Verificação de conflitos normativos em sistemas multiagentes: uma abordagem visual

Daniela Godinho Yabe[1], Eduardo Augusto Silvestre[2] e Viviane Torres da Silva[3]

[1]Instituto Federal do Triangulo Mineiro – IFTM

[2]Universidade Federal Fluminense – UFF

[3]IBM Research, Brazil

danielagodinho1@gmail.com, eduardosilvestre@iftm.edu.br, vivianet@br.ibm.com

Abstract. *In multi-agent systems, a norm is simply an established, expected pattern of behavior. 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. Norms are in conflict when the fulfillment of one norm violates the other and vice-versa. Several authors have investigated normative conflicts in the literature, but the approaches are based on a strong logical-mathematical notation and was no found a simple implementation in high level language. This paper presents algorithms that check for conflicts between pairs of norms. It also presents a visual application for creating, importing and verification of conflicts in multi-agent systems.*

Resumo. *Em sistemas multiagentes, uma norma é um padrão de comportamento estabelecido e esperado. Normas descrevem o comportamento que pode ser executado, que deve ser executado e que não pode ser executado. Um dos principais desafios em sistemas multiagentes é que as normas podem entrar em conflito. Normas estão em conflito quando o cumprimento de uma norma viola uma outra norma e vice-versa. Vários autores pesquisaram conflitos normativos na literatura, mas as abordagens são baseadas em uma forte notação lógica-matemática e não foi encontrado uma implementação em linguagem de alto nível. Este trabalho apresenta algoritmos que verificam conflitos entre pares de normas. Apresenta também uma aplicação visual para criação, importação e verificação de conflitos em sistemas multiagentes.*

1. Introdução

Atualmente, os sistemas que usam agentes vem ganhando importância na pesquisa e na prática para o desenvolvimento de aplicações diversas. Segundo Russell e Norvig (2009), um agente de software é uma entidade capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores. Os agentes inteligentes podem ser classificados de acordo com a maneira que eles coletam informações e agem no ambiente. No caso de vários agentes cooperando ou disputando entre si, inseridos em um mesmo ambiente e trocando informações, chamamos esse

sistema de multiagente (SMA) (RUSSELL E NORVIG, 2009). SMAs são sociedades autônomas, heterogêneas e podem trabalhar a fim de alcançar objetivos comuns ou diferentes (WOOLRIDGE, 2009).

Um agente é um sistema de computador que está situado em algum ambiente e é capaz de realizar comportamentos autônomos neste ambiente a fim de atingir seus objetivos (WOOLRIDGE, 2009). Autonomia é considerada uma propriedade que possibilita agentes tomarem decisões e, para isto, eles consideram os seus interesses em realizar determinados comportamentos, ou seja, consideram a sua motivação para atingir determinados objetivos ou para executar determinadas ações (LÓPEZ, 2003).

A fim de lidar com a heterogeneidade, autonomia e diversidade de interesses entre os agentes da sociedade, projetistas desses sistemas estabelecem um conjunto de normas que é usado como um mecanismo de controle social que visa possibilitar que os agentes possam trabalhar em conjunto (SILVA, 2008).

Essas normas regulam o comportamento dos agentes, com definições de obrigações, proibições e permissões. Como os agentes são autônomos, podem existir situações onde um agente prefere violar uma norma a fim de realizar um determinado comportamento que é mais importante do que o cumprimento da norma. A introdução de normas em sistemas multiagentes tem sido considerada como um fator importante para garantir a eficácia dos agentes (LÓPEZ, 2003).

Neste contexto, vale ressaltar a possibilidade de existir conflitos entre normas. Os conflitos acontecem quando duas normas regulando o mesmo comportamento estão ativadas e tem restrições inconsistentes (VASCONCELOS ET AL., 2009). A verificação e a resolução de conflitos entre normas em sistemas multiagentes são importantes para manter a consistência do comportamento autônomo dos agentes.

