MAIRON DE ARAÚJO BELCHIOR

DETECÇÃO DE POTENCIAIS CONFLITOS NORMATIVOS QUE DEPENDEM DA ORDEM DE EXECUÇÃO DOS EVENTOS NOS SISTEMAS MULTI-AGENTES

Tese apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Doutor. Área de Concentração: ENGENHARIA DE SISTEMAS E INFORMAÇÃO.

Orientador: Profa. Dra. VIVIANE TORRES DA SILVA

Niterói

2017

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

B427 Belchior, Mairon de Araújo
Detecção de potenciais conflitos normativos que dependem da ordem de execução dos eventos nos sistemas multi-agentes / Mairon de Araújo Belchior. – Niterói, RJ : [s.n.], 2017.
137 f.

Tese (Doutorado em Computação) - Universidade Federal Fluminense, 2017.
Orientadora: Viviane Torres da Silva.

1. Sistema multiagente. 2. Inteligência artificial. 3. Agente inteligente (Software). 4. Engenharia de software. I. Título.

CDD 006.3

MAIRON DE ARAÚJO BELCHIOR

DETECÇÃO DE POTENCIAIS CONFLITOS NORMATIVOS QUE DEPENDEM DA ORDEM DE EXECUÇÃO DOS EVENTOS NOS SISTEMAS MULTI-AGENTES

Tese apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Doutor. Área de Concentração: Engenharia de Sistemas e Informação.

Aprovada em Dezembro de 2017.

BANCA EXAMINADORA

Prof. Dr. Viviane Torres da Silva, UFF – Orientador

Prof. Dr. Anselmo Antunes Montenegro, UFF

Prof. Dr. Daniel Cardoso Moraes de Oliveira, UFF

Prof. Dr. Joel Luis Carbonera, IBM

Prof. Dr. Sean Wolfgand Matsui Siqueira, UNIRIO

Niterói

2017

Este trabalho é dedicado ao meu pai Arnaldo Dias Belchior (*in memorium*)
por tudo que me proporcionou.

AGRADECIMENTOS

A minha orientadora, Profa. Viviane Torres da Silva, por todo o seu apoio, atenção e dedicação dados a este trabalho. Seu suporte foi fundamental para tornar possível este trabalho. Meus sinceros agradecimentos.

Aos professores Daniel de Oliveira, Anselmo Montenegro, Joel Carbonera e Sean Siqueira por terem aceitado prontamente participar da comissão examinadora.

Aos colegas da UFF pela amizade, especialmente, Jéssica Soares, Eduardo Silvestre, Laura Sarkis, Jean Zahn, Hedi Minin, Carol Sacramento, Leo Pio, Yona Lopes, Orlando Júnior, Rogério Avellar, Wilton Filho, Leandro Miranda, e tantos outros ...

A todos os professores e funcionários do Instituto de Computação da UFF, pelos ensinamento, atenção e auxílios.

A CAPES pelo auxílio concedido.

Aos meus pais, Arnaldo e Fátima, pelo amor, incentivo e preocupação. Obrigado por todos os ensinamentos e por terem me ensinado a ser uma pessoa melhor. Ao meu irmão, Arnaldo Segundo, e as minhas tias Luciana, Dayne, Suely e Neyla por todo apoio e carinho. Ao meu primo Ricardo e a Edian, por todo suporte e atenção recebidos no Rio de Janeiro.

A minha esposa Ingrid pelo seu carinho, companhia e amor.

Ao meu bebê Miguel, de nove meses, dom precioso recebido de Deus.

A Deus e Maria por todas as bênçãos concedidas e por mais uma conquista.

RESUMO

Normas em sistemas multi-agentes são usadas como um mecanismo para regular o comportamento de agentes autônomos e heterogêneos e para manter a ordem na sociedade de agentes. Elas prescrevem como os agentes idealmente deveriam se comportar. As normas definem ações que devem ser executadas (obrigações), ações que podem ser executadas (permissões), e ações que não podem ser executas (proibições) por determinada entidade em certa situação. Um dos desafios no projeto e gerenciamento de sistemas governados por normas é garantir que as normas especificadas não estejam em conflito. Duas normas estão em conflito quando o cumprimento de uma causa a violação da outra, e vice-versa. Tais conflitos fazem com que o agente obediente às normas fique "paralisado": não importa o que ele faça (ou deixe de fazer), ele irá violar uma das normas, ou seja, irá provocar a quebra de uma restrição social. Existem vários trabalhos na literatura que lidam com os conflitos entre normas em sistemas multi-agentes. Porém, existe uma categoria de conflitos, não investigado ainda na fase de design, que somente podem ser detectados se soubermos de alguma informação sobre a execução do sistema multi-agentes. Tais conflitos são chamados aqui de Potenciais Conflitos Normativos (PCN). Potenciais Conflitos Normativos são conflitos entre normas que *podem* acontecer durante a execução dos sistemas multi-agentes, dependendo da ordem de execução dos eventos do sistema. Este trabalho apresenta duas abordagens baseadas em cenários de execução para detectar Potenciais Conflitos Normativos em sistemas multi-agentes. Na primeira abordagem, o designer fornece ao sistema normativo um conjunto de normas e possíveis exemplos de cenários de execução do sistema. Como saída, o sistema normativo retorna os conflitos normativos que aconteceriam caso tais cenários fossem executados no sistema multi-agentes. Na segunda abordagem, o designer do sistema fornece somente um conjunto de normas e o sistema normativo gera os cenários de execução que causariam os conflitos normativos caso tais cenários aconteçam no sistema multi-agentes. Ambas as abordagens fazem a detecção do conflito entre pares de normas.

Palavras-chave: Normas, Conflito normativo, Sistemas multi-agentes, OWL, SWRL

ABSTRACT

Norms in multi-agent systems are used as a mechanism to regulate the behavior of autonomous and heterogeneous agents and to maintain the social order of the society of agents. They prescribe how agents should ideally behave. Norms describe actions that must be performed (obligations), actions that can be performed (permissions) and actions that cannot be performed (prohibitions) by a given entity in a certain situation. One of the challenges in designing and managing systems governed by norms is to deal with normative conflicts. Two norms are in conflict when the fulfillment of one causes the violation of the other, and vice-versa. Such conflicts make the norm-compliant agent "paralyzed", i.e., whatever the agent does (or refrains from doing) will lead to a social constraint being broken. Several researches have been proposed mechanisms to detect conflicts between norms. However, there is a category of normative conflicts not investigated yet in the design phase that can only be detected if we know information about the runtime execution of the system. Such conflicts are called here Potential Normative Conflicts (PNC). Potential Normative Conflicts are normative conflicts that *may* happen during the multi-agent system execution depending on execution order of runtime events of the system. This work presents two approaches, based on execution scenarios, to detect Potential Normative Conflicts in multi-agent systems. In the first approach, the system designer provides a set of norms and possible examples of system execution scenarios. As output, the normative system returns normative conflicts that would happen if such scenarios were executed in the multi-agent system. In the second approach, the system designer provides only a set of norms and the normative system generates the execution scenarios that would cause normative conflicts if such scenarios would be executed in the system. Both approaches detect normative conflicts between pair of norms.

Keywords: Norms, Normative conflict, Multi-agent systems, OWL, SWRL

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

DL – *Description Logic*;

LPO – Lógica de Primeira Ordem;

OWL – *Web Ontology language*;

PCN – Potenciais Conflitos Normativos;

PCC – *Potential Conflict Checker*;

POLAR – *Policy Language for Agent Reasoning*;

RDF – *Resource Description Framework*;

SMA – Sistemas Multi-agentes;

SPARQL – *SPARQL Protocol and RDF Query Language*;

SNU – Suposição de Nome Único;

W3C – *World Wide Web Consortium*.

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Sistemas multi-agentes (SMA) são sistemas constituídos de múltiplos agentes que interagem uns com os outros, tipicamente por meio de trocas de mensagens. Os agentes em SMA são capazes de executar ações de forma independente, ou seja, eles podem descobrir por si só o que eles precisam fazer a fim de atingir seus objetivos. Na grande maioria das vezes eles são representados ou agem em nome de um usuário. Para que os agentes possam interagir entre si, eles devem ter a capacidade de cooperar, coordenar e negociar uns com os outros (WOOLDRIDGE, 2002).

Sistemas multi-agentes abertos são sistemas multi-agentes que possuem as seguintes características: (i) os agentes podem ter diferentes arquiteturas internas e diferentes objetivos, (ii) os agentes podem falhar em obedecer às especificações do sistema a fim de atingir seus objetivos individuais, (iii) o comportamento e as interações entre os agentes não podem ser previstos antecipadamente, e (iv) os agentes são capazes de entrar ou sair do sistema a qualquer momento (ARTIKIS e PITT, 2009).

Um dos principais desafios enfrentados na pesquisa em sistemas multi-agentes é o controle social (HOLLANDER e WU, 2011). Como os sistemas multi-agentes abertos podem ser configurados e organizados dados os seus dinamismos e suas constantes mudanças em suas estruturas? De acordo com LAM *et al.* (2008a), para expressar o comportamento esperado dos agentes em seguir as restrições sociais, não devemos reduzir a autonomia dos agentes. Ao invés disso, devemos especificar normas sociais em termos do que é permitido, proibido e obrigado, e deixar os agentes decidirem o que fazer dadas tais restrições.

Sistemas multi-agentes normativos combinam normas sociais e sistemas multi-agentes. De acordo com BOELLA *et al.*, (2006), os sistemas multi-agentes normativos são sistemas multi-agentes onde os agentes são controlados por normas explicitamente representadas por um sistema normativo. Segundo MEYER e WIERINGA, (1993), sistemas normativos são quaisquer sistemas onde as normas e os conceitos normativos são requeridos para descrever e especificar o comportamento do sistema. O termo normativo significa estar de acordo ou baseado em normas.

Normas em SMA são usadas como um mecanismo para regular o comportamento de agentes autônomos e heterogêneos e para manter a ordem na sociedade de agentes. Elas prescrevem como os agentes idealmente deveriam se comportar. As normas definem ações que

devem ser executadas (obrigações), ações que podem ser executadas (permissões), e ações que não podem ser executas (proibições) por determinada entidade em certa situação. A modelagem de normas é uma importante parte da especificação do sistema e uma relevante tarefa durante o design de SMA (FIGUEIREDO *et al.*, 2011).

Porém, nem todos os agentes comportam-se de acordo com as normas. Os agentes podem decidir se irão ou não cumprir com uma norma, optando, por exemplo, por violar intencionalmente determinada norma (HOLLANDER e WU, 2011). Desta forma, os sistemas normativos devem monitorar o comportamento dos agentes e implementar sanções a eles com o objetivo de regular os seus comportamentos, evitando que violações de normas ocorram (BOELLA *et al.*, 2006, 2007). Sanções são ações que impõem penalidades aos agentes que não obedecem às normas ou recompensas a aqueles que as obedecem (HOLLANDER e WU, 2011). Na grande maioria das vezes, as sanções são também normas.

Um dos desafios no projeto e gerenciamento de sistemas governados por normas é garantir que as normas especificadas não estejam em conflito (SANTOS e DA SILVA, 2016 a). Duas normas estão em conflito quando o cumprimento de uma causa a violação da outra, e vice-versa. Por exemplo, duas normas estão em conflito quando uma está obrigando e a outra proibindo um mesmo agente a executar uma mesma ação. Tais conflitos fazem com que o agente obediente às normas fique "paralisado": não importa o que ele faça (ou deixe de fazer), ele irá violar uma das normas, ou seja, irá provocar a quebra de uma restrição social (KOLLINGBAUM *et al.*, 2008).

Existem vários trabalhos na literatura que lidam com os conflitos entre normas em SMA. Como relatado por (SANTOS e DA SILVA, 2016a), o conflito normativo pode ser classificado como *conflito direto* ou *conflito indireto*. Conflitos diretos entre duas normas ocorrem quando elas estão associadas à mesma entidade, regulam o mesmo comportamento, tem conceitos deônticos contraditórios ou contrários (ou seja, proibição *versus* permissão ou obrigação *versus* proibição) e são definidas em um mesmo contexto. A detecção deste conflito pode ser feita por uma simples comparação entre os elementos da norma (isto é, entidade, comportamento, contexto) a fim de identificar se eles são iguais nas duas normas. O outro tipo de conflito, conflitos indiretos, envolve duas normas cujos elementos não são os mesmos, mas são relacionados. Segundo DA SILVA *et al.* (2015), conflito normativo indireto é um conflito entre duas normas que (i) não necessariamente têm conceitos deônticos contraditórios, (ii) podem regular diferentes entidades (porém relacionadas) e (iii) diferentes comportamentos (porém relacionados), e (iv) estão definidas em diferentes contextos (porém relacionados). A detecção

de conflitos indiretos só pode ser feita quando os relacionamentos entre os elementos da norma são conhecidos.

1.2 DEFINIÇÃO DO PROBLEMA

Existe uma classificação de conflitos, não investigada ainda na *fase de design*, que somente podem ser detectados se soubermos de alguma informação sobre a execução do sistema multi-agentes. Tais conflitos serão chamados aqui de *Potenciais Conflitos Normativos* (PCN). Os PCN são conflitos que *podem* acontecer entre duas normas durante a execução do sistema multi-agente, dependendo da ordem de execução dos eventos do sistema referidos nas definições das normas. Os eventos aqui podem ser quaisquer situações que ocorrem em tempo de execução. Por exemplo, sejam *N1*, *N2*, *N3* e *N4* quatro normas definidas em um mesmo contexto *C* e regulando um mesmo agente *Riley*. O contexto é um elemento da norma que determina onde a mesma deve ser cumprida. Os contextos serão apresentados na Seção 2.1.1. As quatro normas estão definidas, como segue.

- *N1 obriga Riley* a executar ação *doHomework*;
- *N2 proíbe Riley* de executar ação *playGame*, *antes* dele executar *takeAShower*;
- *N3 autoriza Riley* a executar *playGame*, *depois* do cumprimento de *N1*;
- *N4 proíbe Riley* de executar *playGame*, *depois* que *Dora* executar *serveLunch* e *antes* da situação *doneLunching* tornar-se verdade para *Riley*;

Analisando as normas *N2* e *N3*, a execução da ação *takeAShower* e o cumprimento da norma *N1* são eventos que ocorrem em tempo de execução e, durante a especificação das normas, não sabemos quando que estes eventos serão executados pelos agentes do sistema. Porém, observando a ordem de execução destes eventos, podemos afirmar com certeza que se a execução da ação *takeAShower* acontecer primeiro em relação ao cumprimento da norma *N1*, as normas *N2* e *N3 não* estarão em conflito (Figura 1.b). Porém, se o oposto acontecer, existirá um conflito normativo entre *N2* e *N3* (Figura 1.a). O agente *Riley* estará permitido e proibido de executar a ação *playGame* em intervalos de tempo que interceptam.

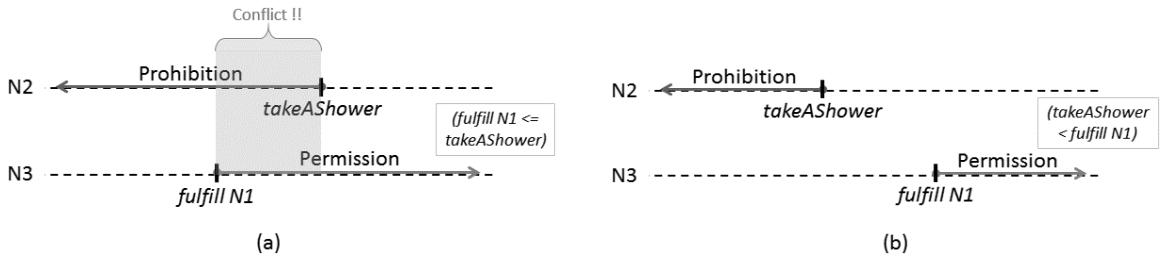


Figura 1: Potencial conflito normativo entre uma proibição com uma condição *antes* e uma permissão com uma condição *depois* (BELCHIOR e DA SILVA, 2017c).

A possibilidade de conflito pode ser observada também entre as normas *N3* e *N4*. Se a ação *serveLunch* e a situação *doneLunching* acontecerem antes do cumprimento da norma *N1*, então pode-se afirmar seguramente que as normas *N3* e *N4 não* estarão em conflito normativo (Figura 2.c). No entanto, caso a execução destes eventos ocorram de qualquer outra forma, tais normas estarão em conflito normativo (Figura 2.a e Figura 2.b).

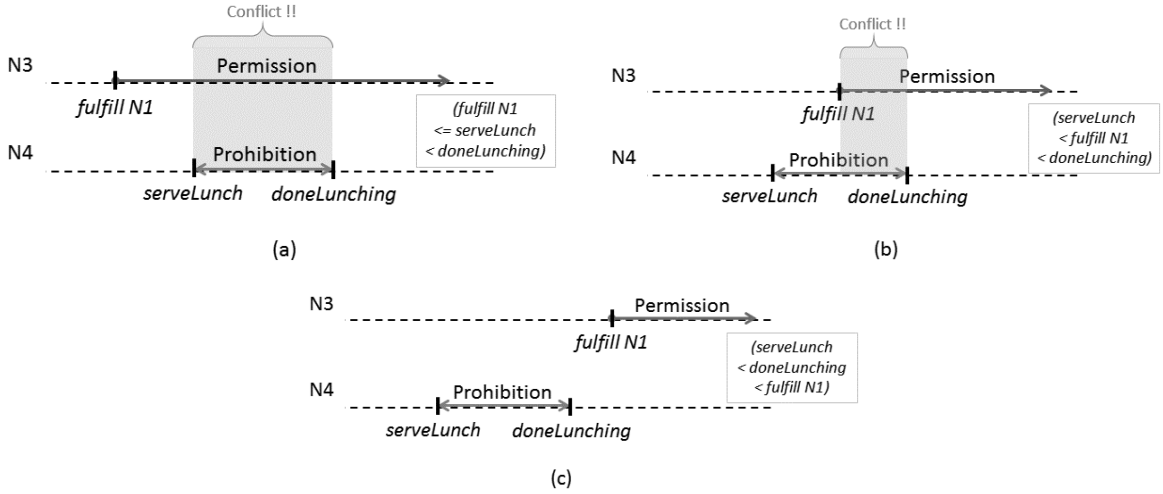


Figura 2: Potencial conflito normativo entre uma permissão com uma condição *depois* e uma proibição com uma condição *depois* e *antes* de um evento.

Deste modo, se tivermos informação sobre o momento de quando as condições que definem os períodos de ativação das normas ocorrerão no sistema, isto é, o momento que os eventos ocorrerão no sistema, seria possível fazer a detecção destes conflitos. Os eventos considerados neste trabalho podem ser a (i) execução de uma ação por um agente, a (ii) ativação ou (iii) desativação de uma norma, o (iv) cumprimento ou a (v) violação de uma norma ou (vi) um fato no sistema que tenha se tornado verdade para um agente.

