during while the norms are active intersect (algorithm 3); and (iv) if the actions being governed by the norms 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 *suborgnaization* 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 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 1 Function: Verifying Context Relationship

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

A

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
   if (checkEntitiesRelationship(n1.e, n2.e) = play) then
      return true
   endif
   if (checkEntitiesRelationship(n1.e, n2.e) = playin) then
      return true
   endif
   if (checkEntitiesRelationship(n1.e, n2.e) = ownership)
then
      return true
   endif
endif
return false
endfunction
```

B

Figure 1. (A) Verifying context relationship and (B) verifying entities relationship

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. Note that in case the actions are related by the dependency relationship it is not necessary to check if the activation periods intersect. We need (only) to guarantee that the client action can be executed before the dependent action.

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
```

Figure 3. Verifying time 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:

- *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 *subaction* 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.

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

Figure 4. Verifying actions relationship.

- *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 obliges 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. In order to exemplify the algorithm, let's consider the following norms described according to the definition in Section 2.

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,_}*

Algorithm 5 Conflict Checker Main

Require: The *norm* base (represented by *norms*)
Require: The n_1 and n_2 from the *norm* base
if *contextsRelationship*(n_1, n_2) **then**
  if *entitiesRelationship*(n_1, n_2) **then**
    if *timeIntersect*(n_1, n_2) **then**
      if *actionsRelationship*(n_1, n_2) **then**
        return *Norms are in conflict!*
      endif
    endif
  endif
endif
return *Norms are not in conflict!*

Figure 5. Conflict checker main.

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)

Based on the domain description above and executing algorithm 5, we can conclude that N1 is in conflict with N2. In the first step of algorithm 5, the contexts of the norms are checked and algorithm 1 concludes that they are related. Since norm1 is applied in the context of workshops and workshops are composed of sections, N1 can be rewritten to:

N1a:*{obligation, section, attendee, makeSilence, talkStarted, talkFinished,_}*

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

N1b:*{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.

5. Related Work

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 [Kollingbaum et al. 2008], [Vasconcelos et al. 2007], [Vasconcelos et al. 2009] and [Vasconcelos and Norman 2009] 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 [Dung and Sartor 2011], [Gaertner et al. 2007], [Garcia-Camino et al. 2006], [Kollingbaum et al. 2008a] and [Vasconcelos et al. 2007]. The approaches in [Gaertner et al. 2007] and [Garcia-Camino et al. 2006]

take into account the normative position when checking for conflicts. Normative position describes activities that are propagated to other activities. The approach presented in [Gaertner et al. 2007] considers that multiple, concurrent and related activities are executed by agents and present a conflict checker that takes that into account. In [Garcia-Camino et al. 2006] the authors consider the composition relationship between activities, i.e., an activity can be composed of several sub-activities.

In [Kollingbaum et al. 2008b] and [Vasconcelos et al. 2007] 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 [Dung and Sartor 2011] focuses on conflicts between norms defined in different contexts. Similar to such approach, the works presented in [Li et al. 2013a] and [Li et al. 2013b] are able to detect conflict among different laws defined in different jurisdictions.

In [Silva 2013] the author presented an algorithm to detect conflicts between two 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. In addition, this first version of the conflict checker algorithm can also identify conflicts between a prohibition and an obligation that are applied to different actions related by the refinement and composition relationships. In this paper, the algorithm was extended to be able to detect conflicts between prohibition and permission and to consider two new kinds of relationships that link actions: orthogonal and dependency.

6. Conclusion and Future Work

This work presents the identification of normative conflicts that can only be found when considering the system characteristics. In [Silva 2013] 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.

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, to provide possible conflict resolutions, and to apply one of the possibilities, it the user wishes to do so.

References

Dung, P., Sartor, G. (2011) “The modular logic of private international law”, In: Artificial Intelligence and Law. Springer, 19(2-3), pp. 233-261.

Figueiredo, K., Silva, S., Braga, C. (2011) “Modeling Norms in Multi-agent Systems with Norm-ML”, In: International Workshop on Coordination, Organizations, Institutions and Norms VI. LNAI 6541, Springer, pp. 39-57.

Gaertner, D., Garcia-Camino, A., Noriega, P., Vasconcelos, W. (2007) “Distributed Norm Management in Regulated Multi-agent Systems”, In: International Conference on Autonomous Agents and Multiagent Systems. ACM, pp. 624-631.

Garcia-Camino, A., Noriega, P., Rodrigues-Aguilar, J. (2006) “An Algorithm for Conflict Resolution in Regulated Compound Activities”, In: Engineering Societies in the Agents World VII, LNCS 4457, Springer, pp. 193-208.

Kollingbaum, M., Vasconcelos, W., Garcia-Camino, A., Norman, T. (2008a) “Managing Conflict Resolution in Norm-Regulated Environments”, In: Engineering Societies in the Agents World VIII, LNCS 4995, Springer, pp. 55-71.

Kollingbaum, M., Vasconcelos, W., Garcia-Camino, A., Norman, T. (2008b) “Conflict Resolution in Norm regulated Environments via Unification and Constraints”, In: Declarative Agent Languages and Technologies V, LNCS 4897,Springer,pp.158-174.

Li, T., Balke, T., De Vos, M., Padget, J., Satoh, K. (2013a) “Legal Conflict Detection in Interacting Legal Systems”, In: The 26th International Conference on Legal Knowledge and Information Systems (JURIX).

Li, T., Balke, T., De Vos, M., Satoh, K., Padget, J. (2013b) “Detecting Conflicts in Legal Systems”, In: New Frontiers in Artificial Intelligence. LNCS 7856, Springer, pp. 174-189.

Silva, V., (2013) “Normative Conflicts that Depend on the Application Domain”, In: Int. Workshop on Coordination, Organizations, Institutions and Norms, p.p. 119-130.

Vasconcelos, W., Kollingbaum, M., Norman, T. (2007) “Resolving conflict and inconsistency in norm regulated virtual organizations”, In: International Conference on Autonomous Agents and Multiagent Systems. ACM, pp. 632-639.

Vasconcelos, W., Kollingbaum, M., Norman, T. (2009) “Normative conflict resolution in multi-agent systems”, In: Journal of Autonomous Agents and Multi-Agent Systems. ACM, 19(2), pp. 124-152.

Vasconcelos, W., Norman, T. (2009) “Contract Formation through Preemptive Normative Conflict Resolution”, In: International Conference of the Catalan Association for Artificial Intelligence. ACM, pp. 179-188.