Após uma extensa pesquisa bibliográfica, vários artigos foram encontrados (Cholvy and Cuppens (1995), Elhag et al (2000), Kollingbaum et al (2007), Vasconcelos et al (2009)). A partir da análise desses artigos constatou-se que a maioria deles utiliza uma notação lógica matemática de forma extensiva e não existe implementação dos algoritmos de verificação de conflitos em uma linguagem de alto nível. Além disso, não foi encontrada nenhuma ferramenta capaz de criar normas e realizar a verificação de pares de conflitos normativos. Os autores desenvolveram uma ferramenta onde é possível criar normas, importar normas e também verificar pares de conflitos normativos.

Este artigo apresenta uma ferramenta visual para verificação de conflitos entre pares de normas. As principais contribuições apresentadas neste artigo são: (i) algoritmos para verificação de conflitos entre pares de normas; (ii) aplicação em linguagem de alto nível que verifica conflitos entre pares de normas e (iii) uma interface visual para verificação de conflitos e possível inserção de novas funcionalidades.

O restante do artigo é organizado da seguinte forma: a Seção 2 define normas e conflitos normativos, a Seção 3 define a aplicação desenvolvida, os algoritmos e exemplos de entradas de dados no sistema. Por fim, a seção 4 apresenta as considerações finais.

2. Normas e conflitos normativos

Neste artigo, a definição de uma norma é a mesma usada por Figueiredo et. al. (2011). A definição é ampla e cobre definições apresentadas em artigos anteriores.

Várias especificações, linguagens e metodologias definem uma norma de forma semelhante. A norma é associada a um conceito deôntico, um contexto, uma entidade e uma ação (ou estado) a ser regulada.

Definição (norma): Uma norma n é uma tupla da forma {deoC, c, e, a, ac, dc} onde deoC é o conceito deôntico do conjunto {obrigação, proibição e permissão}, c é contexto onde a norma é definida, e é a entidade que está sendo regulada, a é a ação a ser regulada, ac é a condição de ativação e dc e a condição de desativação (FIGUEIREDO ET AL., 2011).

O contexto de uma norma indica o âmbito onde a norma está definida. A norma deve ser cumprida apenas quando a entidade está em execução em tal contexto. Fora do seu contexto, a norma não é válida. Quase todas as abordagens consideram que uma norma está definida no contexto de uma organização. Observa-se que algumas abordagens consideram que o contexto pode ser uma interação ou uma cena, por exemplo. Neste trabalho, consideramos que uma norma pode ser definida no contexto de uma organização ou de um ambiente que é o habitat das entidades.

Em geral, normas não são aplicadas o tempo todo, mas apenas em circunstâncias especiais ou dentro de um contexto específico. Assim, as normas devem especificar as situações onde os responsáveis devem cumpri-las ou as situações onde os responsáveis podem desconsiderá-las (LÓPEZ, 2003).

Duas normas estão em conflito se tem conceitos deônticos opostos, se estão no mesmo contexto, se são referentes a mesma entidade e ação, tem períodos de validade que se interceptam e não foram cumpridas e nem violadas.

3. Aplicação desenvolvida

A literatura que analisa conflitos normativos carece de aplicações que facilitem a visualização e o estudo de conflitos normativos. Neste trabalho, é desenvolvida uma aplicação capaz de verificar todos os conflitos entre pares de normas existentes em um sistema multiagente. Os autores desenvolveram algoritmos que oferecem ao usuário condições de inserção ou importação de normas que serão analisadas e mostradas, caso haja conflitos entre elas.

O sistema apresenta algoritmos para verificação de cenários de conflito, classes para apresentar uma norma e uma arquitetura que permite a implantação de novas funcionalidades.

A Figura 1 apresenta o diagrama de classes UML do programa implementado.