1.3 OBJETIVOS

1.3.1 GERAIS

O objetivo principal desta tese visa propor métodos para a detecção de uma nova classificação de conflitos normativos, não investigada ainda na literatura, chamados aqui de *Potenciais Conflitos Normativos* (PCN). Os PCN são conflitos que *podem* acontecer entre duas normas durante a execução de um sistema multi-agentes dependendo da ordem de execução dos eventos do sistema referidos nas definições de tais normas. Tais conflitos só podem ser verificados quando levamos em consideração a ordem de ocorrência dos eventos do sistema multi-agentes. Os métodos de detecção propostos nesta tese ocorrem *tempo de design*.

Foram propostas duas abordagens, ambas baseadas em cenários de execução do sistema, para detectar os *Potenciais Conflitos Normativos*. Os cenários de execução representam certas situações que *podem* acontecer com os agentes durante a execução do sistema multi-agente, tais como a execução de uma ação ou fatos que tenham se tornado verdade para um agente. Exemplos de cenários de execução podem ser: o agente *A1* abriu a porta da sala no tempo 5, ligou o ar-condicionado no tempo 6, passou uma atividade para os alunos no tempo 20, o conteúdo da aula foi concluído no tempo 30, etc.

A primeira abordagem proposta neste trabalho permite que o designer do sistema forneça possíveis exemplos de cenários de execução do sistema e um conjunto de normas, e avalie se tais cenários causariam algum potencial conflito normativo caso a execução do sistema multi-agentes acontecesse tal como o cenário de execução passado pelo designer. Como saída, o sistema normativo retorna os potenciais conflitos normativos, resultado da aplicação dos cenários de execução, e uma justificativa explicando a causa de cada conflito.

Na segunda abordagem de detecção de conflitos, o designer do sistema fornece apenas um conjunto de normas e o sistema normativo gera para o designer do sistema os cenários de execução e faz a detecção dos conflitos normativos entre pares de normas no sistema para cada cenário de execução gerado. A saída da segunda abordagem são todos os potenciais conflitos normativos entre duas normas que ocorreriam em um determinado cenário de execução. Os cenários de execução da segunda abordagem são sequências ordenadas dos eventos mencionados nas normas que causariam o conflito caso tais eventos fossem executados nessa ordem no sistema multi-agentes. As duas abordagens fazem as detecções do conflito entre pares de normas.

Toda a estrutura normativa desenvolvida neste trabalho foi representada usando a linguagem OWL DL e regras em *Semantic Web Rule Language* (SWRL). A linguagem OWL (*Web Ontology language*) é uma expressiva linguagem de representação do conhecimento, padronizada pela *World Wide Web Consortium* (*W3C*). Ela é usada para definir formalmente um conjunto comum de termos que são usados para descrever e representar um certo domínio. Uma das vantagens de usar OWL DL é que ela é uma linguagem bastante expressiva e há ferramentas de inferências para OWL DL eficientes, livres e amplamente usadas, tais como, HermiT[1] e Pellet[2]. Além disso, inferências em OWL DL garantem que todas as conclusões sejam computáveis (*completeness*) e que todos os cálculos terminem em tempo finito

[1] http://hermit-reasoner.com
[2] http://clarkparsia.com/pellet

(*decidability*) (SMITH *et al.*, 2004). A *Semantic Web Rule Language*[3] é uma linguagem de regras baseada na OWL DL (HORROCKS *et al.*, 2004). SWRL estende os axiomas OWL para incluir regras na forma de *Cláusulas Horn*. Ela fornece recursos de raciocínio dedutivo mais poderosos do que OWL (HEBELER *et al.*, 2009). Uma das vantagens de se usar SWRL é o seu suporte a funções incorporadas para executar operações de comparações, matemática, *strings*, data e outros.

1.3.2 ESPECÍFICOS

Este trabalho tem os seguintes objetos específicos:

- Especificação da estrutura normativa usando a linguagem OWL DL para representar os principais conceitos de uma norma em um sistema multi-agente;
- Especificação de regras SWRL para determinar os tempos referentes à execução de uma ação por um agente, ativação ou desativação de uma norma, cumprimento ou violação de uma norma e fatos no sistema que tenham se tornado verdade para um agente;
- Especificação de regras SWRL para fazer a verificação dos potenciais conflitos normativos entre uma obrigação e proibição e entre uma permissão e proibição;
- Implementação de uma ferramenta chamada *Potential Conflict Checker* para dar suporte à verificação dos potenciais conflitos normativos em um sistema multi-agentes. A ferramenta deve fornecer ao usuário um ambiente de criação das normas, dos cenários de execução e apresentar a detecção dos conflitos de acordo com as duas abordagens descritas neste trabalho;
- Demonstração das abordagens de detecção dos potenciais conflitos normativos por meio de um estudo de caso;
- Realização da validação da detecção dos potenciais conflitos normativos por meio de especialistas na área de normas em SMA.

1.4 ORGANIZAÇÃO DA TESE

Esta tese foi dividida em sete capítulos, incluindo esta introdução, e dois apêndices, descritos, resumidamente, a seguir:

No capítulo 2, *Fundamentação Teórica*, são apresentados brevemente os principais fundamentos teóricos necessários para o desenvolvimento desta tese.

[3] http://www.w3.org/Submission/SWRL/

No capítulo 3, *Mecanismo para Detecção dos PCN*, é apresentada a especificação da estrutura normativa, dividida em duas ontologias OWL DL: *ontologia de Norma* e *ontologia de Cenário de Execução*. Em seguida, as aplicações das regras SWRL para inferir os tempos das condições que definem os intervalos de ativação das normas e as duas abordagens para a detecção dos PCN entre pares de normas são apresentadas. Além disso, uma ferramenta para a verificação dos potenciais conflitos normativos, chamada *Potential Conflict Checker*, e as tarefas que podem ser executas nela também são descritas neste capítulo.

No capítulo 4, *Estudo de Caso e Análise dos Resultados*, é apresentado um estudo de caso para demonstrar as duas abordagens de detecção dos potenciais conflitos normativos. Além disso, são apresentadas as avaliações dos especialistas na área de normas para validar os resultados da detecção dos potenciais conflitos normativos entre pares de normas.

No capítulo 5, *Trabalhos Relacionados*, são apresentados os trabalhos relacionados e uma análise comparativa com a abordagem descrita na tese.

No capítulo 6, *Conclusão e Trabalhos Futuros*, são apresentadas as contribuições geradas com o desenvolvimento desta tese e sugestões para trabalhos futuros.

O Apêndice A contém o *Formulário de Avaliação de Conflitos Normativos pelo Especialista*, e o Apêndice B apresenta as ontologias OWL DL de Normas e de Cenário de Execução completas, descritas neste trabalho, exportadas a partir da ferramenta de edição de ontologias, chamada Protégé, versão 5.0.0, usando o formato *Turtle*.

CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais fundamentos teóricos necessários para o desenvolvimento desta tese, tais como explanações teóricas sobre normas em SMA, conflitos normativos, lógica de descrição para representar as ontologias OWL DL e regras em SWRL.

2.1 NORMAS EM SISTEMAS MUTI-AGENTES

Sistemas multi-agentes são sistemas formados por múltiplos elementos de computação iterativos, chamados de *agentes*. Os agentes são sistemas de software que possuem duas habilidades importantes. Primeiro, os agentes são capazes de realizar ações de forma autônoma, ou seja, decidir por si mesmos o que eles precisam fazer para satisfazer seus objetivos. A segunda habilidade que os agentes possuem é a capacidade de interagir com outros agentes, não somente através de troca de dados, mas também realizando atividades sociais como cooperação, coordenação, negociação, dentre outras atividades similares (WOOLDRIDGE, 2002).

Devido aos SMA serem compostos de múltiplos agentes individuais que interagem uns com os outros, muitos paralelos com sociedades humanas podem ser formulados. Um dos principais desafios enfrentados na pesquisa em SMA é o *controle social* (HOLLANDER e WU, 2011). Como os SMA podem ser configurados e organizados dado os seus dinamismos e suas constantes mudanças em suas estruturas? De acordo com LAM *et al.* (2008a), para expressar o comportamento esperado dos agentes em seguir as restrições sociais, não devemos reduzir a autonomia dos agentes. Ao invés disso, devemos especificar *normas sociais* em termos do que é permitido, proibido e obrigado, e deixar os agentes decidirem o que fazer dadas tais restrições. As *normas sociais* atuam como restrições comportamentais que dependem do domínio regulando e estruturando a ordem social dentro de um sistema multi-agentes (BOELLA e TORRE, 2007). Nas sociedades humanas, as normas sociais são essenciais para regulação, coordenação e cooperação. Seguindo o mesmo pensamento, estes mesmos princípios podem ser aplicados a sociedades de agentes, dos quais os SMA são um tipo (HOLLANDER e WU, 2011).

Sistemas multi-agentes normativos combinam normas sociais e sistemas multi-agentes. De acordo com BOELLA *et al.*, (2006), os sistemas multi-agentes normativos são sistemas multi-agentes onde os agentes são controlados por normas explicitamente representadas por um sistema normativo. As normas são usadas para restringir o comportamento dos agentes e então reduzir o tamanho do espaço de busca dos agentes (HOLLANDER e WU, 2011). Segundo

MEYER e WIERINGA, (1993), sistemas normativos são quaisquer sistemas onde as normas e os conceitos normativos são requeridos para descrever e especificar o comportamento do sistema. O termo normativo significa estar de acordo ou baseado em normas.

As normas em SMA são usadas como um mecanismo para regular o comportamento de agentes autônomos e heterogêneos e para manter a ordem social da sociedade de agentes. Elas prescrevem como os agentes idealmente deveriam se comportar. As normas definem ações que devem ser executadas (obrigações), ações que podem ser executadas (permissões), e ações que não podem ser executas (proibições) por determinada entidade em certa situação (FIGUEIREDO *et al*., 2011).

É importante perceber que, devido à característica autônoma do agente nos SMA, é possível que seu comportamento possa se desviar do ideal, permitindo que violações de suas obrigações ocorram. Portanto, nem todos os agentes comportam-se de acordo com as normas, ou seja, os agentes podem violar intencionalmente uma norma e decidir se irão ou não cumpri-las (HOLLANDER e WU, 2011). Como os agentes são entidades orientadas a objetivo, então, dependendo dos benefícios que o cumprimento de uma norma possa trazer a eles para atingir os seus objetivos, tais agentes podem decidir cumprir ou violar a norma (DOS SANTOS NETO *et al.*, 2013).

Para fazer com que os agentes cumpram com suas obrigações, sistemas normativos podem aplicar incentivos (ou premiações) e sanções. As sanções têm o objetivo de punir aqueles agentes que cometem violações e desencorajar que violações sejam cometidas e os incentivos têm o objetivo de recompensar aqueles agentes que cumprem com suas obrigações (HOLLANDER e WU, 2011). Por exemplo, quando um agente viola uma norma de um contrato, ele pode ter que pagar uma penalidade. Por outro lado, um agente pode ser premiado quando ele tem um bom comportamento, isto é, quando ele cumpre com suas obrigações. Na grande maioria das vezes, as sanções e os incentivos são também representados por normas do sistema. Os sistemas normativos são os responsáveis por monitorar o comportamento dos agentes e implementar as sanções e incentivos a eles (BOELLA *et al.*, 2006, 2007).

2.1.1 ELEMENTOS QUE COMPÕEM UMA NORMA

Uma norma explicitamente estabelece uma proibição, uma obrigação ou uma permissão sobre o comportamento de uma entidade em um dado contexto. Em outras palavras, uma norma está sempre associada a um *conceito deôntico*, regula um determinado *comportamento* e é dirigida a uma *entidade* (SANTOS *et al.*, 2017). Segundo os autores, uma norma em SMA

especifica o que está sendo regulado (ou seja, a execução de uma ação), quando que o regulamento ocorre (ou seja, o período de tempo que a norma fica ativa) e quem é o sujeito deste regulamento (ou seja, o agente, um papel ou um grupo).

FIGUEIREDO *et al.* (2011) investigaram dez linguagens de modelagem e metodologias usadas para descrever e implementar normas em SMA e concluíram que os *principais* conceitos que compõem uma norma são: *conceito deôntico*, *entidades*, *ações*, *restrições de ativação/desativação*, *sanções* (e *premiações*) e *contexto*. Tais conceitos são descritos brevemente a seguir.

Conceito deôntico. O *conceito deôntico* é uma restrição comportamental para os agentes na forma de obrigação, permissão e proibição. Normas cujo conceito deôntico é uma proibição proíbem o agente de realizar certas ações. De forma contrária, normas cujo conceito deôntico é uma obrigação descrevem as atividades que deverão ser realizadas pelos agentes, e as normas onde conceito deôntico é uma permissão descrevem quais ações são autorizadas de serem executadas pelos agentes. Tais conceitos estão relacionados uns com os outros da seguinte forma. Se uma ação é obrigada, então ela também é permitida, mas não pode ser simultaneamente proibida. Se alguma ação é proibida, então ela não pode ser, a mesmo tempo, nem permitida e nem obrigada. Se uma ação é permitida, então não necessariamente ela é obrigada, mas com certeza não pode ser proibida de ser executada. De alguma outra forma, um conflito irá acontecer.

Entidades envolvidas. As *entidades* representam sujeitos sendo regulados por uma norma. As normas regulam o comportamento das *entidades*. As entidades podem ser um agente, um grupo de agentes (organização), ou um papel (SANTOS *et al.*, 2017). Quando uma norma regula o comportamento de um grupo de agentes, somente agentes que pertencem a esse grupo são o sujeito do regulamento. Quando a norma é aplicada a um papel, somente aqueles agentes que desempenham tal papel são os sujeitos do regulamento. Neste trabalho, as entidades representadas são os próprios agentes.

Ações. As *ações* representam o comportamento sendo controlado por uma norma que pode, deve ou não pode ser executado por uma entidade, dependendo do conceito deôntico da norma. As ações podem ser atômicas ou ações complexas, ou seja, ações parametrizadas ou compostas (SANTOS *et al.*, 2017). Por simplicidade, as ações consideradas neste trabalho são ações atômicas.

Contexto. Uma norma pode estar associada também a um *contexto*. O *contexto* determina a área de aplicação da norma. Ele define o escopo onde a norma é definida (SANTOS

e DA SILVA, 2016b). Portanto, o contexto de uma norma determina onde a norma deve ser cumprida. As normas são geralmente definidas no contexto de uma *organização* ou *ambiente*. Uma norma definida em certo contexto deve ser cumprida somente por aqueles agentes que executam neste contexto. Fora do contexto, a norma não é aplicável. Portanto, um agente sendo regulado por uma norma pode pertencer a uma organização ou habitar um ambiente. Neste trabalho, outros contextos poderão também ser implementados.

Restrição de ativação/desativação. A *restrição de ativação* (ou condição de ativação) determina o momento a partir do qual uma norma será ativada enquanto que a *restrição de desativação* (ou condição de desativação) define quando a norma irá ser desativada. A norma fica ativa quando a condição de ativação é satisfeita e ela torna-se inativa quando a condição de desativação é satisfeita. As condições de ativação/desativação podem ser uma data, um evento, ou um estado (SANTOS *et al.*, 2017). Se uma norma não tem condição de ativação e nem de desativação, então a norma é sempre aplicável.

Neste trabalho, as condições de ativação e desativação da norma foram chamadas simplesmente de *condições* e são responsáveis por definir o período no qual a norma fica ativa. Além disso, tais condições representadas neste trabalho são eventos que acontecem em tempo de execução, tais como, execução de uma ação, ativação ou desativação de uma norma, cumprimento ou violação de uma norma, ou um fato na base de dados.

Sanções/premiações. Por fim, as *sanções* são punições que podem ser aplicadas a àqueles agentes que violam uma determinada norma e as *premiações* são recompensas a àqueles que cumprem uma norma (DA SILVA, 2008). Sanções e premiações são usadas como incentivadores para se manter o controle nos SMA. Na grande maioria das vezes, as sanções/premiações são representadas por uma norma (HOLLANDER e WU, 2011). Neste trabalho, as sanções/premiações não foram representadas como elementos de uma norma.

Para ilustrar os conceitos apresentados anteriormente, o seguinte exemplo é apresentado. Vamos supor que *N1* seja uma norma que define o seguinte: “No hospital Central, todos os visitantes são proibidos de fumar enquanto estejam dentro do hospital, ou seja, depois que entram no hospital e antes de saírem”. Neste exemplo, o termo “hospital Central” é o *contexto* onde a norma deve ser cumprida, “visitantes” é a *entidade* cujo comportamento está sendo regulado, o *conceito deôntico* da norma é uma proibição, a *ação* sendo controlada é “fumar” e as *condições* que determinam o período de ativação da norma é “enquanto estejam dentro do hospital, ou seja, depois que entram no hospital e antes de saírem”.

2.1.2 CONFLITOS NORMATIVOS

Um conflito normativo acontece quando o cumprimento de uma norma causa a violação de outra norma, e vice-versa. Por exemplo, vamos supor que *N1* e *N2* sejam duas normas. Existe um conflito normativo entre *N1* e *N2* quando a norma *N1* proíbe um agente de executar uma ação particular e a norma *N2* obriga o mesmo agente a executar a mesma ação em um mesmo período de tempo. Situações de conflito como a do exemplo fazem com que o agente fique em uma posição na qual não importa o que ele faça (ou deixe de fazer) irá resultar em uma restrição social sendo quebrada (KOLLINGBAUM *et al.*, 2008). Em outras palavras, não importa o que o agente faça, vai acabar violando uma das normas, *N1* ou *N2*.

Os conflitos normativos nos SMA podem ser classificados como *conflitos diretos* e *conflitos indiretos* (SANTOS e DA SILVA, 2016a). Os conflitos normativos são chamados de *conflitos diretos* quando o conflito pode ser detectado através de uma comparação direta entre componentes das normas. A comparação é feita da seguinte forma. Duas normas estão em conflito normativo direto quando

1. Os *conceitos deônticos* das normas são *contraditórios* ou *contrários* (isto é, obrigação *versus* proibição ou permissão *versus* proibição);
2. As normas estão regulando uma *mesma entidade*;
3. As normas estão regulando um *mesmo comportamento*; e
4. As normas estão definidas em um *mesmo contexto*.

Por exemplo, sejam *N1* e *N2* duas normas definidas como segue:

- *N1 obriga* agente *John* a executar ação *comprar* no contexto da *loja Saraiva*.
- *N2 proíbe* agente *John* a executar ação *comprar* no contexto da *loja Saraiva*.

Analisando as normas *N1* e *N2*, pode-se observar que elas estão em conflito normativo direto, pois, fazendo uma comparação direta entre os elementos das duas normas, obtém-se o seguinte. As normas *N1* e *N2* têm conceitos deônticos contraditórios (obrigação vs. proibição), regulam uma mesma entidade (“John”) e um mesmo comportamento (“comprar”) e estão definidas em um mesmo contexto (“loja Saraiva”).

Por outro lado, um conflito normativo é chamado de *conflito indireto* quando o conflito só pode ser detectado quando se conhece alguma característica do domínio. Mas especificamente, a detecção de conflitos indiretos requer que se conheça os relacionamentos entre os elementos das normas. Segundo DA SILVA *et al.* (2015), conflito normativo indireto é um conflito entre duas normas que (i) não necessariamente têm conceitos deônticos contraditórios, (ii) podem regular diferentes entidades (porém relacionadas) e (iii) diferentes

comportamentos (porém relacionados), e (iv) podem estar definidas em diferentes contextos (porém relacionados).

Por exemplo, sejam *N3* e *N4* duas normas definidas como segue:

- *N3 obriga* agente *John* a executar ação *comprar* no contexto do *Rio de Janeiro*.
- *N4 proíbe* agente *John* a executar ação *comprar* no contexto da *Brasil*.

E suponhamos que temos a seguinte informação do domínio da aplicação.

- Rio de Janeiro é uma cidade do Brasil.

Analisando as normas *N3* e *N4*, pode-se observar que elas regulam uma mesma entidade ("John") e um mesmo comportamento ("comprar"), têm conceitos deônticos contraditórios (obrigação vs. proibição), porém elas estão definidas em contextos diferentes ("Rio de Janeiro" e "Brasil"). Uma obriga a execução de uma ação dentro da cidade do Rio de Janeiro e a outra proíbe a mesma execução no contexto do Brasil. Analisando a informação do domínio da aplicação, pode-se observar que as normas *N3* e *N4* estão em conflito normativo indireto, pois os contextos destas normas estão relacionados através de um relacionamento de partonomia ("parte de"), ou seja, Rio de Janeiro é parte do Brasil.

Agora, sejam *N5* e *N6* duas normas definidas como segue:

- *N5 obriga* agente *John* a executar ação *concordar* no contexto *casa*.
- *N6 obriga* agente *John* a executar ação *discordar* no contexto *casa*.

E suponhamos que temos a seguinte informação do domínio da aplicação.

- Ação *concordar* é ortogonal a ação *discordar*.

Analisando as normas *N5* e *N6*, pode-se perceber que elas regulam uma mesma entidade ("John") e estão definidas em um mesmo contexto ("casa"), porém regulam ações diferentes ("concordar" e "discordar") e não têm conceitos deônticos contraditórios (obrigação vs. obrigação). Analisando a informação do domínio da aplicação, existe um relacionamento de ortogonalidade entre as ações *concordar* e *discordar* e, portanto, não podem ser executadas ao mesmo tempo (DA SILVA *et al.*, 2015). Portanto, as normas *N5* e *N6* estão em conflito normativo indireto.

2.2 OWL – *WEB ONTOLOGY LANGUAGE*

A OWL, *WEB ONTOLOGY LANGUAGE*, é uma linguagem para especificação de ontologias recomendada pela *World Wide Web Consortium* (*W3C*). Ela pode ser usada para definir formalmente um conjunto de conceitos que são usados para descrever e representar um certo domínio. A linguagem OWL é baseada em um modelo lógico que permite a definição e

descrição de conceitos. O modelo lógico permite o uso de máquinas de inferências que verificam se todas as sentenças e definições na ontologia estão consistentes e, além disso, reconhecem quais conceitos se enquadram em quais definições, ou seja, automaticamente computam a hierarquia de classes da ontologia (HORRIDGE e BRANDT, 2011).

A linguagem OWL pode ser categorizada em três sub-linguagens:

- OWL Lite
- OWL DL
- OWL Full

A linguagem OWL Lite é a menos expressiva da OWL. Ela suporta a classificação de hierarquias de classes e restrições simplificadas da OWL (SMITH *et al.*, 2004). Por exemplo, na OWL Lite, a restrição de cardinalidade é permitida somente valores 0 e 1 para a cardinalidade, não é possível definir uma classe pela enumeração de seus membros, uniões e complementos não podem ser usadas para descrever classes, disjunções de classes não podem ser especificadas em OWL Lite, entre outras limitações. (SYNAK *et al.*, 2009).

A linguagem OWL DL, onde DL é o acrônimo para Lógica de Descrição, é uma linguagem baseada na Lógica de Descrição, que é um fragmento decidível da Lógica de Primeira Ordem (RUDOLPH, 2011). OWL DL é uma linguagem bastante expressiva e inclui todas as construções da linguagem OWL com algumas restrições, tal como a separação de tipo (uma classe não pode ser também um indivíduo ou uma propriedade, e uma propriedade não pode ser também um indivíduo ou uma classe). Além disso, OWL DL garante que todas as conclusões sejam computáveis (*completeness*) e que todos os cálculos terminem em tempo finito (*decidability*) (SMITH *et al.*, 2004). Para a linguagem OWL DL, existem ferramentas de inferências eficientes, livres e amplamente usadas, tais como, HermiT (SHEARER *et al.*, 2008) e Pellet (SIRIN *et al.*, 2007).

A linguagem OWL Full suporta aqueles usuários que queiram o máximo de expressividade e independência, porém sem garantia computacional (SMITH *et al.*, 2004). Por exemplo, ontologias em OWL Full permitem que uma classe seja tratada como uma coleção de indivíduos e como um indivíduo, ao mesmo tempo. OWL Full não garante que todos os cálculos terminem em tempo finito e que todas as conclusões sejam computáveis, além de não ser possível executar máquinas de inferências para ontologias OWL Full (HORRIDGE *et al.*, 2004). Toda ontologia OWL DL é uma ontologia OWL Full válida, mas o inverso não é verdade.

As ontologias desenvolvidas neste trabalho estão escritas através da linguagem OWL DL, devido a sua correspondência com a Lógica de Descrição, garantindo que ela tenha completude e decidibilidade computacional, além de um máquinas de inferências para OWL DL eficientes, livres e amplamente usadas.

2.2.1 COMPONENTES DA LINGUAGEM OWL

Uma ontologia OWL consiste de indivíduos, propriedades e classes (HORRIDGE e BRANDT, 2011). Os indivíduos representam objetos em um domínio de interesse. Os indivíduos também podem ser referidos como instâncias de classe. A linguagem OWL não usa a *Suposição de Nome Único* (SNU), em inglês *Unique Name Assumption*, ou seja, dois nomes diferentes para indivíduos não necessariamente significam que eles se referem a indivíduos diferentes, eles *podem* se referir a um mesmo indivíduo. Por exemplo, "UFF" e "Universidade Federal Fluminense" *podem* referir-se a um mesmo indivíduo. Portanto, em OWL, é obrigatório declarar explicitamente que indivíduos são os mesmos que outros (através da propriedade *owl:sameAs*), ou diferentes (através das propriedades *owl:differentFrom* ou *owl:AllDifferent*).

As propriedades em OWL são relacionamentos binários e permitem especificar fatos sobre os indivíduos. Por exemplo, a propriedade *isChildOf* pode relacionar dois indivíduos da classe *Persson*. A linguagem OWL distingue dois tipos de propriedades: propriedades do tipo objeto (*owl:ObjectProperty*), no qual descreve relacionamentos binários entre dois indivíduos, e propriedades do tipo *datatype* (*owl:DatatypeProperty*), que representa um relacionamento binário entre um indivíduo e um valor de dado obtido a partir do *XML Schema datatypes* ou um LITERAL definido em RDF.

As propriedades na linguagem OWL podem também ter algumas características (HORRIDGE e BRANDT, 2011). Elas podem ser transitivas (*owl:TransitiveProperty*), simétricas (*owl:SymmetricProperty*), funcionais (*owl:FunctionalProperty*), reflexivas (*owl:ReflexiveProperty*), inversas (*owl:inverseOf*), entre outras características. Se *P* é uma propriedade *transitiva*, e ela relaciona indivíduo *x* com indivíduo *y*, e também indivíduo *y* com indivíduo *z*, então pode-se inferir que indivíduo *x* está relacionado com indivíduo *z* via propriedade *P*, ou seja, *P(x,y)* e *P(y,z)* implica em *P(x,z)*, para qualquer *x, y* e *z*. Se a propriedade *P* é *simétrica*, e ela relaciona indivíduo *x* com indivíduo *y*, então indivíduo *y* é também relacionado com indivíduo *x* através da mesma propriedade *P*, ou seja, *P(x,y)* sse *P(y,x)*, para qualquer *x* e *y*. Se *P* é uma propriedade *funcional*, para um dado indivíduo, deve existir no máximo um indivíduo relacionado a ele via propriedade *P*, ou seja, para todo *x, y* e *z*, P(x, y) e

P(x, z) implica em *y = z*. Uma propriedade *P* é dita *reflexiva* quando *P* deve relacionar um indivíduo *x* a ele mesmo. Se uma propriedade *P1* é declarada como *inversa* da propriedade *P2*, então, para todo *x* e *y*, *P1(x,y)* sse *P2(y, x)*.

A linguagem OWL também permite a criação de restrições nas propriedades (SMITH *et al.*, 2004). As restrições nas propriedades são uma forma especial de descrição de classe. Elas descrevem classes anônimas de indivíduos que satisfazem as restrições. As restrições nas propriedades podem ser de valores (*owl:allValuesFrom*, *owl:someValuesFrom*, *owl:hasValue*) ou de cardinalidade (*owl:maxCardinality*, *owl:minCardinality*, *owl:cardinality*). As restrições de valor restringem o valor de uma propriedade enquanto que as restrições de cardinalidade restringem o número de valores que uma propriedade pode ter.

A restrição *owl:allValuesFrom* requer que, para cada instância da classe que tenha uma propriedade *P* com essa restrição, os valores dessa propriedade devem ser *todos* membros da classe indicada pela cláusula *owl:allValuesFrom*. Essa restrição é análoga ao quantificador universal (∀) da lógica de predicados, ou seja, ela requer que todos os valores de uma propriedade *P* sejam de um tipo *T*, mas que se não existir valores, tal restrição é igualmente verdadeira. A restrição *owl:someValuesFrom* requer que, para cada instância da classe que tenha uma propriedade *P* com essa restrição, exista *pelo menos* um valor dessa propriedade que seja membro da classe indicada pela cláusula *owl:someValuesFrom*. A restrição de propriedade *owl:someValuesFrom* é análoga ao quantificador existencial (∃) da lógica de predicados. A restrição *owl:hasValue* permite especificar classes baseadas na existência de valores particulares nas propriedades. Portanto, um indivíduo será membro de tal classe sempre que pelo menos um dos valores da propriedade com esta restrição seja igual ao valor especificado na cláusula *owl:hasValue*.

A restrição de cardinalidade *owl:maxCardinality* restringe a quantidade máxima de valores semanticamente distintos que uma propriedade com esta restrição pode ter. A restrição *owl:minCardinality* especifica o número mínimo de valores semanticamente distintos que uma propriedade com esta restrição pode ter. A restrição *owl:cardinality* permite especificar o exato número de elementos que uma propriedade com essa restrição pode ter.

Por fim, as *classes* em OWL proveem um mecanismo de abstração para agrupar recursos com características similares, ou seja, uma classe define conjuntos de indivíduos que compartilham propriedades comuns (DEAN *et al.*, 2004). Elas são descritas usando descrições formais que declaram precisamente os requerimentos para os membros da classe (HORRIDGE e BRANDT, 2011). Existem duas classes OWL predefinidas, chamadas *owl:Thing* e

owl:Nothing. A classe *owl:Thing* é o conjunto de todos os indivíduos e a classe *owl:Nothing* é o conjunto vazio. Consequentemente, toda classe OWL é uma subclasse da classe *owl:Thing* e *owl:Nothing* é uma subclasse de todas as classes.

2.2.2 LÓGICA DE DESCRIÇÃO

Logicas de Descrição (DLs) são uma família de linguagens para representação do conhecimento que são amplamente usadas na modelagem de ontologias (KRÖTZSCH *et al.*, 2012). A Lógica de Descrição é uma lógica, fragmento decidível da LPO, equipada com uma semântica formal, ou seja, uma precisa especificação do significado de ontologias baseadas em DL. Esta semântica formal permite que os seres humanos e os sistemas de computador troquem ontologias baseadas em DL sem ambiguidade quanto ao seu significado, e também possibilita o uso de dedução lógica para inferir informações adicionais a partir de fatos explicitamente declarados em uma ontologia – uma importante característica que distingue a Lógica de Descrição de outras linguagens de modelagem, tal como a UML (KRÖTZSCH *et al.*, 2014).

Como mencionado na Seção 2.2, a linguagem OWL DL é baseada na Lógica de Descrição. Portanto, uma ontologia OWL DL pode ser representada por uma base de conhecimento *KB* da Lógica de Descrição. As classes e propriedades de uma ontologia OWL DL são chamadas de *conceitos* e *papéis* na literatura da DL. A Lógica de Descrição na qual a OWL DL é baseada chama-se $\mathcal{SHOIN}(\mathcal{D})$ (HORROCKS *et al.*, 2007).

Uma base de conhecimento *KB* da Lógica de Descrição é composta por três partes: *Terminological Knowledge* (*TBox*), *Assertional Knowledge* (*ABox*) e *Relational Knowledge* (*RBox*) (KRÖTZSCH *et al.*, 2014). O *TBox* contém um conjunto de axiomas assertivos que definem os conceitos do domínio. O *ABox* compreende de um conjunto de declarações sobre os indivíduos de um domínio. O *RBox* engloba um conjunto de axiomas assertivos sobre os *papéis* do domínio. Os axiomas TBox, ABox e RBox são descritos nas próximas subseções.

2.2.2.1 ASSERTIONAL KNOWLEDGE (ABOX)

Os axiomas de ABox capturam o conhecimento sobre os indivíduos, ou seja, os conceitos os quais eles pertencem (axiomas *concept assertions*) e como eles estão relacionados uns com os outros (axiomas *role assertions*) (KRÖTZSCH *et al.*, 2014). Por exemplo,

```
Child(Riley)
```

declara que *Riley* é uma criança ou, mais precisamente, que o indivíduo chamado Riley é uma instância do conceito *Child*. Os axiomas *role assertions* descrevem relacionamentos entre indivíduos. Por exemplo, a sentença

```
isMotherOf(Susan, Riley)
```

declara que *Susan* é a mãe de *Riley* ou, mais precisamente, que o indivíduo chamado *Susan* está em uma relação representada por *isMotherOf* com o indivíduo chamado *Riley*.

Assim como OWL DL, a Lógica de Descrição não usa a *Suposição de Nome Único* (SNU), em inglês *Unique Name Assumption*, ou seja, diferentes nomes para indivíduos *podem* referirem-se ao mesmo indivíduo, a menos que explicitamente declarado o contrário (KRÖTZSCH *et al.*, 2012). Por exemplo, a seguinte declaração

```
Susan ≠ Riley
```

é usada para declarar que os indivíduos *Susan* e *Riley* são de fato diferentes entre si. Por outro lado, a declaração, tal como

```
Susan ≈ Su
```

declara que os indivíduos *Susan* e *Su* referem-se ao mesmo indivíduo.

2.2.2.2 TERMINOLOGICAL KNOWLEDGE (TBOX)

Os axiomas de TBox descrevem relacionamentos entre conceitos. Por exemplo, o fato de que todos os atores são artistas é expresso pelo axioma *concept inclusion*, como segue.

```
Actor ⊑ Artist
```

O axioma *Concept inclusion* permite criar hierarquia de conceitos. Portanto, no exemplo acima pode-se dizer que o conceito *Actor* é um sub-conceito do conceito *Artist*. Outra forma de descrever relacionamentos entre conceitos é através do axioma *concept equivalence*, no qual permite declarar que dois conceitos têm as mesmas instâncias. Por exemplo,

```
Person ≡ Human
```

declara que um *Person* é um *Human* e vice-versa, ou seja, toda instância de *Person* é também instância de *Human*.

A Lógica de Descrição também permite que conceitos sejam construídos usando expressões mais complexas, chamados de *conceitos complexos*, como serão mostrados a seguir.

Conceitos complexos podem ser construídos, por exemplo, usando operadores booleanos, semelhantes aos operadores de *interseção*, *união* e *complemento* dos conjuntos ou aos operadores de conjunção, disjunção e negação de fórmulas lógicas (KRÖTZSCH *et al.*, 2014). Por exemplo, podemos criar um *conceito complexo Mother* que é a *interseção* dos conceitos *Female* e *Parent*, como mostrado abaixo.

```
Mother ≡ Female ⊓ Parent
```

O *conceito complexo Mother* representa o conjunto de indivíduos que são ao mesmo tempo *Female* e *Parent*. Em relação ao conceito *união*, podemos declarar, por exemplo, o *conceito complexo Parent* que representa o conjunto de indivíduos que são *Father* ou *Mothter*, como mostrado abaixo.

$$\texttt{Parent} \equiv \texttt{Father} \sqcup \texttt{Mother}$$

Se quisermos representar o *conceito complexo* das mulheres não casadas, chamado *WomenNotMarried*, podemos declará-lo usando a interseção do conceito *Female* com conceito ¬*Married*, como segue.

$$\texttt{WomenNotMarried} \equiv \texttt{Female} \sqcap \neg\texttt{Married}$$

O conceito ¬*Married* representa o conjunto de todos os indivíduos não casados. O operador de complemento ¬ é usado para definir uma classe de indivíduos que *não* pertencem a um certo conceito.

Na Lógica de Descrição, existem dois conceitos especiais chamados *top concept* ⊤ e *bottom concept* ⊥. O conceito ⊤ representa o conjunto de todos os indivíduos da base de conhecimento *KB* e o conceito ⊥ representa um conceito especial que não possui instância alguma. O *top concept* ⊤ é útil quando queremos fazer sentenças sobre todos os indivíduos. Por exemplo, para declarar que o conjunto de todos os indivíduos do domínio pertence ao conceito *Male* ou *Female*, usa-se o conceito especial *top concept* ⊤, como segue.

$$\top \sqsubseteq \texttt{Male} \sqcup \texttt{Female}$$

Se quisermos declarar que, no nosso domínio, os conceitos *Male* e *Female* são disjuntos, deve-se usar o conceito especial *bottom concept* ⊥, como mostrado abaixo.

$$\texttt{Male} \sqcap \texttt{Female} \sqsubseteq \bot$$

Uma das características mais interessantes da Lógica de Descrição é sua habilidade de formar sentenças que conectam *conceitos* e *papéis* juntos através de *restrições nos papéis* (*role restrictions* em inglês) (KRÖTZSCH *et al.*, 2014). As *restrições nos papéis* podem ser categorizadas em dois tipos: *restrição existencial* e *restrição universal*. A *restrição existencial* é usada para descrever classes de indivíduos que participam de um relacionamento com *pelo menos um* indivíduo de uma determinada classe. A *restrição universal* é usada para representar um conjunto de indivíduos que participam de um relacionamento com *somente* indivíduos de uma classe especificada. É importante observar que a *restrição universal* também representa aqueles indivíduos que não participam de nenhum relacionamento com indivíduos da classe especificada pela restrição.