A linguagem Java foi utilizada para o desenvolvimento do programa. A IDE Eclipse foi utilizada como ambiente de desenvolvimento. As principais abstrações, bibliotecas e suas funcionalidades básicas são:

- Bahavior: classe abstrata que representa o comportamento de cada norma;
- BehaviorAtomicAction: classe concreta que herda de Behavior e instancia uma ação;
- ConflictChecker: classe concreta que possui as funções utilizadas para verificar se há conflito entre pares de normas;
- Constraint: classe abstrata que representa as condições de ativação ou desativação de cada norma, bem como o tipo e a data;
- ConstraintDate: classe concreta que herda de Constraint e armazena a data de ativação e desativação de cada norma;

- ConstraintType: define constantes para representar o tipo da restrição (data ou ação);
- Context: classe abstrata que representa o contexto da norma;
- ContextType: define constantes para representar o tipo do contexto (organização ou ambiente);
- DeonticConcept: define constantes para representar o tipo de conceito deôntico (obrigação, permissão, proibição);
- Entity: classe abstrata que representa a entidade da norma;
- EntityType: define constantes para representar o tipo da entidade (agente, papel, organização ou todos);
- Norm: classe concreta que representa uma norma;
- GUI: conjunto de classes que fornecem acesso à interface visual;
- Biblioteca *joda-time*: biblioteca utilizada para restrições temporais.