Por exemplo, a sentença abaixo descreve o conceito *CheesyPizza* como sendo uma *Pizza* que tenha *pelo menos uma* cobertura de queijo, ou seja, conceito *CheesyPizza* é formado por aqueles os indivíduos que tem *pelo menos um* relacionamento *hasTopping* com um indivíduo que seja membro do conceito *CheeseTopping* (HORRIDGE e BRANDT, 2011).

$$\mathtt{CheesyPizza} \equiv \mathtt{Pizza} \sqcap \exists\, \mathtt{hasTopping.CheeseTopping}$$

O exemplo abaixo mostra a descrição do conceito *MargheritaPizza* como sendo uma *Pizza* que tem *pelo menos uma* cobertura de queijo e *pelo menos uma* cobertura de tomate e *somente* coberturas de *Mozzarella* ou de *Tomato* (HORRIDGE e BRANDT, 2011).

$$\begin{aligned}&\mathtt{MargheritaPizza} \sqsubseteq \\ &\qquad \forall\, \mathtt{hasTopping.(\ MozzarellaTopping} \sqcup \mathtt{\ TomatoTopping\ )} \sqcap \\ &\qquad \exists\, \mathtt{hasTopping.MozzarellaTopping} \sqcap \\ &\qquad \exists\, \mathtt{hasTopping.TomatoTopping}\end{aligned}$$

O exemplo abaixo descreve o conceito *ParentOfGirl* como sendo um conjunto de indivíduos que tem *pelo menos um* filho e que *todos* os filhos são *Female*.

$$\mathtt{ParentOfGirl} \sqsubseteq \exists\ \mathtt{parentf.}\top \sqcap \forall\ \mathtt{parentOf.Female}$$

Em DL, pode-se usar *restrições de número* para restringir o número de indivíduos que são atingidos através de um *papel* (KRÖTZSCH *et al.*, 2014). Por exemplo, o axioma abaixo declara que toda *Person* é filho de exatamente dois indivíduos membros do conceito *Parent*.

$$\mathtt{Person} \sqsubseteq \geq 2\ \mathtt{childOf.Parent} \sqcap \leq 2\ \mathtt{childOf.Parent}$$

A Lógica de Descrição permite definir conceitos pela simples enumeração de suas instâncias através do uso de *nominal*. *Nominal* é um conceito que tem exatamente uma instância (KRÖTZSCH *et al.*, 2014). Por exemplo, *{sunday}* é um conceito, cuja única instância é o indivíduo representado por *sunday*. Enumerações em DL são representadas como a união de *nominais*, como mostrado no exemplo abaixo.

$$\begin{aligned}\mathtt{DaysOfTheWeek} \equiv\ &\mathtt{\{sunday\}} \sqcup \mathtt{\{monday\}} \sqcup \mathtt{\{tuesday\}} \sqcup \\ &\mathtt{\{wednesday\}} \sqcup \mathtt{\{thursday\}} \sqcup \mathtt{\{friday\}} \sqcup \\ &\mathtt{\{saturday\}}\end{aligned}$$

Segundo KRÖTZSCH *et al.*, (2014), o uso de *Nominal* permite expressar axiomas do ABox como parte dos axiomas do TBox. Por exemplo, o axioma

$$\mathtt{Mother(julia)}$$

pode ser expressado como

$$\mathtt{\{Julia\}} \sqsubseteq \mathtt{Mother}$$

2.2.2.3 RELATIONAL KNOWLEDGE (RBOX)

Os axiomas do RBox capturam interdependências entre os *papéis* de uma base de conhecimento *KB* da Lógica de Descrição. Como nos conceitos, a Lógica de Descrição suporta axiomas de *role inclusion* e *role equivalence* para os *papéis* (KRÖTZSCH *et al.*, 2014). Por exemplo, o axioma abaixo declara o *papel hasMother* é um sub-papel de *hasParent*, ou seja, todo par de indivíduos relacionados através do *papel hasMother* é também relacionado através do *papel hasParent*.

$$\texttt{hasMother} \sqsubseteq \texttt{hasParent}$$

Em DL, pode-se declarar papéis disjuntos. Por exemplo, a sentença abaixo declara que os *papéis childOf* e *parentOf* são disjuntos, que significa que ninguém pode ser *parentOf* e *childOf* de um mesmo indivíduo.

$$\texttt{Disjoint(parentOf, childOf)}$$

Nos axiomas de *role inclusion*, pode-se criar *composições de papéis* capturado através do axioma *complex role inclusion* (KRÖTZSCH *et al.*, 2014). A *composição de papéis* pode ser usada para descrever *papéis*, tal como o *papel uncleOf*. Intuitivamente, se o indivíduo *A* é *irmão de B* e o indivíduo *B* é *pai de C*, então pode-se inferir que *A* é *tio de C*. O *papel uncleOf* pode ser capturado pelo seguinte axioma.

$$\texttt{brotherOf} \circ \texttt{parentOf} \sqsubseteq \texttt{uncleOf}$$

Um dos principais axiomas do RBox, é a representação de *papéis inversos*. Por exemplo, pode-se definir o *papel parentOf* como inverso ao papel *childOf*, através do seguinte axioma

$$\texttt{parentOf} \equiv \texttt{childOf}^{-}$$

onde o *papel childOf*$^{-}$ representa o inverso do *papel childOf*. Então, se o indivíduo *A* é *pai de B*, então pode-se inferir que *B* é *filho de A*.

Além disso, a Lógica de descrição provê alguns axiomas para representar algumas características nos *papéis*, tais como *transitividade*, *simetria* e *reflexividade*. Para definir que uma propriedade é transitiva, pode-se usar a palavra *Trans*. Por exemplo, o axioma abaixo define que o *papel subRegionOf* tem a característica transitiva.

$$\texttt{Trans(subRegionOf)}$$

A transitividade, como as demais características, pode ser também capturada usando axiomas já introduzidos nesta seção. Por exemplo, pode-se usar o axioma *complex role inclusion* para definir a transitividade. O exemplo abaixo mostra a definição da transitividade para o *papel subRegionOf* , usando o axioma *complex role inclusion*.

$\texttt{subRegionOf} \circ \texttt{subRegionOf} \sqsubseteq \texttt{subRegionOf}$

Um *papel* é simétrico se ele é equivalente com o seu inverso. Então, por exemplo, para definir que o *papel marriedTo* é simétrico, basta escrever o seguinte axioma.

$\texttt{marriedTo} \equiv \texttt{marriedTo}^{-}$

Um *papel* é assimétrico quando ele é disjunto de seu inverso. Portanto, se os *papéis* parentOf e parentOf $^{-}$ são assimétricos, então o seguinte axioma pode ser usado.

$\texttt{Disjoint(parentOf, parentOf}^{-}\texttt{)}$

Para definir uma propriedade reflexiva em um *papel*, usa-se a palavra *self* para referir-se ao mesmo indivíduo. Por exemplo, a característica reflexiva do *papel knows* é definida pelo seguinte axioma (KRÖTZSCH *et al.*, 2014).

$\top \sqsubseteq \exists\ \texttt{knows.Self}$

2.2.3 CORRESPONDÊNCIA DOS AXIOMAS DA LINGUAGEM OWL DL COM A LÓGICA DE DESCRIÇÃO

Como mencionado na Seção 2.2.2, a Lógica de Descrição na qual a OWL DL é baseada chama-se $\mathcal{SHOIN}(\mathcal{D})$ (HORROCKS *et al.*, 2007). A Lógica de Descrição $\mathcal{SHOIN}(\mathcal{D})$ é baseada em uma extensão da Lógica de Descrição, chamada $\mathcal{ALC}$, que também é conhecida como lógica $\mathcal{S}$, para incluir transitividade nos *papéis* (HORROCKS *et al.*, 2007). Em seguida, outras extensões foram realizadas para incluir algumas características, tais como *Role Inclusion Axioms* ($\mathcal{H}$), *Nominals* ($\mathcal{O}$), *Inverse Roles* ($\mathcal{I}$), *Number Restrictions* ($\mathcal{N}$) e *Datatypes* ($\mathcal{D}$).

A Tabela 1 mostra a correspondência dos axiomas da OWL DL, usando a sintaxe abstrata, com a Lógica de Descrição. A primeira coluna mostra a sintaxe abstrata das estruturas da linguagem OWL DL e a segunda coluna apresenta a equivalência na sintaxe da Lógica de Descrição (HORROCKS *et al.*, 2007). Segundo os autores, as letras *A* e *C* representam conceitos e *R*, *U*, *o* e *v* representam, respectivamente, *object properties* (*abstract roles*), *datatype properties* (*concrete roles*), indivíduos (*nominals*) e *data values*, obtido a partir do *XML Schema datatypes* ou um LITERAL definido em RDF.

Tabela 1. Axiomas da OWL DL, usando sintaxe abstrata, e sua correspondência na Lógica de Descrição (HORROCKS *et al.*, 2007).

Abstract Syntax	DL Syntax
Class(A partial $C_1 \ldots C_n$)	$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$
Class(A complete $C_1 \ldots C_n$)	$A \equiv C_1 \sqcap \ldots \sqcap C_n$
EnumeratedClass(A $o_1 \ldots o_n$)	$A \equiv \{o_1\} \sqcup \ldots \sqcup \{o_n\}$
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$
EquivalentClasses($C_1 \ldots C_n$)	$C_1 \equiv \ldots \equiv C_n$
DisjointClasses($C_1 \ldots C_n$)	$C_i \sqcap C_j \subseteq \bot, i \not\mathbb{D} j$
Datatype(D)	
ObjectProperty(R super(R_1)...super(R_n)	$R \sqsubseteq R_i$
domain(C_1)...domain(C_m)	$\geqslant 1\, R \sqsubseteq C_i$
range(C_1)...range(C_ℓ)	$\top \sqsubseteq \forall R.C_i$
[inverseOf(R_0)]	$R \equiv R_0^-$
[Symmetric]	$R \equiv R^-$
[Functional]	$\top \sqsubseteq \leqslant 1\, R$
[InverseFunctional]	$\top \sqsubseteq \leqslant 1\, R^-$
[Transitive])	$Tr(R)$
SubPropertyOf(R_1 R_2)	$R_1 \sqsubseteq R_2$
EquivalentProperties($R_1 \ldots R_n$)	$R_1 \equiv \ldots \equiv R_n$
DatatypeProperty(U super(U_1)...super(U_n)	$U \sqsubseteq U_i$
domain(C_1)...domain(C_m)	$\geqslant 1\, U \sqsubseteq C_i$
range(D_1)...range(D_ℓ)	$\top \sqsubseteq \forall U.D_i$
[Functional])	$\top \sqsubseteq \leqslant 1\, U$
SubPropertyOf(U_1 U_2)	$U_1 \sqsubseteq U_2$
EquivalentProperties($U_1 \ldots U_n$)	$U_1 \equiv \ldots \equiv U_n$
AnnotationProperty(S)	
OntologyProperty(S)	
Individual(o type(C_1)...type(C_n)	$o \in C_i$
value(R_1 o_1)...value(R_n o_n)	$\langle o, o_i \rangle \in R_i$
value(U_1 v_1)...value(U_n v_n))	$\langle o, v_i \rangle \in U_i$
SameIndividual($o_1 \ldots o_n$)	$\{o_1\} \equiv \ldots \equiv \{o_n\}$
DifferentIndividuals($o_1 \ldots o_n$)	$\{o_i\} \sqsubseteq \neg\{o_j\}, i \not\mathbb{D} j$

2.3 REGRAS EM SWRL

Regras na Web Semântica são um meio de representar o conhecimento por meio de inferências que geralmente vai além da representação usada pela linguagem OWL DL (*Web Ontology Language*) (HEBELER *et al.*, 2009, p. 232). Elas são tipicamente sentenças condicionais do tipo “se-então”, mais formalmente referidas como *implicação*. Quando o conjunto de fatos representado no antecedente de uma regra for verdadeiro, os fatos

especificados no consequente da regra são também considerados verdadeiros e, por isso, são adicionados à base de conhecimento.

A *Semantic Web Rule Language*[4] (SWRL) é uma linguagem de regras expressiva baseada em OWL DL e OWL Lite, e um subconjunto da linguagem RuleML[5] (*Rule Markup Language*), que modela *cláusulas Horn* (HORROCKS *et al.*, 2004). SWRL permite que regras na forma de *cláusulas Horn* sejam combinadas com ontologias OWL DL. O subconjunto de RuleML suportado por SWRL inclui somente predicados unários e binários. Segundo Connor (2017), SWRL permite aos usuários escreverem regras que podem ser expressadas em termos de conceitos OWL a fim de prover recursos dedutivos mais poderosos que a linguagem OWL unicamente. Além disso, semanticamente, a linguagem SWRL é construída com o mesmo fundamento lógico da linguagem OWL DL e provê as mesmas garantias formais quando as inferências são executadas.

Uma das vantagens de se usar regras SWRL em relação a linguagem OWL é o seu suporte a funções incorporadas. As funções incorporadas permitem a transformação dos dados e envolvem operações matemáticas, comparações, construções de URI, manipulação de *strings*, datas, tempo, listas, entre outros exemplos. A máquina de inferência *Pellet* para ontologias OWL também oferece suporte à linguagem SWRL (SIRIN *et al.*, 2007). SWRL tornou-se W3C *Member Submission* em 2004.

Uma regra em SWRL possui a forma de uma *implicação* entre um *antecedente* (comumente referido como *body* ou cláusula *se*) e um *consequente* (referido também como *head* ou cláusula *então*). O antecedente e consequente da regra são formados pela *conjunção* de zero ou mais átomos. Se o *antecedente* de uma regra tem nenhum átomo, então ela é avaliada como verdadeira, fazendo que todas as sentenças no *consequente* sejam verdadeiras. No caso de o *consequente* da regra ter nenhum átomo, então nenhuma informação é adicionada na ontologia e a regra é avaliada como falsa. Uma regra com *consequente* vazio significa que não pode ser satisfeita por nenhuma ontologia, ou seja, se o *antecedente* desta regra for verdade, então a ontologia contém uma inconsistência lógica (HEBELER *et al.*, 2009). Zero átomos no *antecedente* ou *consequente* são casos especiais. Tipicamente, uma regra SWRL terá pelo menos um átomo no *antecedente* e um no *consequente*.

Disjunção (operador lógico OR) não é diretamente suportado pela SWRL, porém ela pode ser implementada dividindo a disjunção em regras separadas (HEBELER *et al.*, 2009). A

[4] http://www.w3.org/Submission/SWRL/
[5] http://ruleml.org/

linguagem SWRL possui três diferentes sintaxes: uma sintaxe abstrata que é uma extensão da sintaxe abstrata da OWL, e duas sintaxes concretas, uma baseada no XML e a outra baseada em RDF. Há também a sintaxe *human-readable* das regras SWRL. Tal sintaxe é usada neste trabalho e apresentada a seguir.

A implicação em SWRL é representada por uma seta (⟶), conectando logicamente um *antecedente* de um *consequente* e os átomos são associados através do caractere de conjunção (∧), como mostrado abaixo.

```
átomo ∧ átomo ... ⟶ átomo ∧ átomo
```

Um átomo é uma expressão com a seguinte forma

```
p(arg1, arg2, ... argn)
```

onde *p* é um símbolo de predicado e *arg1*, *arg2*, ... *argn* são argumentos da expressão. Em SWRL, o símbolo de predicado pode ser uma classe OWL, propriedades ou tipos de dados. Os argumentos podem ser indivíduos, valores de dados ou variáveis. *Todas as variáveis* em SWRL são tratadas como *universalmente quantificadas*, com escopo limitado pela regra (Connor, 2017). As variáveis são precedidas por um símbolo de interrogação (?).

O significado pretendido de uma regra em SWRL pode ser lido como: a qualquer hora em que as sentenças especificadas no *antecedente* forem satisfeitas, então as sentenças especificadas no *consequente* devem ser satisfeitas também. Em outras palavras, todas os átomos no *antecedente* devem ser simultaneamente verdadeiros para causar que todos os átomos no *consequente* sejam verdadeiros.

Segundo HEBELER *et al.*, (2009), um átomo pode ser um dos seguintes tipos:

- Átomo Classe
- Átomo *Individual Property*
- Átomo *Data Valued Property*
- Átomo *Same Individual*
- Átomo *Different Individuals*
- Átomo *Data Range*
- Átomo *Built-in*

Um átomo *Classe* consiste de uma classe OWL e um único argumento representando um indivíduo OWL. Por exemplo,

```
Person(?p)
Man(John)
```

mostram dois átomos *Classe*, onde os nomes *Person* e *Man* são classes OWL, *?p* é uma variável representando um indivíduo OWL e *John* é o nome de um indivíduo OWL. Se átomo *Classe* aparece no *antecedente* de uma regra SWRL, então o especificado indivíduo ou a variável devem pertencer a esta classe para que o átomo seja verdadeiro. Se o átomo *Classe* aparece no *consequente* da regra, então o especificado indivíduo ou a variável serão uma instância desta classe, caso a regra seja satisfeita.

Um exemplo de regra SWRL envolvendo apenas átomos *Classe* que declara que todo indivíduo membro da classe *Man* é também indivíduo membro da classe *Person*, pode ser escrito da seguinte forma.

```
Man(?p) ⟶ Person(?p)
```

Um átomo *Individual Property* consiste de uma propriedade *object property* da ontologia OWL seguida de dois argumentos representando indivíduos OWL. O primeiro argumento é o sujeito da propriedade e o segundo argumento é o objeto da propriedade. Abaixo mostra dois exemplos de átomo *Individual Property*.

```
hasBrother(?x, ?y)
hasSibling(John, ?y)
```

As palavras *hasBrother* e *hasSibling* são propriedades *object property*, *?x* e *?y* são variáveis representando indivíduos OWL, e *John* é o nome de um indivíduo OWL. Se o átomo *Individual Property* aparece no *antecedente* de uma regra SWRL, então a tripla <sujeito, propriedade, objeto> deve existir na ontologia OWL para que este átomo seja verdadeiro. Porém, se o átomo aparece no *consequente* da regra, então a tripla será declarada na ontologia OWL, caso a regra seja satisfeita.

Um exemplo de regra SWRL envolvendo átomos *Individual Property* e *Classe* é mostrado abaixo. A regra declara que *toda* pessoa, membro da classe *Person*, que tem um relacionamento com um indivíduo OWL através da propriedade *hasSibling*, e esse indivíduo é membro da classe *Man*, deve estar relacionada também com o mesmo indivíduo através da propriedade *hasBrother*.

```
Person(?p) ∧ hasSibling(?p,?s) ∧ Man(?s)
⟶ hasBrother(?p,?s)
```

Um átomo *Data Valued Property* consiste de uma propriedade *datatype property* da ontologia OWL seguida de dois argumentos, o primeiro representando um indivíduo OWL e o segundo argumento é um literal. Assim como no átomo *Individual Property*, se o átomo *Data Valued Property* aparece no *antecedente* de uma regra SWRL, então a tripla <sujeito,

propriedade, objeto> deve existir na ontologia OWL para que o átomo seja verdadeiro. Porém, se ele aparece no *consequente* da regra, então a tripla será declarada na ontologia OWL, caso a regra seja satisfeita. Abaixo mostra alguns exemplos de átomos *Data Valued Property*.

```
hasHeight(John, ?h)
hasAge(?x, 35)
hasName(?x, "John")
```

As palavras *hasHeight*, *hasAge* e *hasName* são propriedades *datatype property*, *?x* é uma variável representando um indivíduos OWL, *John* é o nome de um indivíduo OWL, *?h* é uma variável representando um literal, e *35* e *"John"* são literais do tipo inteiro e *string*, respectivamente. Um exemplo de átomo *Data Valued Property* é mostrado na regra SWRL abaixo. A regra declara que *toda* pessoa, membro da classe *Person*, que tem um carro é classificada como membro da classe *Driver*.

```
Person(?p) ∧ hasCar(?p, true) ⟶ Driver(?p)
```

Um átomo *Same Individual* consiste do símbolo *sameAs* seguido de dois argumentos representando indivíduos OWL. Ele declara igualdade entre dois indivíduos OWL. Essa declaração é equivalente a usar *owl:sameAs* nas ontologias OWL. Por exemplo,

```
sameAs(JohnHebeler, JohnnyHebeler)
sameAs(?x, ?y)
```

declaram, respectivamente, que os indivíduos OWL *JohnHebeler* e *JohnnyHebeler* são iguais, assim como as variáveis *?x* e *?y* representando indivíduos OWL.