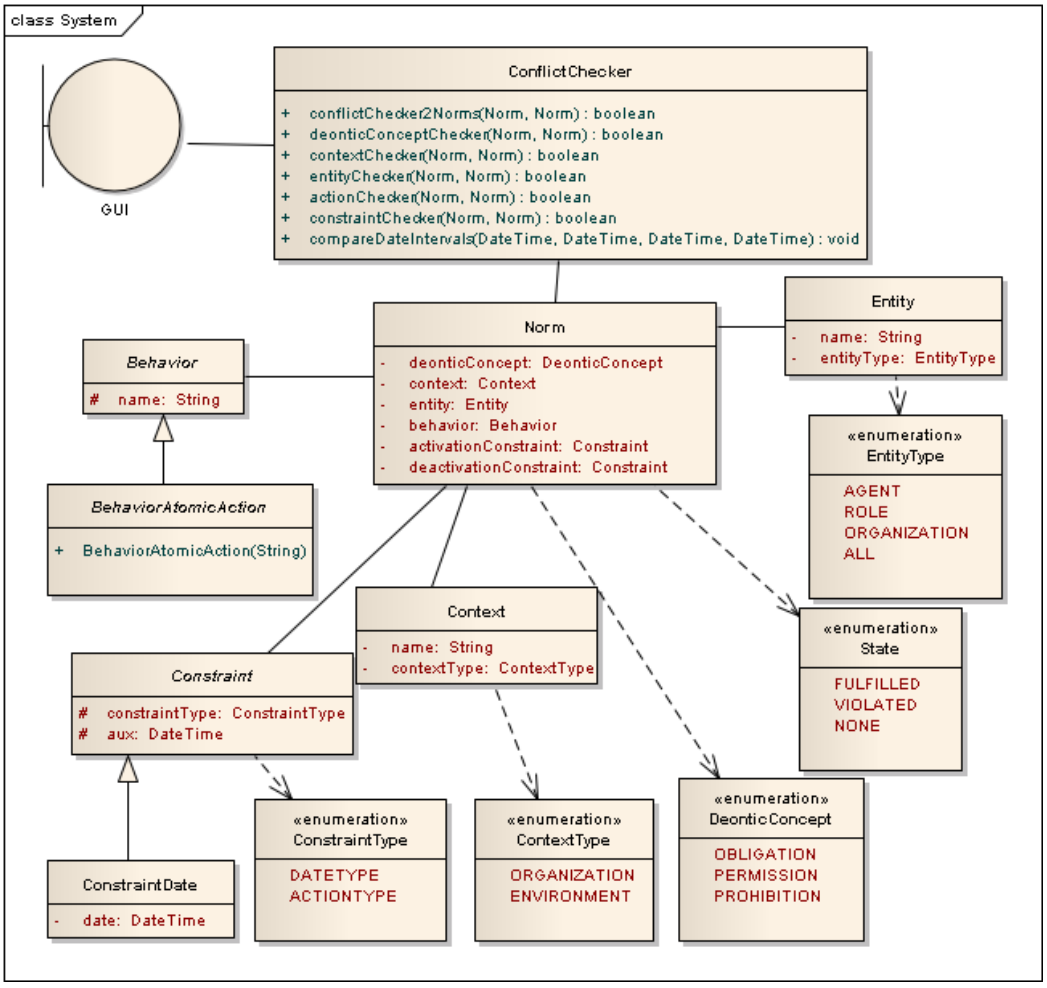

Figura 1 - Diagrama de classes UML.

A tela principal do software (Figura 2) apresenta um menu contendo as opções "Import Norm", "Conflict between 2 norms" e um formulário que permite ao usuário cadastrar normas manualmente de acordo com as características do seu sistema multiagente. Após preencher o formulário o usuário poderá carregar a norma cadastrada

para a tabela de verificação de conflitos através do botão “Save”. Ao escolher a opção “Import Norm” o usuário será direcionado para tela de importação (Figura 3) de normas pré-cadastradas no software. Estas normas pré-cadastradas permitem realizar uma análise prévia e ao mesmo tempo demonstrar a utilização da aplicação. Após o usuário inserir ou importar as normas, o usuário pode clicar no botão “Conflict between 2 norms”, o sistema irá analisar as normas selecionadas em pares e verificar se há conflito entre elas.

Nas próximas seções serão apresentados os principais algoritmos utilizados nesta abordagem. Além disso, o código implementado está disponível para download em: github.com/eduardoasilvestre/SimpleConflictsShowAllVisual.

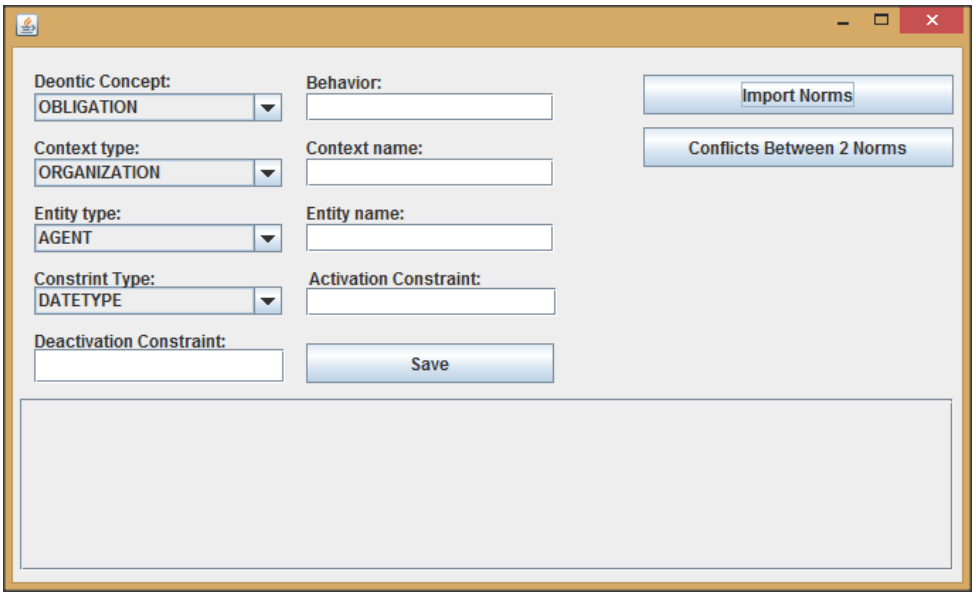

Figura 2 - Tela inicial do sistema.