Um átomo *Different Individuals* consiste do símbolo *differentFrom* seguido de dois argumentos representando indivíduos OWL. Esse átomo declara que dois indivíduos OWL são diferentes. Tal declaração é equivalente a usar *owl:differentFrom* nas ontologias OWL. Abaixo mostramos um exemplo de átomo *Different Individuals*, no qual declara que *John* e *Joseph* são indivíduos OWL distintos.

```
differentFrom(John, Joseph)
```

Um átomo *Data Range* consiste de um *datatype* ou um conjunto de literais seguido por um único argumento que é uma variável representando um valor de dado. Abaixo seguem alguns exemplos de átomo *Data Range*.

```
xsd:int(?x)
[3, 4, 5](?x)
[“John”, “Matt”, “Ryan”, “Andrew”](?fname)
```

As variáveis *?x* e *?fname* estão representando valores de dados. O primeiro exemplo declara que *?x* tem um valor inteiro. O segundo e terceiro exemplos declaram que *?x* e *?fname* têm um dos valores fornecidos pelos correspondentes conjuntos de literais em colchetes.

Uma das características mais poderosas do SWRL é a sua habilidade de suportar funções incorporadas para executar operações de comparações, matemática, *strings*, data e outros (HORROCKS *et al.*, 2004). O átomo *Built-in* declara funções embutidas da linguagem. Ele é um predicado que pode ter um ou mais argumentos, e é avaliado como verdadeiro se os argumentos satisfazem o predicado. Por convenção, as funções incorporadas são precedidas pelo prefixo *swrlb*, cujo *namespace* é http://www.w3.org/2003/11/swrlb.

Um exemplo de regra SWRL com átomo *Built-in* que executa operação de comparação é apresentado abaixo. A regra define que uma pessoa com idade maior que 17 é um adulto.

```
Person(?p) ∧ hasAge(?p, ?age) ∧ swrlb:greaterThan(?age, 17)
⟶ Adult(?p)
```

Um exemplo de regra SWRL com átomo *Built-in* que faz manipulação de *strings* é mostrado a seguir. A regra que determina se o número de telefone de uma pessoa inicia com o código internacional “+”, pode ser escrita da seguinte forma.

```
Person(?p) ∧ hasNumber(?p, ?number) ∧
swrlb:startsWith(?number, "+")
⟶ hasInternationalNumber(?p, true)
```

Outro exemplo de regra SWRL com átomo *Built-in* que faz manipulação de *strings* é mostrado a seguir. Tal regra faz uso da função embutida *swrlb:stringConcat* que é responsável por fazer a concatenação de *strings*. O primeiro argumento da função embutida armazena o resultado da concatenação das *strings* referentes ao segundo argumento até o último argumento da função (HORROCKS *et al.*, 2004).

```
foaf:Person(?person) ∧
foaf:gender(?person, "female") ∧
foaf:name(?person, ?name)
⟶ swrlb:stringConcat(?s, "Dear Ms. ", ?name, ":") ^
hasFormalGreeting(?person, ?s)
```

A linguagem SWRL não provê mecanismos para verificar se os argumentos de um átomo *Built-in* estão com quantidade incorreta ou com algum tipo errado. Caso isso aconteça, a regra SWRL contendo tal átomo é avaliada como falsa. Além disso, a linguagem SWRL além de não suportar disjunções, como já mencionado nesta seção, ela também não suporta negação

de átomos (*negation as failure*) (HORROCKS *et al.*, 2005). Então, por exemplo, a seguinte regra não é permitida em SWRL.

$$\texttt{Person(?p)} \wedge \neg\ \texttt{hasCar(?p, ?c)} \longrightarrow \texttt{CarlessPerson(?p)}$$

CAPÍTULO 3 – MECANISMO PARA DETECÇÃO DOS PCN

Este capitulo apresentada a especificação da estrutura normativa, dividida em duas ontologias OWL DL: *ontologia de Norma* e *ontologia de Cenário de Execução*. Em seguida, as aplicações das regras SWRL para inferir os tempos das condições que definem os intervalos de ativação das normas e as duas abordagens para a detecção dos PCN entre pares de normas são apresentadas. Além disso, uma ferramenta para a verificação dos potenciais conflitos normativos, chamada *Potential Conflict Checker*, e as tarefas que podem ser executas nela também são descritas neste capítulo.

3.1 ESPECIFICAÇÃO DA NORMA BASEADA EM ONTOLOGIA

Ontologias são usadas para capturar o conhecimento sobre um certo domínio de interesse (HORRIDGE *et al.*, 2004). Uma ontologia formalmente descreve conceitos e relacionamentos que podem existir entre estes conceitos (SYNAK *et al.*, 2009). Em outras palavras, uma ontologia descreve uma parte do mundo. Segundo SYNAK *et al.* (2009), um conceito em uma ontologia pode representar uma variedade de coisas, como um objeto: um carro, um prédio, como também pode descrever uma atividade ou um estado: nadando, estar ocupado ou disponível. Conceitos em uma ontologia podem também representar conceitos abstratos, tais como tempo ou valor. A única restrição, de acordo com os autores, é que a ontologia deve tentar refletir o mundo real.

A estrutura normativa utilizada neste trabalho foi representada usando a linguagem OWL DL. A OWL *Web Ontology language* é uma expressiva linguagem de representação do conhecimento, padronizada pela *World Wide Web Consortium* (*W3C*). Ela pode ser usada para definir formalmente um conjunto comum de termos que são usados para descrever e representar um certo domínio. Como descrito na Seção 2.2, uma das vantagens de usar OWL DL é que ela é uma linguagem bastante expressiva e há ferramentas de inferências para OWL DL eficientes, livres e amplamente usadas, tais como, HermiT[6] e Pellet[7]. Além disso, inferências em OWL DL garantem que todas as conclusões sejam computáveis (*completeness*) e que todos os cálculos terminem em tempo finito (*decidability*) (SMITH *et al.*, 2004).

Uma ontologia OWL DL permite a definição de classes (também conhecida como *conceitos* na literatura da DL), propriedades (também conhecida como *papéis* em DL), e

[6] http://hermit-reasoner.com
[7] http://clarkparsia.com/pellet

indivíduos (também chamado de instâncias). Propriedades são relacionamentos binários e podem ser de dois tipos principais: propriedades do tipo objeto (*owl:ObjectProperty*), no qual descreve relacionamentos binários entre dois indivíduos, e propriedades do tipo *datatype* (*owl:DatatypeProperty*), que representa um relacionamento binário entre um indivíduo e um valor de dado obtido a partir do *XML Schema datatypes* ou um LITERAL definido em RDF.

A especificação da norma neste trabalho foi dividida em duas ontologias OWL DL: *ontologia de Norma* e *ontologia de Cenário de Execução* (BELCHIOR e DA SILVA, 2017a, 2017b). A ontologia de Norma descreve os principais conceitos de uma norma em um sistema multi-agentes. A ontologia de Cenário de Execução descreve cenários referentes a possíveis casos de execução do sistema. As ontologias de Norma e Cenário de Execução são apresentadas nas seções 3.1.1 e 3.1.2, respectivamente.

3.1.1 ONTOLOGIA DE NORMA

As principais classes da ontologia de Norma em OWL DL desenvolvida neste trabalho são: *Norm*, *Context*, *DeonticConcept*, *Entity*, *Action*, *Condition*, *FulfillmentStatus* e *ActivationStatus*. A classe *Norm* representa a definição de uma norma como mecanismo para regular o comportamento dos agentes em SMA. A classe *Context* determina a área de aplicação da norma. A classe *DeonticConcept* descreve restrições de comportamento aos agentes na forma de obrigações, permissões e proibições. A classe *Entity* descreve as entidades cujo comportamento é regulado por uma norma. O comportamento sendo regulado pela norma é definido pela classe *Action*. A classe *Condition* determina o período durante o qual uma norma fica ativa. As classes *ActivationStatus* e *FulfillmentStatus* representam o estado de ativação da norma e o estado de obediência do agente quanto ao cumprimento da norma, respectivamente.

A definição de cada um destes conceitos representados na ontologia de Norma é apresentada nas próximas subseções.

3.1.1.1 DEFINIÇÃO DA CLASSE NORMA

A classe *Norm* representa a definição de uma norma como mecanismo para regular o comportamento dos agentes em SMA. Uma instância da classe *Norm* está relacionada com indivíduos, membros das classes *Context*, *DeonticConcept*, *Entity*, *Action*, *ActivationStatus* e *FulfillmentStatus*, através das propriedades do tipo objeto *hasContext*, *hasDeonticConcept*, *hasEntity*, *hasAction*, *hasActivationStatus* e *hasFulfillmentStatus*, respectivamente. Uma norma está também conectada com instâncias da classe *Condition* via duas propriedades do tipo objeto: *hasBefore* e *hasAfter*. Uma norma pode também ter um auto-relacionamento com instâncias de

norma através da propriedade do tipo objeto e simétrica (*owl:SymmetricProperty*) chamada *hasConflict*. Se *P* é uma propriedade simétrica, e ela relaciona indivíduo *a* com indivíduo *b*, então indivíduo *b* é também relacionado com indivíduo *a* através da mesma propriedade *P*. A propriedade *hasConflict* representa um conflito normativo entre duas instâncias de normas. Uma instância da classe *Norm* pode ter um nome e uma justificativa para o potencial conflito normativo através das propriedades do tipo datatype chamadas *normName* e *conflictJustification*, respectivamente. A classe *Norm* da ontologia OWL DL de Norma é definida usando Lógica de Descrição, como segue.

$$
\begin{aligned}
\text{Norm} \equiv\ & =1\ \text{hasContext.Context} \sqcap \\
& =1\ \text{hasDeonticConcept.DeonticConcept} \sqcap \\
& =1\ \text{hasEntity.Entity} \sqcap \\
& =1\ \text{hasAction.Action} \sqcap \\
& \leq 1\ \text{hasBefore.Condition} \sqcap \\
& \leq 1\ \text{hasAfter.Condition} \sqcap \\
& =1\ \text{hasActivationStatus.ActivationStatus} \sqcap \\
& =1\ \text{hasFulfillmentStatus.FulfillmentStatus} \sqcap \\
& \forall\ \text{hasConflict.Norm} \sqcap \\
& =1\ \text{normName.String} \sqcap \\
& \forall\ \text{conflictJustification.String}
\end{aligned}
$$

3.1.1.2 DEFINIÇÃO DA CLASSE CONTEXTO

A classe *Context* representa o escopo onde a norma é definida, ou seja, onde a norma deve ser aplicada. Fora do contexto, a norma não é válida. O contexto é definido na ontologia de Norma através da propriedade do tipo objeto *hasContext* entre as classes *Norm* e *Context*. As normas são geralmente especificadas em dois tipos de contextos: contexto de uma organização (classe *Organization*) ou contexto de um ambiente (classe *Environment*) (FIGUEIREDO *et al.*, 2011, DA SILVA e ZAHN, 2014). As classes *Organization* e *Environment* são definas na ontologia como subclasses de *Context*, como mostrado abaixo.

$$
\begin{aligned}
& \text{Organization} \sqsubseteq \text{Context} \\
& \text{Environment} \sqsubseteq \text{Context}
\end{aligned}
$$

$$
\text{Context} \sqsubseteq =1\ \text{contextName.String}
$$

3.1.1.3 DEFINIÇÃO DA CLASSE CONCEITO DEÔNTICO

A classe *DeonticConcept* descreve restrições de comportamento aos agentes na forma de *obrigações*, *permissões* e *proibições*. O conceito deôntico de uma norma é definido através da propriedade do tipo objeto *hasDeonticConcept*. Os indivíduos *Obligation*, *Permission* e

Prohibition foram adicionados na ontologia de Norma e a classe *DeonticConcept* foi definida pela enumeração de seus membros usando o conceito de *Nominals* da Lógica de Descrição (RUDOLPH, 2011), como mostrado abaixo.

$$\text{DeonticConcept} \equiv \{ \text{Obligation} \} \sqcup \{ \text{Permission} \} \sqcup \{ \text{Prohibition} \}$$

3.1.1.4 DEFINIÇÃO DA CLASSE ENTIDADE

A classe *Entity* descreve as entidades cujo comportamento é regulado pela norma. A entidade é o sujeito de uma ação sendo controlada pela norma. As entidades definidas neste trabalho representam os próprios agentes. Uma instância da classe *Entity* pode ter um relacionamento com uma instância da classe *Context*, através da propriedade do tipo objeto *actsIn*, para determinar em qual contexto tal entidade atua. Uma entidade pode executar uma ação no sistema multi-agentes, representada pela propriedade do tipo objeto *performsAction* entre a entidade e a ação, instância da classe *Action*. Uma entidade pode também participar de situações, instâncias da classe *Situation*. Uma situação neste trabalho representa um fato na base de dados. Por exemplo, um agente tem um carro, vive no Rio de Janeiro, ou possui uma graduação são exemplos de situações que um agente pode participar. Um agente pode participar de zero, uma ou várias situações através da propriedade do tipo objeto *participatesIn*. A propriedade do tipo datatype chamada *entityName* define o nome da entidade. A classe *Entity* é definida em DL, como segue.

$$\begin{aligned} \text{Entity} \sqsubseteq\ & \forall\ \text{actsIn.Context} \sqcap \\ & \forall\ \text{performsAction.Action} \sqcap \\ & \forall\ \text{participatesIn.Situation} \sqcap \\ & =1\ \text{entityName.String} \end{aligned}$$

3.1.1.5 DEFINIÇÃO DA CLASSE AÇÃO

O comportamento sendo regulado por uma norma é definido pela classe *Action*. As ações representadas neste trabalho são ações atômicas. Uma ação pode ser executada por indivíduos que são membros da classe *Entity* através da propriedade do tipo objeto *isPerformedBy*, que é definida como propriedade inversa (*owl:inverseOf*) da propriedade *performsAction*. A propriedade *performsAction* é apresentada na Seção 3.1.1.4. A propriedade do tipo datatype chamada *actionName* define o nome da ação. A classe *Action* é definida em DL, como segue.

$$\begin{aligned} \text{Action} \sqsubseteq\ & \forall\ \text{isPerformedBy.Entity} \sqcap \\ & =1\ \text{actionName.String} \end{aligned}$$

$$isPerformedBy \equiv performsAction^{-}$$

3.1.1.6 DEFINIÇÃO DA CLASSE CONDIÇÃO

A classe *Condition* determina o período durante o qual uma norma fica ativa. Uma norma é considerada ativa quando ela foi ativada e ainda não foi desativada. Quando uma norma está ativa no sistema, ela deve ser cumprida em algum momento pelos agentes. Neste trabalho, as condições de uma norma estão relacionadas a eventos que ocorrem em tempo de execução. Deste modo, foram considerados seis tipos de condições que ocorrem em tempo de execução: (i) execução de uma ação por um agente, (ii) ativação ou (iii) desativação de uma norma, (iv) cumprimento ou (v) violação de uma norma e um (vi) fato no sistema que tenha se tornado verdade para um agente. Todas essas condições dependem da execução do sistema e não podem ser determinadas em tempo de design. Tais condições são representadas na ontologia de Norma pelas seguintes classes, respectivamente: *ExecutionOfAction*, *ActivationOfNorm*, *DeactivationOfNorm*, *FulfillmentOfNorm*, *ViolationOfNorm* e *SituationParticipation*. Tais classes são definidas como subclasses da classe *Condition*. As classes *ActivationOfNorm*, *DeactivationOfNorm*, *FulfillmentOfNorm* e *ViolationOfNorm* possuem um relacionamento com a classe *Norm* através da propriedade do tipo objeto *hasRelatedNorm* para definir a norma relacionada a condição. As classes *ExecutionOfAction* e *SituationParticipation* devem definir, respectivamente, uma entidade que executa uma ação ou participa de uma situação, através da propriedade do tipo objeto *hasRelatedEntity*. Além disso, a classe *ExecutionOfAction* tem um relacionamento com uma ação através da propriedade *hasRelatedAction* e a classe *SituationParticipation* está associada a uma situação através da propriedade *hasRelatedSituation*. A propriedade do tipo datatype chamada *condName* define o nome da condição. As seis condições definidas neste trabalho são descritas em DL, como segue.

$$\begin{aligned}
&ActivationOfNorm \sqsubseteq Condition \sqcap \\
&\quad =1\ hasRelatedNorm.Norm
\end{aligned}$$

$$\begin{aligned}
&DeactivationOfNorm \sqsubseteq Condition \sqcap \\
&\quad =1\ hasRelatedNorm.Norm
\end{aligned}$$

$$\begin{aligned}
&FulfillmentOfNorm \sqsubseteq Condition \sqcap \\
&\quad =1\ hasRelatedNorm.Norm
\end{aligned}$$

$$\begin{aligned}
&ViolationOfNorm \sqsubseteq Condition \sqcap \\
&\quad =1\ hasRelatedNorm.Norm
\end{aligned}$$

$$\begin{aligned}
&ExecutionOfAction \sqsubseteq Condition \sqcap \\
&\quad =1\ hasRelatedAction.Action \sqcap \\
&\quad =1\ hasRelatedEntity.Entity
\end{aligned}$$

```
SituationParticipation ⊑ Condition ⊓
          =1 hasRelatedSituation.Situation ⊓
          =1 hasRelatedEntity.Entity

Condition ⊑ =1 condName.String
```

Na definição da classe *Norm*, uma norma pode estar relacionada com uma instância da classe *Condition* através da propriedade *hasBefore* ou da propriedade *hasAfter*. A norma pode estar relacionada com duas condições usando ambas as propriedades, como também pode não estar relacionada com condição alguma, ou seja, a norma não possui uma condição que determina seu período de ativação. Quando isto ocorre, a norma é considerada sempre ativa no sistema, ou seja, seu período de ativação se inicia do zero e dura até o termino da execução do sistema. Deste modo, pode-se definir cinco configurações de intervalos de ativação da norma no sistema, como ilustrado na Figura 3. A cor cinza na figura indica o período em que a norma está ativa de acordo com seu tipo de intervalo de ativação.

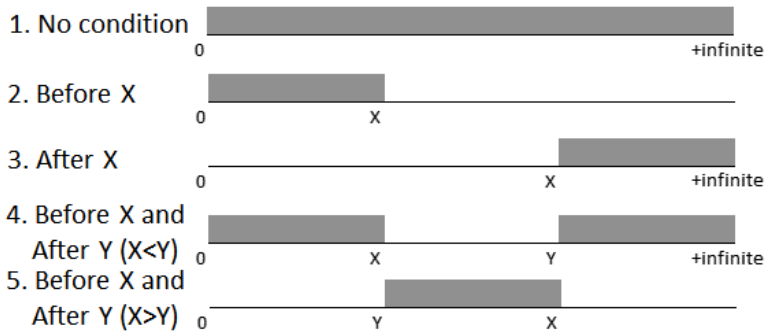


Figura 3: Os cinco tipos de intervalos de ativação da norma (BELCHIOR e DA SILVA, 2017a).

O primeiro tipo de intervalo de ativação refere-se a uma norma que não tem condição e está sempre ativa. Seu intervalo de ativação se inicia no tempo zero e dura até o +infinito, ou seja, até o termino da execução do sistema. O segundo tipo de intervalo representa uma norma associada a somente uma condição usando a propriedade *hasBefore*. Seu intervalo de ativação se inicia no tempo zero e dura até quando a condição *X* acontecer no sistema. O terceiro tipo representa uma norma associada a somente uma condição usando a propriedade *hasAfter*. Este intervalo de ativação se inicia quando a condição *X* ocorrer no sistema e dura até o +infinito. O quarto e quinto tipos de intervalo de ativação referem-se a uma norma que está associada a duas condições, uma (condição *X*) através do relacionamento *hasBefore* e a outra (condição *Y*) através do relacionamento *hasAfter*. Estes intervalos se diferem um do outro pelo momento em que as condições acontecem no sistema. Caso a condição com relacionamento *hasBefore* aconteça primeiro, então o intervalo de ativação da norma é caracterizado pelo quarto tipo. Caso

contrário, o intervalo de ativação da norma é representado pelo quinto tipo de intervalo. O quarto tipo de intervalo é o único em que a norma tem dois períodos de ativação, ou seja, começando do tempo zero e durando até quando a condição com relacionamento *hasBefore* acontecer no sistema, e a partir de quando a condição com relacionamento *hasAfter* for verdade no sistema até o +infinito. Portanto, as normas especificadas neste trabalho podem ter um ou no máximo dois intervalos de ativação, dependendo do seu tipo.

3.1.1.7 DEFINIÇÃO DA CLASSE ESTADO DE ATIVAÇÃO

A classe *ActivationStatus* representa o estado de ativação da norma que pode ser *ativado*, *desativado* ou *sem estado*. Quando uma norma foi ativada, significa que ela se tornou ativa e deve ser de alguma forma cumprida pelo agente. Uma vez que a norma foi ativada, ela pode ser desativada em algum momento e nenhuma ação é requerida pelo agente. O estado de ativação *sem estado* significa que a norma não foi ainda nem ativada e nem desativada. Esse é o estado de ativação inicial das normas ao se iniciar o sistema. Os indivíduos *Activated*, *Deactivated* e *None* foram adicionados na ontologia de Norma e a classe *ActivationStatus* foi definida pela enumeração de seus membros, como mostrado abaixo.

```
ActivationStatus ≡ {Activated } ⊔ { Deactivated } ⊔
                   { None }
```

3.1.1.8 DEFINIÇÃO DA CLASSE ESTADO DE OBEDIÊNCIA

A classe *FulfillmentStatus* representa o estado de obediência do agente quanto ao cumprimento da norma. O estado de obediência da norma pode ser *cumprido*, *violado* ou *desconhecido*. O estado *cumprido* significa que a norma foi cumprida pelo agente. O estado *violado* significa que a norma não foi cumprida pelo agente, ou seja, o agente violou a norma. O estado *desconhecido* significa que a norma ainda não foi nem cumprida e nem violada. Por exemplo, vamos supor que *N1* seja uma norma ativa que estabeleça que uma dada ação deve ser executada por um certo agente, mas tal ação ainda não foi executada. Neste caso, o estado de obediência da norma é *desconhecido*. Porém, se a ação foi executa, o estado de obediência mudará para *cumprido*. No entanto, se a norma se tornou desativada, ou seja, não está mais ativa, e a ação não foi executada pelo agente, então o estado de obediência da norma será *violado*. Todas as normas são iniciadas no sistema com o estado de obediência *desconhecido*. Os indivíduos *Fulfilled*, *Violated* e *Unknown* foram adicionados na ontologia de Norma e a classe *FulfillmentStatus* foi definida pela enumeração de seus membros, como segue.

```
FulfillmentStatus ≡ { Fulfilled } ⊔ { Violated } ⊔
```

```
{ Unknown }
```

3.1.1.9 DEFINIÇÃO DAS CLASSES QUE CLASSIFICAM A NORMA QUANTO AO SEU CUMPRIMENTO

Na ontologia de Norma, foram definidas cinco classes para classificar a norma quanto ao seu cumprimento, ou seja, para categorizar a norma em cumprida ou violada. Estas classes são posteriormente usadas na definição de algumas regras de inferência SWRL, que serão apresentadas na Seção 3.2. Portanto, foram definidas as seguintes classes na ontologia de Norma: *FulfilledObligationNorm*, *ViolatedObligationNorm*, *FulfilledProhibitionNorm*, *ViolatedProhibitionNorm* e *ViolatedPermissionNorm*. O cumprimento de uma norma depende do seu conceito deôntico. Quando a norma é uma obrigação sobre uma ação *AC* aplicada a um agente *AG*, o cumprimento desta norma ocorre quando ela está ativa e a ação *AC* foi executada pelo agente *AG*, e a violação acontece quando a norma foi desativada, porém a ação não foi executada pelo agente durante o período em que a norma esteve ativa. Caso a norma seja uma proibição sobre a ação *AC* aplicada ao agente *AG*, o cumprimento desta norma ocorre de forma contrária ao cumprimento da obrigação. Ela é violada quando a mesma está ativa e a ação *AC* foi executada pelo agente *AG*, e cumprida quando a norma foi desativada, porém a ação não foi executada pelo agente durante o período em que a norma esteve ativa. No caso de uma norma cujo conceito deôntico é uma permissão sobre a ação *AC* aplicada ao agente *AG*, a violação acontece quando a norma não está ativa (não existe a permissão) e a ação *AC* foi executada pelo agente *AG*. No entanto, uma permissão não pode ser cumprida visto que o agente pode executar ou não a ação enquanto a norma estiver ativa. As normas onde o conceito deôntico é uma permissão descrevem as ações que são autorizadas, mas não esperadas de serem executas por um agente. A seguir seguem as definições das classes *FulfilledObligationNorm*, *ViolatedObligationNorm*, *FulfilledProhibitionNorm*, *ViolatedProhibitionNorm* e *ViolatedPermissionNorm*.

```
FulfilledObligationNorm ⊑ Norm ⊓
          hasDeonticConcept.{Obligation} ⊓
          hasFulfillmentStatus.{Fulfilled}

ViolatedObligationNorm ⊑ Norm ⊓
          hasDeonticConcept.{Obligation} ⊓
          hasActivationStatus.{Deactivated} ⊓
          ¬ FulfilledObligationNorm

FulfilledProhibitionNorm ⊑ Norm ⊓
          hasDeonticConcept.{Prohibition} ⊓
```

```
          hasActivationStatus.{Deactivated} ⊓
          ¬ ViolatedProhibitionNorm

ViolatedProhibitionNorm ⊑ Norm ⊓
          hasDeonticConcept.{Prohibition} ⊓
          hasFulfillmentStatus.{Violated}

ViolatedPermissionNorm ⊑ Norm ⊓
          hasDeonticConcept.{Permission} ⊓
          hasFulfillmentStatus.{Violated}
```

A classe *FulfilledObligationNorm* é definida como uma norma cujo conceito deôntico é uma obrigação e tenha o valor *Fulfilled* para a propriedade *hasFulfillmentStatus*. Esse valor é atualizado através de uma regra SWRL, apresentada na Seção 3.2.1. A classe *ViolatedObligationNorm* é definida como uma norma cujo conceito deôntico é uma obrigação, que não esteja mais ativa e que não tenha sido cumprida, ou seja, não foi classificada como *FulfilledObligationNorm*. A classe *FulfilledProhibitionNorm* é definida como uma norma cujo conceito deôntico é uma proibição, que não esteja mais ativa e que não tenha sido violada, ou seja, não foi classificada como *ViolatedProhibitionNorm*. A classe *ViolatedProhibitionNorm* é definida como uma norma cujo conceito deôntico é uma proibição e tenha o valor *Violated* para a propriedade *hasFulfillmentStatus*. Esse valor é atualizado através de uma regra SWRL, mostrada na Seção 3.2.2.

Por fim, a classe *ViolatedPermissionNorm* é definida como uma norma cujo conceito deôntico é uma permissão, que não esteja mais ativa e tenha o valor *Violated* para a propriedade *hasFulfillmentStatus*. Esse valor é atualizado através de uma regra SWRL, apresentada na Seção 3.2.2. É importante observar que, para que uma permissão seja violada, o agente tem que ter executado a ação em algum momento no sistema multi-agente, mas que não tenha sido durante o período em que a permissão esteve ativa. O mesmo não é necessário para a violação de uma obrigação (*ViolatedObligationNorm*) ou para o cumprimento de uma proibição (*FulfilledProhibitionNorm*), visto que essas classes não exigem que a ação tenha sido executada em algum momento pelo agente.

3.1.2 ONTOLOGIA DE CENÁRIO DE EXECUÇÃO

A ontologia de Norma permite a representação de um agente executando uma ação e participando de uma situação, e a representação de uma norma sendo cumprida, violada, ativada e desativada. Porém, para que o mecanismo de detecção de conflitos consiga identificar os conflitos que dependem da ordem de execução dos eventos nos SMA, não é suficiente saber somente que tais eventos ocorreram no sistema. Mais que isso, é preciso saber quando que tais

eventos foram executados no sistema e se eles aconteceram antes ou depois de outro. Em outras palavras, se soubéssemos o momento em que cada evento mencionado na condição das normas acontecesse no sistema, seria possível garantir se uma norma estará em conflito ou não.

Sendo assim, a ontologia de Cenário de Execução estende a ontologia de Norma com o objetivo de adicionar a noção de *tempo*. O momento em que uma condição de uma norma acontece no sistema é capturado pela propriedade do tipo datatype chamada *hasConditionTime* e é representada por um valor inteiro (xsd:integer). Portanto, o seguinte axioma foi adicionado na classe *Condition*.

$$\text{Condition} \sqsubseteq \forall\ \text{hasConditionTime.Integer}$$

Neste trabalho, adotou-se tempos discretos, por simplificação, representados por um valor inteiro (xsd:integer). Na ontologia de Cenário de Execução, foram definidas três principais classes. São elas: *Time*, *PositionInInterval* e *ActivationPeriod*, que são apresentadas nas próximas subseções.

3.1.2.1 DEFINIÇÃO DA CLASSE TEMPO

A ontologia de Cenário de Execução introduziu a classe *Time* para representar o momento em que uma ação é executada por um agente ou o momento em que uma situação se torna verdade no sistema para um agente. Esses momentos são representados por um valor inteiro através da propriedade do tipo datatype chamada *hasTime*, cujos *rdfs:domain* e *rdfs:range* é a classe *Time* e o *xsd:integer*, respectivamente. A classe *Time* é definida como segue.

$$\begin{aligned}\text{Time} \sqsubseteq\ & \forall\ \text{hasTime.Integer} \sqcap \\ & =1\ \text{hasPosition.PositionInInterval} \sqcap \\ & (\forall\ \text{timeEntity.Entity} \sqcup \\ & \ \forall\ \text{timeAction.Action} \sqcup \\ & \ \forall\ \text{timeSituation.Situation})\end{aligned}$$

A propriedade do tipo objeto *hasPosition* relaciona uma instância da classe *Time* com uma instância da classe *PositionInInterval*. A classe *PositionInInterval* define se uma instância da classe *Time* aconteceu dentro ou fora do intervalo de ativação da norma. A classe *PositionInInterval* será apresentada na Seção 3.1.2.2 e o período de ativação da norma será apresentado na Seção 3.1.2.3.

É imponte observar que o tempo de uma ação (ou situação) executada por um agente é único no sistema, ou seja, cada vez que um agente executa uma ação em um determinado tempo, uma nova instância da classe *Time* é criada no sistema.

As propriedades *entityTime*, *actionTime* e *situationTime* foram definidas na ontologia como propriedades inversas das propriedades *timeEntity*, *timeAction* e *timeSituation*, respectivamente. Tais propriedades serão usadas em regras SWRL, apresentadas na Seção 3.2.

$$entityTime \equiv timeEntity^{-}$$

$$actionTime \equiv timeAction^{-}$$

$$situationTime \equiv timeSituation^{-}$$

O tempo na ontologia de cenário de execução pode ser classificado como *TimeInsideInterval*, *TimeUnknownInterval* ou *TimeOutsideInterval*, definidos como segue.

$$TimeInsideInterval \sqsubseteq Time \sqcap hasPosition.\{InsidePosition\}$$

$$TimeUnknownInterval \sqsubseteq Time \sqcap hasPosition.\{UnknownPosition\}$$

$$TimeOutsideInterval \sqsubseteq Time \sqcap \neg\, TimeInsideInterval \sqcap \neg\, TimeUnknownInterval$$

A classe *TimeInsideInterval* é definida como uma subclasse de *Time* que tenha o valor *InsidePosition* para a propriedade *hasPosition*. Esse valor é atualizado através de uma regra SWRL, apresentada na Seção 3.2.2. A classe *TimeUnknownInterval* é definida como uma subclasse de *Time* que tenha o valor *UnknownPosition* para a propriedade *hasPosition*. Instâncias da classe *Time* têm posição desconhecida quando a ação foi executada no sistema, mas ainda não se sabe se foi dentro ou fora do intervalo. Isso acontece quando não conseguimos determinar se a ação foi executada entre o tempo inicial e o tempo final do intervalo de ativação da norma, pois tais tempos (que representam eventos) não foram fornecidos.

Por fim, a classe *TimeOutsideInterval* é um *Time* que tenha acontecido fora do período de ativação da norma, ou seja, não foi classificada como *TimeInsideInterval* e além disso não é um *TimeUnknownInterval*. Essa classificação (*TimeOutsideInterval*) é importante para detectar se uma norma, cujo conceito deôntico é uma permissão, foi violada. Tal violação será apresentada na Seção 3.2.2.

3.1.2.2 DEFINIÇÃO DA CLASSE POSIÇÃO NO INTERVALO

A classe *PositionInInterval* da ontologia de Cenário de Execução representa a posição do tempo, instância da classe *Time*, da execução de uma ação por um agente relativa ao intervalo de ativação da norma. Em outras palavras, a classe *PositionInInterval* indica se a ação foi executada dentro ou fora do intervalo de ativação da norma. Essa posição pode ser dentro do intervalo, fora do intervalo ou posição desconhecida.

Por exemplo, a Figura 4 ilustra um agente *AG* executando uma ação *A* três vezes no sistema multi-agentes em tempos diferentes: tempo *T1*, *T2* e *T3*. Pode-se observar que o tempo *T2* = 15 aconteceu dentro do período de ativação da norma *N1*, ou seja, quando *N1* estava ativa. No entanto, a execução da ação nos tempos *T1* = 5 e *T3* = 25 aconteceram fora do período de aviação de *N1*, ou seja, quando a norma não estava ativa. Como mencionado acima, essa classificação é importante para detectar se uma norma, cujo conceito deôntico é uma permissão, foi violada ou não, e será apresentada na Seção 3.2.2.

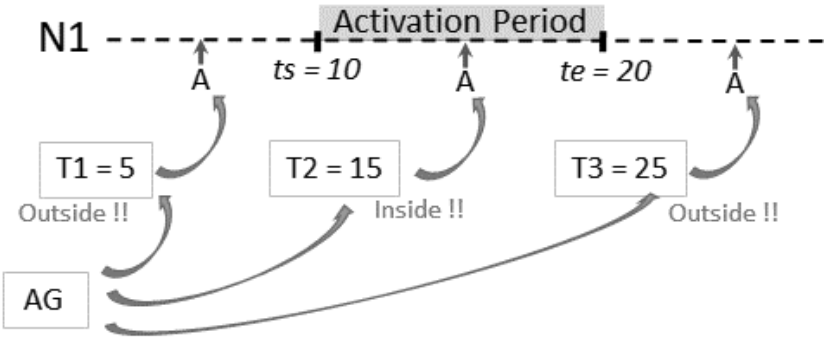


Figura 4: Execução de uma ação por um agente em diferentes tempos.

Sendo assim, os indivíduos *InsidePosition*, *OutsidePosition*, *UnknownPosition* foram adicionados na ontologia de Cenário de Execução e a classe *PositionInInterval* foi definida pela enumeração de seus membros, como segue.

$$\texttt{PositionInInterval} \equiv \{\texttt{InsidePosition}\} \sqcup \{\texttt{OutsidePosition}\} \sqcup \{\texttt{UnknownPosition}\}$$

3.1.2.3 DEFINIÇÃO DA CLASSE PERÍODO DE ATIVAÇÃO

A classe *ActivationPeriod* representa um período na linha do tempo em que a norma esteve ativa. De acordo com os tipos de intervalos de ativação de uma norma, ilustrados na Figura 3, a norma pode ter um e no máximo dois períodos de ativação, como observado no intervalo tipos 4 da Figura 3. A classe *ActivationPeriod* possui duas propriedades do tipo datatype chamadas *hasStart* e *hasEnd*, cujos *rdfs:range* são um xsd:integer, para representar o tempo inicial e final do período de ativação da norma, respectivamente. Esses tempos são importantes para identificar se uma ação foi executada durante o período em que a norma estava ativa, ou seja, entre o tempo de início e fim de seu intervalo de ativação. Essa informação será usada posteriormente em regras de inferências SWRL, nas seções 3.2.1 e 3.2.2. A seguir segue a definição em DL da classe *ActivationPeriod* e o axioma adicionado na classe *Norm*.

$$\texttt{ActivationPeriod} \sqsubseteq =1\ \texttt{hasStart.Integer} \sqcap =1\ \texttt{hasEnd.Integer} \sqcap =1\ \texttt{actPrdName.String}$$

$\mathtt{Norm} \sqsubseteq \leq 2\ \mathtt{hasActvPrd.ActivationPeriod}$

A propriedade do tipo datatype chamada *actPrdName* é usada para representar parte da justificativa do potencial conflito normativo. Tal justificativa é apresentada na Seção 3.3.1.1.

As classes e propriedades da ontologia de Cenário e Execução é graficamente mostrada na Figura 5.