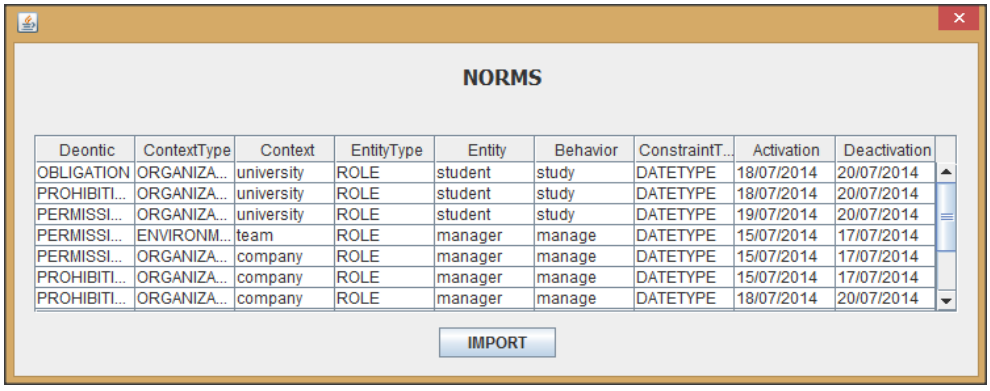

Figura 3 - Tela de importação de normas.

3.1. Algoritmos

Nesta seção, apresentaremos o algoritmo para verificação de conflitos entre pares de normas que foi implementado. O Algoritmo 1 recebe uma lista de normas que são cadastradas ou importadas pelo usuário. Desta forma, o algoritmo verifica o conflito direto entre duas normas utilizando funções auxiliares que analisam os conceitos

deônticos, ações, entidades, contextos e a condição de ativação e desativação de cada norma.

Algoritmo 1 Main

```
normSet ← receives the norms of the GUI
for i ← 1 until size(normSet) do
  for j ← i + 1 until size(normSet) do
    norm1 ← normSet[i]
    norm2 ← normSet[j]
    conclflictChecker ← conclflictChecker2Norms(norm1,norm2)
    if (conclflictChecker = true) then
      print "The following norms are in conflict"
      print norm1
      print norm2
    endif
  endfor
endfor
```

Algoritmo 1 - Algoritmo principal (main).

O Algoritmo 2 recebe as normas de entrada e apenas envia para funções específicas para análise.

Algoritmo 2. Função: Chama as funções de verificação de conflito

```
Require: norm1 and norm2 as parameter
function conclflictChecker2Norms(norm1, norm2)
      if (deonticConceptChecker(norm1, norm2) = false) then
        return false
      endif
      if (stateChecker(norm1, norm2) = false) then
        return false
      endif
      if (contextChecker(norm1, norm2) = false) then
        return false
      endif
      if (entityChecke(norm1, norm2) = false) then
        return false
      endif
      if (constraintChecke(norm1, norm2) = false) then
        return false
      endif
      if (actionChecke(norm1, norm2) = false) then
        return false
      endif
      return true
endfunction
```

Algoritmo 2 - Chama as funções de verificação de conflito.

O Algoritmo 3 analisa se os conceitos deônticos das duas normas são opostos, verificando se há conflito entre uma norma que obriga e outra que proíbe, uma norma que proíbe e outra que permite e uma norma que obriga e outra que permite.

Algoritmo 3. Função: Verifica conceito deôntico

```
Require: norm1 and norm2 as parameter
function deonticConceptChecker(norm1, norm2)
      if ((norm1.deonticConcept = PERMISSION) and (norm2.deonticConcept  = OBLIGATION) or
```

```
(norm1.deonticConcept = PROHIBITION)  and (norm2.deonticConcept =PERMISSION) ) then
        return true
    endif
    if ((norm2.deonticConcept = PROHIBITION) and (norm1.deonticConcept  = OBLIGATION) or
(norm2.deonticConcept =  PROHIBITION)  and (norm1.deonticConcept =PERMISSION) ) then
        return true
    endif
    return false
endfunction
```

Algoritmo 3 - Verifica o conceito deôntico entre pares de normas.

O Algoritmo 4 analisa se as duas normas tem as mesmas ações.