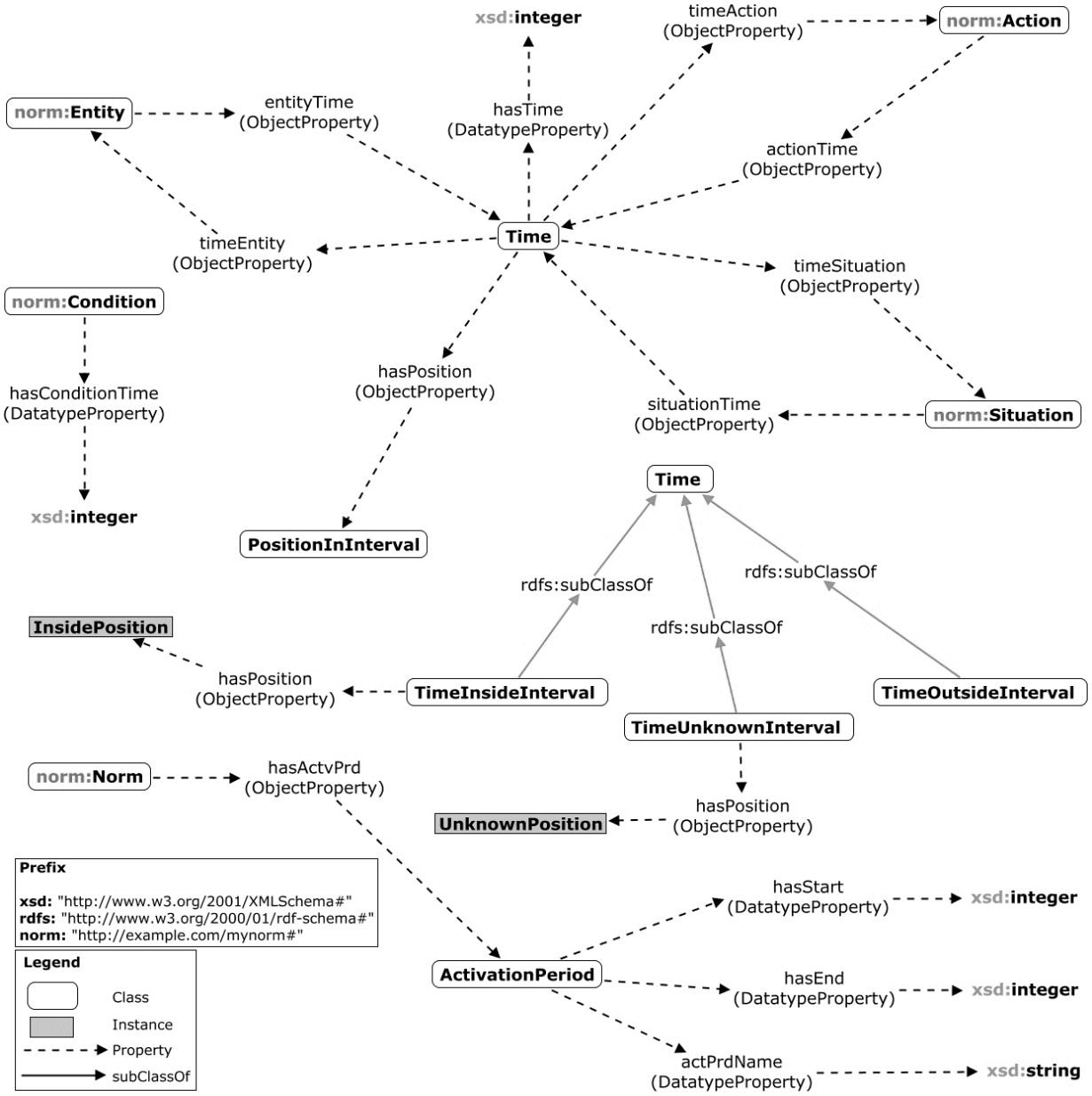


Figura 5: Representação gráfica da ontologia de Cenário de Execução, adaptado de BELCHIOR e DA SILVA (2017c).

3.2 APLICAÇÃO DAS REGRAS SWRL

Regras na Web Semântica são um meio de representar o conhecimento por meio de inferências que geralmente vão além da representação usada pela linguagem OWL (*Web*

Ontology Language) (HEBELER *et al.*, 2009, p. 232). Elas são tipicamente sentenças condicionais do tipo “se-então”, mais formalmente referidas como implicação. Quando um conjunto particular de sentenças, especificadas no antecedente de uma regra, for verdadeiro, novos conhecimentos, especificados no consequente da regra, são adicionados à base de conhecimento.

Como apresentado na Seção 2.3, SWRL é uma linguagem de regras baseada em OWL DL e OWL Lite, e um subconjunto da linguagem RuleML (*Rule Markup Language*), que modela *cláusulas Horn* (HORROCKS *et al.*, 2004). O subconjunto de RuleML suportado por SWRL inclui somente predicados unários e binários. Uma das vantagens de se usar SWRL é o seu suporte a funções incorporadas para executar operações de comparações, matemática, *strings*, data e outros.

Como vimos na Seção 3.1.1.6, seis tipos de condições, responsáveis por determinar o período durante o qual uma norma fica ativa, foram definidos como subclasses da classe *Condition*: *ExecutionOfAction*, *ActivationOfNorm*, *DeactivationOfNorm*, *FulfillmentOfNorm*, *ViolationOfNorm* e *SituationParticipation*. Regras SWRL foram usadas para calcular o tempo em que cada uma destas condições acontece no sistema. Além disso, regras em SWRL foram usadas para fazer a verificação dos conflitos normativos entre uma obrigação e proibição e entre uma permissão e proibição. Após o cálculo dos tempos das condições das normas, as regras responsáveis por fazer as detecções dos conflitos podem ser executadas. Nas próximas subseções serão apresentadas como que tais inferências são realizadas.

3.2.1 CÁLCULO DO TEMPO DA CONDIÇÃO FULFILLMENT OF NORM

A condição *FulfillmentOfNorm* refere-se ao cumprimento de uma norma por um agente. Uma norma, cujo conceito deôntico é uma obrigação, é cumprida quando a ação regulada por tal norma é executada pelo agente sendo controlado durante o período em que a norma esteve ativa, ou seja, entre o tempo inicial e final do período de ativação da norma. A regra do Quadro 1 verifica se uma obrigação foi cumprida por um agente e, caso seja, atualiza o tempo da condição *FulfillmentOfNorm*. Em outras palavras, ela testa se uma ação *?a* foi executada por um agente *?e* (linha 5), ambos definidos por uma norma *?n*, cujo conceito deôntico é uma obrigação (linha 3), e se *?a* aconteceu durante o período em que *?n* esteve ativa, ou seja, se *?ti* aconteceu depois de *?ts* e antes de *?te* através do uso de funções embutidas de comparação da SWRL (linha 8) *swrlb:greaterThan OrEqual(?ti, ?ts)* e *swrlb:lessThan(?ti, ?te)*, onde *?ti* é o momento em que a ação *?a* foi executada no sistema, e *?ts* e *?te* são os tempos inicial e final

do período de ativação da norma, respectivamente. Como resultado da regra, o valor da propriedade *hasConditionTime* é atualizado com o valor da variável *?ti* (linha 9).

1	FulfillmentOfNorm(?c) ∧
2	hasRelatedNorm(?c, ?n) ∧ Norm(?n) ∧
3	hasDeonticConcept(?n, Obligation) ∧
4	hasAction(?n, ?a) ∧ Action(?a) ∧
5	hasEntity(?n, ?e) ∧ Entity(?e) ∧ performsAction(?e, ?a) ∧
6	entityTime(?e, ?t) ∧ Time(?t) ∧ timeAction(?t, ?a) ∧ hasTime(?t, ?ti) ∧
7	hasActvPrd(?n, ?ap) ∧ hasStart(?ap, ?ts) ∧ hasEnd(?ap, ?te) ∧
8	swrlb:greaterThanOrEqual(?ti, ?ts) ∧ swrlb:lessThan(?ti, ?te)
9	⟶ hasConditionTime(?c, ?ti) ∧ hasFulfillmentStatus(?n, Fulfilled)

Quadro 1. Regra para calcular o tempo da condição *FulfillmentOfNorm* relacionada com uma norma cujo conceito deôntico é uma obrigação.

A regra mostrada no Quadro 2 atualiza o tempo da condição *FulfillmentOfNorm* quando ela está relacionada com uma norma cujo conceito deontico é uma proibição. Quando o conceito deôntico da norma é uma proibição, o seu cumprimento ocorre quando a norma foi desativada, porém a ação não foi executada pelo agente durante o período em que a norma esteve ativa, ou seja, a norma não foi violada e foi classificada como *FulfilledProhibitionNorm* (linha 3). A Seção 3.1.1.9 mostra quando uma norma é classificada como *FulfilledProhibitionNorm*. Caso tal verificação seja verdade, o valor da propriedade *hasConditionTime* é atualizado com o valor da variável *?te* (linha 5), onde *?te* é o tempo final do período de ativação da norma e representa o momento que a norma deixou de estar ativa.

1	FulfillmentOfNorm(?c) ∧
2	hasRelatedNorm(?c, ?n) ∧
3	FulfilledProhibitionNorm(?n) ∧
4	hasActvPrd(?n, ?ap) ∧ hasEnd(?ap, ?te)
5	⟶ hasConditionTime(?c, ?te)

Quadro 2. Regra para calcular o tempo da condição *FulfillmentOfNorm* relacionada com uma norma cujo conceito deôntico é uma proibição.

Como mencionado na Seção 3.1.1.9, uma norma cujo conceito deôntico é uma permissão não pode ser cumprida, pois o agente pode executar ou não a ação enquanto a norma estiver ativa. As normas onde o conceito deôntico é uma permissão descrevem as ações que são autorizadas, mas não esperadas de serem executas por um agente.

3.2.2 CÁLCULO DO TEMPO DA CONDIÇÃO VIOLATION OF NORM

A condição *ViolationOfNorm* representa a violação de uma norma por um agente. Uma norma, cujo conceito deôntico é uma obrigação, é violada quando ela foi desativada, porém a ação regulada não foi executada pelo agente durante o período em que a norma estava ativa, ou

seja, a norma não foi cumprida e foi classificada como *ViolatedObligationNorm* (Quadro 3 - linha 3). A Seção 3.1.1.9 mostra quando uma norma é classificada como *ViolatedObligationNorm*. Caso tal verificação seja verdade, o valor da propriedade *hasConditionTime* é atualizado com o valor da variável *?te* (Quadro 3 - linha 5), onde *?te* é o tempo final do período de ativação da norma e representa o momento que a norma deixou de estar ativa. A regra do Quadro 3 atualiza o tempo da condição *ViolationOfNorm* quando ela está relacionada com uma norma cujo conceito deôntico é uma obrigação.

1	ViolationOfNorm(?c) ∧
2	hasRelatedNorm(?c, ?n) ∧
3	ViolatedObligationNorm(?n) ∧
4	hasActvPrd(?n, ?ap) ∧ hasEnd(?ap, ?te)
5	⟶ hasConditionTime(?c, ?te)

Quadro 3. Regra para calcular o tempo da condição *ViolationOfNorm* relacionada com uma norma cujo conceito deôntico é uma obrigação.

No caso de a norma ser uma proibição, a violação acontece quando a ação regulada por tal norma é executada pelo agente sendo controlado durante o período em que a norma esteve ativa, ou seja, entre o tempo inicial e final do período de ativação da norma. A regra do Quadro 4 verifica se uma proibição foi violada por um agente e, caso seja, atualiza o tempo da condição *ViolationOfNorm*. Em outras palavras, ela testa se uma ação *?a* foi executada por um agente *?e* (linha 5), ambos definidos por uma norma *?n*, cujo conceito deôntico é uma proibição (linha 3), e se *?a* aconteceu durante o período em que *?n* esteve ativa, ou seja, se *?ti* aconteceu depois de *?ts* e antes de *?te* através do uso de funções embutidas de comparação da SWRL (linha 8), onde *?ti* é o momento que a ação *?a* foi executada no sistema, e *?ts* e *?te* são os tempos inicial e final do período de ativação da norma, respectivamente. Como resultado da regra, o valor da propriedade *hasConditionTime* é atualizado com o valor da variável *?ti* (linha 9).

1	ViolationOfNorm (?c) ∧
2	hasRelatedNorm(?c, ?n) ∧ Norm(?n) ∧
3	hasDeonticConcept(?n, Prohibition) ∧
4	hasAction(?n, ?a) ∧ Action(?a) ∧
5	hasEntity(?n, ?e) ∧ Entity(?e) ∧ performsAction(?e, ?a) ∧
6	entityTime(?e, ?t) ∧ Time(?t) ∧ timeAction(?t, ?a) ∧ hasTime(?t, ?ti) ∧
7	hasActvPrd(?n, ?ap) ∧ hasStart(?ap, ?ts) ∧ hasEnd(?ap, ?te) ∧
8	swrlb:greaterThanOrEqual(?ti, ?ts) ∧ swrlb:lessThan(?ti, ?te)
9	⟶ hasConditionTime(?c, ?ti) ∧ hasFulfillmentStatus(?n, Violated)

Quadro 4. Regra para calcular o tempo da condição *ViolationOfNorm* relacionada com uma norma cujo conceito deôntico é uma proibição.

No caso da violação de uma permissão, ela acontece quando a norma não está ativa (não existe a permissão) e a ação regulada pela norma foi executada pelo agente sendo controlado.

A regra do Quadro 5 calcula o tempo da condição *ViolationOfNorm* relacionada com uma norma cujo conceito deôntico é uma permissão. A regra verifica se uma ação *?a* foi executada por um agente *?e* (linha 5), ambos definidos por uma norma *?n*, cujo conceito deôntico é uma permissão (linha 3), e se *?a* aconteceu durante o período em que *?n não* estava ativa, ou seja, o momento *?t* que *?e* executou a ação *?a* foi classificado como *TimeOutsideInterval* (linha 7). A definição da classe *TimeOutsideInterval* foi apresentada na Seção 3.1.2.1. Como resultado da regra, o valor da propriedade hasConditionTime é atualizado com o valor da variável *?ti* (linha 8), onde *?ti* é o momento que a ação *?a* foi executada no sistema.

1	ViolationOfNorm(?c) ∧
2	hasRelatedNorm(?c, ?n) ∧ Norm(?n) ∧
3	hasDeonticConcept(?n, Permission) ∧
4	hasAction(?n, ?a) ∧ Action(?a) ∧
5	hasEntity(?n, ?e) ∧ Entity(?e) ∧ performsAction(?e, ?a) ∧
6	entityTime(?e, ?t) ∧ Time(?t) ∧ timeAction(?t, ?a) ∧ hasTime(?t, ?ti) ∧
7	TimeOutsideInterval(?t)
8	⟶ hasConditionTime(?c, ?ti) ∧ hasFulfillmentStatus(?n, Violated)

Quadro 5. Regra para calcular o tempo da condição *ViolationOfNorm* relacionada com uma norma cujo conceito deôntico é uma permissão.

Para que um tempo, instância da classe Time da ontologia de cenário de execução, seja classificado como *TimeInsideInterval* ou *TimeOutsideInterval* corretamente, é preciso atualizar o valor da propriedade do tipo objeto *hasPosition* para *InsidePosition* quando uma ação governada por uma norma foi executada pelo agente sendo controlado durante o período em que a norma esteve ativa. As classes *TimeInsideInterval* e *TimeOutsideInterval* foram apresentadas na Seção 3.1.2.1. A regra do Quadro 6 faz essa atualização. Ela verifica se uma ação *?a* foi executada por um agente *?e* (linha 3), ambos definidos por uma norma *?n* (linha 1), independentemente de seu conceito deôntico, e se *?a* aconteceu durante o período em que *?n* estava ativa (linha 6). Como consequente da regra, o valor da propriedade *hasPosition* é atualizado para *InsidePosition*.

1	Norm(?n) ∧
2	hasAction(?n, ?a) ∧ Action(?a) ∧
3	hasEntity(?n, ?e) ∧ Entity(?e) ∧ performsAction(?e, ?a) ∧
4	agentTime(?e, ?t) ∧ Time(?t) ∧ timeAction(?t, ?a) ∧ hasTime(?t, ?ti) ∧
5	hasActvPrd(?n, ?ap) ∧ hasStart(?ap, ?ts) ∧ hasEnd(?ap, ?te) ∧
6	greaterThanOrEqual(?ti, ?ts) ∧ lessThan(?ti, ?te)
7	⟶ hasPosition(?t, InsidePosition)

Quadro 6. Regra para atualizar o valor da propriedade *hasPosition* para *InsidePosition*.

3.2.3 CÁLCULO DO TEMPO DA CONDIÇÃO ACTIVATION OF NORM

A condição *ActivationOfNorm* refere-se à condição da norma que captura o momento que uma norma foi ativada no sistema. Tal momento é determinado pelo começo do intervalo de ativação da norma. A regra do Quadro 7 calcula o tempo da condição *ActivationOfNorm* de uma norma. Ela verifica se uma norma *?n* relacionada com uma condição *?c* tem o tempo inicial *?ts* do seu período de ativação definido (linha 3) e, caso tenha, o valor da propriedade *hasConditionTime* é atualizado com esse valor (linha 4). Quando há mais de um intervalo de ativação na norma, todos os tempos de ativação da norma serão capturados pela regra.

1	ActivationOfNorm(?c) ∧
2	hasRelatedNorm(?c, ?n) ∧ Norm(?n) ∧
3	hasActvPrd(?n, ?ap) ∧ hasStart(?ap, ?ts)
4	⟶ hasConditionTime(?c, ?ts)

Quadro 7. Regra para calcular o tempo da condição *ActivationOfNorm*

3.2.4 CÁLCULO DO TEMPO DA CONDIÇÃO DEACTIVATION OF NORM

A condição *DeactivationOfNorm* representa o momento que uma norma foi desativada no sistema. Tal momento é determinado pelo tempo de término do intervalo de ativação da norma. A regra do Quadro 8 calcula o tempo da condição *DeactivationOfNorm* de uma norma. A regra verifica se uma norma *?n* relacionada com uma condição *?c* tem o tempo de término *?te* do seu período de ativação definido (linha 3) e, caso tenha, o valor da propriedade *hasConditionTime* é atualizado com esse valor (linha 4). Assim como na condição *ActivationOfNorm*, se a norma tiver mais de um intervalo de ativação, todos os tempos de desativação da norma serão capturados por esta regra.

1	DeactivationOfNorm(?c) ∧
2	hasRelatedNorm(?c, ?n) ∧ Norm(?n) ∧
3	hasActvPrd(?n, ?ap) ∧ hasEnd(?ap, ?te)
4	⟶ hasConditionTime(?c, ?te)

Quadro 8. Regra para calcular o tempo da condição *DeactivationOfNorm*

3.2.5 CÁLCULO DO TEMPO DA CONDIÇÃO EXECUTION OF ACTION

A condição *ExecutionOfAction* refere-se à execução de uma ação por um agente. O tempo da condição *ExecutionOfAction* é atualizado a partir do valor da propriedade *hasTime* da classe *Time*, relacionada com uma entidade e uma ação. Esse valor é passado pelo designer do sistema, como será visto na primeira abordagem de detecção de conflitos, Seção 3.3.1. A regra do Quadro 9 verifica as ações executadas pelo agente (linha 3) e checa se o tempo da execução

desta ação (*?ti*) foi fornecida pelo designer (linha 4). Caso seja, o valor da propriedade *hasConditionTime* é atualizado com o valor desta varável (linha 5).

1	ExecutionOfAction(?c) ∧
2	hasRelatedAction(?c, ?a) ∧ Action(?a) ∧
3	hasRelatedEntity(?c, ?e) ∧ Entity(?e) ∧ performsAction(?e, ?a) ∧
4	entityTime(?e, ?t) ∧ Time(?t) ∧ timeAction(?t, ?a) ∧ hasTime(?t, ?ti)
5	⟶ hasConditionTime(?c, ?ti)

Quadro 9. Regra para calcular o tempo da condição *ExecutionOfAction*

3.2.6 CÁLCULO DO TEMPO DA CONDIÇÃO SITUATION PARTICIPATION

A condição *SituationParticipation* representa a participação de uma situação por um agente. O tempo da condição *SituationParticipation* é atualizado a partir do valor da propriedade *hasTime* da classe *Time*, relacionada com uma entidade e uma situação. Esse valor também é passado pelo designer do sistema, como será apresentado na primeira abordagem de detecção de conflitos, Seção 3.3.1. A regra do Quadro 10 verifica as situações participadas pelo agente (linha 3) e checa se o tempo que o agente participou da situação (*?ti*) foi fornecido pelo designer (linha 4). Caso seja, o valor da propriedade *hasConditionTime* é atualizado com o valor da variável *?ti* (linha 5).