Algoritmo 4. Função: Verifica ação

```
Require: norm1 and norm2 as parameter
function actionChecker(norm1, norm2)
    if (b1.behavior = null or b2.behavior = null)  then
        return false
    endif
    return (b1.behavior = b2.behavior)
endfunction
```

Algoritmo 4 - Analisa a ação entre pares de normas.

O Algoritmo 5 verifica se as entidades de cada norma são idênticas.

Algoritmo 5. Função: Verifica entidade

```
Require: norm1 and norm2 as parameter
function entityChecker(norm1, norm2)
    return (norm1.entity = norm2.entity)
endfunction
```

Algoritmo 5 – Analisa a entidade entre as normas.

O Algoritmo 6 analisa se o contexto de ambas as normas são iguais.

Algoritmo 6. Função: Verifica contexto

```
Require: norm1 and norm2 as parameter
function contextChecker(norm1, norm2)
    return (norm1.context = norm2.context)
endfunction
```

Algoritmo 6 - Analisa o contexto entre pares de normas.

Enquanto o Algoritmo 7 verifica se a condição de ativação e desativação de cada norma são iguais ou se interceptam.

Algoritmo 7. Função: Verifica condição de ativação e desativação

```
Require: norm1 and norm2 as parameter
function constraintChecker(norm1, norm2)
    if ((norm1.dConstraint > norm2.aConstraint) or (norm2.dConstraint > norm1.aConstraint)) then
        return true
    endif
    return false
endfunction
```

Algoritmo 7 - Verifica a condição de ativação e desativação das normas.

3.2. Exemplo

Para facilitar a compreensão dos algoritmos juntamente com a aplicação visual, esta seção irá fornecer um exemplo de uso do programa. Suponha que o Agente A tenha que cumprir as normas N1, N2 e N3:

N1 = O agente estudante é obrigado a estudar na universidade entre x e y.
N2 = O agente estudante é proibido de estudar na universidade entre x e y.
N3 = O agente estudante é permitido de estudar na universidade entre x e y.

A partir da definição de uma norma apresentada na Seção 2 pode-se obter:

N1 = {deoC=OBRIGADO, c=universidade, e=estudante, a=estudar, ac=x, dc=y}.
N2 = {deoC=PROIBIDO, c=universidade, e=estudante, a=estudar, ac=x, dc=y}.
N3 = {deoC=PERMITIDO, c=universidade, e=estudante, a=estudar, ac=x, dc=y }.

Onde, deoC é o conceito deôntico, c é o contexto onde a norma é definida, e representa a entidade que tem de cumprir a norma, a ação a ser executada, ac o momento que a norma começa a ser válida e dc o momento em que a norma não é mais válida.

A Figura 4 mostra as normas instanciadas e prontas para serem verificadas. A aplicação permite que o usuário crie normas manualmente de acordo com as especificidades do seu sistema multiagente. As normas criadas são carregadas para uma tabela na tela. Além disso, é permitido que o usuário selecione quais normas ele deseja verificar o conflito.

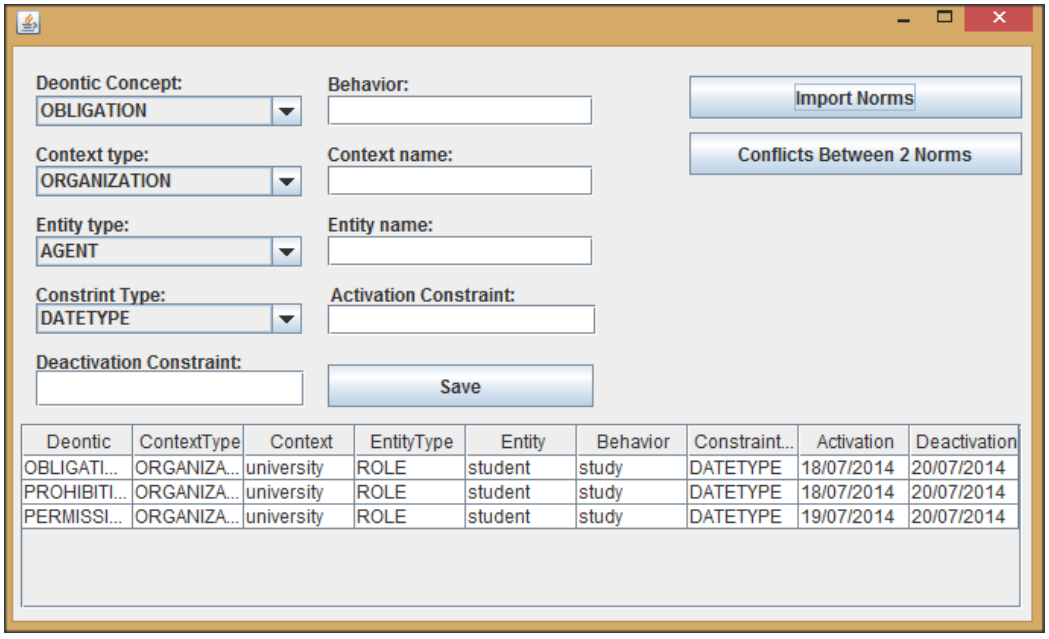

Figura 4 - Tela inicial com normas preenchidas.

Após a execução do algoritmo a aplicação retornará que existe conflito entre as normas N1 e N2 e entre as normas N2 e N3 (Figura 5). O agente A não é capaz de

cumprir porque, no primeiro caso, uma norma proíbe e outra obriga, já no segundo caso, uma proíbe e a outra permite

Figura 5 - Mensagem mostrando as normas que estão em conflito.

4. Considerações finais

Os sistemas multiagentes vem ganhado maior importância nos últimos anos, tanto nas universidades quanto nas empresas. Devido a essa amplitude existem várias problemas em aberto. Um dos principais problemas é garantir a existência de conjunto de normas sem conflito. E, a partir deste problema vários estudos vêm sendo realizados.

Este trabalho identificou uma carência na verificação de conflitos normativos. A maioria dos trabalhos na literatura é baseado em forte notação matemática, mas não apresenta uma implementação algoritmos/códigos que facilitem a solução do problema.

A principal contribuição deste trabalho é a criação de algoritmos e a implementação em linguagem de alto nível de soluções que verificam conflitos diretos entre pares de normas em sistemas multiagentes. Além disso foi desenvolvida uma aplicação visual para criação e análise de conflitos entre pares de normas.

Como a arquitetura suporta inserção de novas funcionalidades, para trabalhos futuros pretende-se implementar novos algoritmos que possam resolver problemas de conflitos normativos recorrentes.

Referências

Cholvy, L., & Cuppens, F. (1995, May). Solving normative conflicts by merging roles. In Proceedings of the 5th international conference on Artificial intelligence and law (pp. 201-209). ACM.

Elhag, A. A. O., Breuker, J. A. P. J., & Brouwer, P. W. (2000). On the formal analysis of normative conflicts. Information and Communication Technology Law, 9(3), 207-217.

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

López, F. (2003). Social Power and Norms: Impact on agent behaviour. Phd thesis,University of Southampton.

Kollingbaum, M. J., Norman, T. J., Preece, A., & Sleeman, D. (2007). Norm conflicts and inconsistencies in virtual organisations. In Coordination, Organizations,

Institutions, and Norms in Agent Systems II (pp. 245-258). Springer Berlin Heidelberg.

Russel, S., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach, 3rd edition. Pearson Education.

Silva, V. T. (2008). From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. Autonomous Agents and Multi-Agent Systems, 17(1), 113-155.

Vasconcelos, W. W., Kollingbaum, M. J., & Norman, T. J. (2009). Normative conflict resolution in multi-agent systems. Autonomous Agents and Multi-Agent Systems, 19(2), 124-152.

Wooldrige, M. (2009). An introduction to multiagent systems. John Wiley & Sons.