1	SituationParticipation(?c) ∧
2	hasRelatedSituation(?c, ?s) ∧ Situation(?s) ∧
3	hasRelatedEntity(?c, ?e) ∧ Entity(?e) ∧ participatesIn(?e, ?s) ∧
4	entityTime(?e, ?t) ∧ Time(?t) ∧ timeSituation(?t, ?s) ∧ hasTime(?t, ?ti)
5	⟶ hasConditionTime(?c, ?ti)

Quadro 10. Regra para calcular o tempo da condição *SituationParticipation*

3.3 DETECÇÃO DO POTENCIAL CONFLITO NORMATIVO

A detecção dos conflitos normativos que ocorrem em tempo de execução depende da ordem de execução dos eventos no sistema multi-agentes. Neste trabalho, foram propostas duas abordagens em *tempo de design* baseadas em cenários de execução para detectar *Potenciais Conflitos Normativos* (PCN) no sistema multi-agentes. Como mencionado anteriormente, potenciais conflitos normativos são conflitos que *podem* acontecer entre duas normas durante a execução do sistema multi-agente, dependendo da ordem de execução dos eventos do sistema referidos nas definições das normas.

Na primeira abordagem (BELCHIOR e DA SILVA, 2017a, 2017b), o designer fornece ao sistema normativo um conjunto de normas e possíveis exemplos de cenários de execução do

sistema. Como saída, o sistema normativo retorna os conflitos normativos que aconteceriam caso tais cenários fossem executados no sistema multi-agentes.

Na segunda abordagem (BELCHIOR e DA SILVA, 2017c), o designer do sistema fornece somente um conjunto de normas e o sistema normativo gera os cenários de execução que causariam os conflitos normativos caso tais cenários aconteçam no sistema multi-agentes. Ambas as abordagens fazem a detecção do conflito entre pares de normas.

Nas próximas subseções serão apresentadas em detalhes essas duas abordagens de detecção de conflitos.

3.3.1 DETECÇÃO BASEADA EM EXEMPLOS DE CENÁRIO DE EXECUÇÃO

A primeira abordagem de detecção de conflitos permite o designer do sistema fornecer possíveis sequências de ações e/ou situações no sistema e avaliar se tais sequências causariam algum potencial conflito normativo no sistema multi-agentes. Para tal, o designer do sistema fornece um exemplo esperado de cenário de execução do sistema e um conjunto de normas. Em seguida, o sistema normativo verifica, a partir do conjunto de normas, se tal cenário causaria algum potencial conflito normativo caso a execução do sistema acontecesse tal como o cenário de execução passado pelo designer. A detecção do conflito é feita entre pares de normas. No cenário de execução, o designer do sistema somente precisa fornecer as seguintes informações:

- Tempo de quando certa ação é executada por um agente, e/ou
- Tempo de quando certa situação torna-se verdade para um agente.

Os demais tempos, tempo da ativação e desativação, cumprimento e violação de uma norma são automaticamente inferidos pelas regras SWRL descritas nas seções anteriores. Uma vez que todos os tempos das condições das normas forem inferidos, as regras responsáveis por fazer as detecções dos conflitos são executadas. Duas normas estão em conflito quando (i) há alguma interseção entre seus períodos de ativação, (ii) os conceitos deônticos das normas são contraditórios (isto é, obrigação *versus* proibição ou permissão *versus* proibição), (iii) elas estão regulando uma mesma entidade e (iv) um mesmo comportamento, e (v) estão definidas em um mesmo contexto. A regra do Quadro 11 faz a detecção de conflitos normativos entre uma obrigação e uma proibição. Para isso, ela precisa comparar os períodos de ativação das normas de dois em dois a fim de encontrar interseções entre eles. Mais detalhadamente, a regra verifica se duas normas *?n1* e *?n2* estão associadas a uma mesma entidade *?e* (linha 2), regulam o mesmo comportamento *?a* (linha 3), estão definidas no mesmo contexto *?c* (linha 4) e tem conceitos deônticos contraditórios, no caso entre uma obrigação e uma proibição (linhas 5 e 6).

Após isso, a regra checa se o período de ativação *?ap1* da norma *?n1* (linha 7) tem interseção com o período de ativação *?ap2* da norma *?n2* (linha 8) pela comparação de seus tempos iniciais e finais (linha 9). A construção da justificativa do conflito é feita pela função embutida *swrlb:stringConcat* na linha 12 que faz a concatenação de *strings* e dos valores das variáveis *?nameN1*, *?nameN2*, *?just1* e *?just2* para formar a justificativa do conflito, onde *?nameN1* e *?nameN2* são os nomes das normas *?n1* e *?n2*, respectivamente, e *?just1* e *?just2* contêm parte da justificativa do conflito normativo. Mais especificamente, *?just1* e *?just2* contêm os tempos dos eventos do sistema que causariam o conflito. A construção destes tempos que compõem a justificativa do conflito será apresentada na Seção 3.3.1.1. O resultado da concatenação é uma explicação do conflito normativo mostrando a ordem de execução dos eventos que causou o conflito. Esse resultado é armazenado na variável *?confJust* (linha 12). Se todo o antecedente da regra for satisfeito, então um conflito é identificado e, como consequência da regra, o conflito entre as duas normas é definido através da propriedade do tipo objeto *hasConflict*, e a justificativa do conflito é registrada através da propriedade do tipo datatype chamada *conflictJustification* (linha 14).

1	Norm(?n1) ∧ Norm(?n2) ∧
2	hasEntity(?n1, ?e) ∧ hasEntity(?n2, ?e) ∧
3	hasAction(?n1, ?a) ∧ hasAction(?n2, ?a) ∧
4	hasContext(?n1, ?c) ∧ hasContext(?n2, ?c) ∧
5	**hasDeonticConcept (?n1, Obligation) ∧**
6	**hasDeonticConcept(?n2, Prohibition) ∧**
7	hasActvPrd(?n1, ?ap1) ∧ hasStart(?ap1, ?st1) ∧ hasEnd(?ap1, ?ed1) ∧
8	hasActvPrd(?n2, ?ap2) ∧ hasStart(?ap2, ?st2) ∧ hasEnd(?ap2, ?ed2) ∧
9	swrlb:lessThan(?st1, ?st2) ∧ swrlb:greaterThanOr Equal(?ed1, ?st2) ∧
10	normName(?n1, ?nameN1) ∧ normName(?n2, ?nameN2) ∧
11	actPrdName(?ap1, ?just1) ∧ actPrdName(?ap2, ?just2) ∧
12	swrlb:stringConcat(?confJust, “Conflict between ”, ?nameN1, “ and ”, ?nameN2,
13	“ if execution order is: ”, ?just1, “, ”, ?just2)
14	⟶ hasConflict(?n1, ?n2) ∧ conflictJustification(?n1, ?confJust)

Quadro 11. Regra para fazer a detecção de conflitos entre duas normas cujos conceitos deônticos são uma obrigação e uma proibição – Parte 1.

A regra do Quadro 12 mostra a mesma detecção de conflitos entre uma obrigação e uma proibição do Quadro 11, porém invertendo os conceitos deônticos das variáveis *?n1* e *?n2* (linhas 5 e 6). Isso se faz necessário para que a regra capture de forma contrária a interseção entre os períodos de ativação das normas, visto que em SWRL não é possível ter disjunção de átomos na mesma regra.

1	Norm(?n1) ∧ Norm(?n2) ∧
2	hasEntity(?n1, ?e) ∧ hasEntity(?n2, ?e) ∧
3	hasAction(?n1, ?a) ∧ hasAction(?n2, ?a) ∧
4	hasContext(?n1, ?c) ∧ hasContext(?n2, ?c) ∧
5	**hasDeonticConcept (?n1, Prohibition) ∧**
6	**hasDeonticConcept(?n2, Obligation) ∧**
7	hasActvPrd(?n1, ?ap1) ∧ hasStart(?ap1, ?st1) ∧ hasEnd(?ap1, ?ed1) ∧
8	hasActvPrd(?n2, ?ap2) ∧ hasStart(?ap2, ?st2) ∧ hasEnd(?ap2, ?ed2) ∧
9	swrlb:lessThan(?st1, ?st2) ∧ swrlb:greaterThanOr Equal(?ed1, ?st2) ∧
10	normName(?n1, ?nameN1) ∧ normName(?n2, ?nameN2) ∧
11	actPrdName(?ap1, ?just1) ∧ actPrdName(?ap2, ?just2) ∧
12	swrlb:stringConcat(?confJust, “Conflict between ”, ?nameN1, “ and ”, ?nameN2,
13	“ if execution order is: ”, ?just1, “, ”, ?just2)
14	⟶ hasConflict(?n1, ?n2) ∧ conflictJustification(?n1, ?confJust)

Quadro 12. Regra para fazer a detecção de conflitos entre duas normas cujos conceitos deônticos são uma obrigação e uma proibição – Parte 2.

As regras do Quadro 13 e Quadro 14 fazem a detecção de conflitos normativos entre duas normas cujos conceitos deônticos são uma permissão e uma proibição. O raciocínio destas duas regras é semelhante ao das regras do Quadro 11 e Quadro 12.

1	Norm(?n1) ∧ Norm(?n2) ∧
2	hasEntity(?n1, ?e) ∧ hasEntity(?n2, ?e) ∧
3	hasAction(?n1, ?a) ∧ hasAction(?n2, ?a) ∧
4	hasContext(?n1, ?c) ∧ hasContext(?n2, ?c) ∧
5	**hasDeonticConcept (?n1, Permission) ∧**
6	**hasDeonticConcept(?n2, Prohibition) ∧**
7	hasActvPrd(?n1, ?ap1) ∧ hasStart(?ap1, ?st1) ∧ hasEnd(?ap1, ?ed1) ∧
8	hasActvPrd(?n2, ?ap2) ∧ hasStart(?ap2, ?st2) ∧ hasEnd(?ap2, ?ed2) ∧
9	swrlb:lessThan(?st1, ?st2) ∧ swrlb:greaterThanOr Equal(?ed1, ?st2) ∧
10	normName(?n1, ?nameN1) ∧ normName(?n2, ?nameN2) ∧
11	actPrdName(?ap1, ?just1) ∧ actPrdName(?ap2, ?just2) ∧
12	swrlb:stringConcat(?confJust, “Conflict between ”, ?nameN1, “ and ”, ?nameN2,
13	“ if execution order is: ”, ?just1, “, ”, ?just2)
14	⟶ hasConflict(?n1, ?n2) ∧ conflictJustification(?n1, ?confJust)

Quadro 13. Regra para fazer a detecção de conflitos entre duas normas cujos conceitos deônticos são uma permissão e uma proibição – Parte 1.

1	Norm(?n1) ∧ Norm(?n2) ∧
2	hasEntity(?n1, ?e) ∧ hasEntity(?n2, ?e) ∧
3	hasAction(?n1, ?a) ∧ hasAction(?n2, ?a) ∧
4	hasContext(?n1, ?c) ∧ hasContext(?n2, ?c) ∧
5	**hasDeonticConcept (?n1, Prohibition) ∧**
6	**hasDeonticConcept(?n2, Permission) ∧**
7	hasActvPrd(?n1, ?ap1) ∧ hasStart(?ap1, ?st1) ∧ hasEnd(?ap1, ?ed1) ∧
8	hasActvPrd(?n2, ?ap2) ∧ hasStart(?ap2, ?st2) ∧ hasEnd(?ap2, ?ed2) ∧
9	swrlb:lessThan(?st1, ?st2) ∧ swrlb:greaterThanOr Equal(?ed1, ?st2) ∧
10	normName(?n1, ?nameN1) ∧ normName(?n2, ?nameN2) ∧
11	actPrdName(?ap1, ?just1) ∧ actPrdName(?ap2, ?just2) ∧
12	swrlb:stringConcat(?confJust, “Conflict between ”, ?nameN1, “ and ”, ?nameN2,
13	“ if execution order is: ”, ?just1, “, ”, ?just2)
14	⟶ hasConflict(?n1, ?n2) ∧ conflictJustification(?n1, ?confJust)

Quadro 14. Regra para fazer a detecção de conflitos entre duas normas cujos conceitos deônticos são uma permissão e uma proibição – Parte 2.

3.3.1.1 CONSTRUÇÃO DA JUSTIFICATIVA DO CONFLITO E ATUALIZAÇÃO DOS TEMPOS DE INÍCIO E FIM DE UM INTERVALO DE ATIVAÇÃO

A construção da justificativa do conflito foi apresentada na Seção 3.3.1. Porém, parte da justificativa que contém os tempos dos eventos que causariam o conflito é construída dependendo do tipo de intervalo de ativação da norma (Figura 3), e não pode ser inferida via regra SWRL, visto que em SWRL não é possível checar a existência ou não da instância de uma classe (ou de uma propriedade) (ELENIUS *et al.*, 2009). No caso, não é possível checar em SWRL se uma norma está relacionada com a classe *Condition* usando apenas uma das propriedades *hasBefore* ou *hasAfter*, usando ambas as propriedades ou simplesmente não está relacionada a nenhuma instância da classe *Condition*. A mesma checagem também é necessária quando os tempos de início e fim de um intervalo de ativação são atualizados. Tais verificações podem ser feitas através da *expressão de filtro* NOT EXISTS via consulta SPARQL (HARRIS *et al.*, 2013). SPARQL é uma linguagem de consulta para RDF. Como OWL DL é uma extensão do RDF (DEAN *et al.*, 2004), pode-se usar SPARQL para fazer consultas na ontologia OWL. A expressão NOT EXISTS é capaz de verificar a presença ou não de uma determinada sentença RDF na ontologia, ou seja, é possível verificar a presença ou não das propriedades *hasBefore* ou *hasAfter* como relacionamentos entre as classes *Norm* e *Condition*.

Portanto, assim como a construção da parte da justificativa que contém os tempos dos eventos que causariam o conflito, a atualização dos tempos de início e fim de um intervalo também é dependente do tipo de intervalo de ativação da norma. Para ficar mais claro, vamos analisar o Algoritmo 1. O algoritmo faz a verificação do tipo de intervalo de ativação de uma

norma *n*, passada como parâmetro de entrada, e atualiza a justificativa e os tempos de início e fim do intervalo. Os métodos *setActivatInterval1* (linhas 8, 10, 12, 16 e 18) e *setActivatInterval2* (linha 19) atualizam os tempos e justificativas do primeiro e do segundo (caso exista) intervalos de ativação da norma *n*, respectivamente, dependendo do seu tipo. Ambos os métodos recebem como entrada três parâmetros: (i) o tempo inicial (ii) e final do intervalo de ativação, e uma (iii) *string*, que representa parte da justificativa do conflito.

O algoritmo começa verificando se a norma não tem condição relacionada a ela (linha 7) e, caso não tenha, o intervalo de ativação é atualizado (linha 8) do tempo zero até o valor da variável *limT*, que representa o tempo final da execução do sistema multi-agente, informado pelo designer. Este valor é necessário para que seja possível fazer as comparações dos tempos nas regras SWRL, como mostradas nas seções anteriores. Além disso, a justificativa para este caso é colocada como segue: *n.name + "is Always Active"*, onde *n.name* é o nome da norma (linha 8). Na linha 9, o algoritmo verifica se a norma está associada a somente uma condição usando a propriedade *hasBefore*. Caso a norma esteja, o intervalo de ativação é atualizado do tempo zero até o valor da variável *cBefT* que representa o tempo da condição com a propriedade *hasBefore* (linha 10). A justificativa é colocada como: *cBefT + ": " + cBefN*, onde *cBefN* é o nome da condição com a propriedade *hasBefore* (linha 10). Na linha 11, é verificado se a norma está associada a somente uma condição usando a propriedade *hasAfter*. Caso esteja, o intervalo de ativação é atualizado partindo do tempo da condição com a propriedade *hasAfter*, representado pela variável *cAftT*, até o valor da variável *limT* (linha 12). A justificativa é estabelecida como *cAftT + ": " + cAftN*, onde *cAftN* é o nome da condição com a propriedade *hasAfter* (linha 12). Para normas associadas com duas condições, uma através do relacionamento *hasBefore* e a outra através do relacionamento *hasAfter* (linha 13), ela terá um intervalo de ativação caso *cBefT >= cAftT*, onde *cBefT* e *cAftT* são os tempos das condições com as propriedades *hasBefore* e *hasAfter*, respectivamente. Sendo assim, o tempo inicial do intervalo de ativação é atualizado com o valor da variável *cAftT*, e o tempo final, com o valor da variável *cBefT*. A justificativa é atualizada com o valor *cAftT + ": " + cAftN + ", " + cBefT + ": " + cBefN* (linha 16). Caso contrário, a norma terá dois intervalos de ativação. O primeiro intervalo vai do tempo zero até o valor da variável *cBefT* (linha 18) e o segundo intervalo começa com o valor da variável *cAftT* até *limT* (linha 19). A justificativa também é colocada como *cAftT + ": " + cAftN + ", " + cBefT + ": " + cBefN*, para ambos os intervalos.

O Algoritmo 1 será utilizado posteriormente pelo Algoritmo 2 para calcular os potenciais conflitos normativos entre pares de normas dado os cenários de execução passados pelo designer, apresentado na Seção 3.3.1.2.

```
1.  function updateNormIntervalsAndJustification(Norm n)
2.      limT ← Limit Time of MAS
3.      cBefT ← Time of Before Condition of Norm n
4.      cAftT ← Time of After Condition of Norm n
5.      cBefN ← Name of Before Condition of Norm n
6.      cAftN ← Name of After Condition of Norm n
7       if hasNoCondition(n) then
8.              n.setActivatInterval1(0, limT, n.name + " is Always Active")
9.      else if hasOnlyBeforeCondition(n) then
10.             n.setActivatInterval1(0, cBefT, cBefT + ": " + cBefN)
11.     else if hasOnlyAfterCondition(n) then
12.             n.setActivatInterval1(cAftT, limT, cAftT + ": " + cAftN)
13.     else if hasBeforeAndAfterCondition(n) then
14.             just ← cAftT + ": " + cAftN + ", " + cBefT + ": " + cBefN
15.             if cBefT >= cAftT then
16.                     n.setActivatInterval1(cAftT, cBefT, just)
17.             else
18.                     n.setActivatInterval1(0, cBefT, just)
19.                     n.setActivatInterval2(cAftT, limT, just)
20.             end if
21.     end if
22. end function
```

Algoritmo 1. Algoritmo para construir a justificativa do conflito e para atualizar os tempos de início e fim dos intervalos de ativação das normas (BELCHIOR e DA SILVA, 2017c).

3.3.1.2 NORMAS DEFINIDAS EM CASCATA E CONFLITOS EM CASCATA

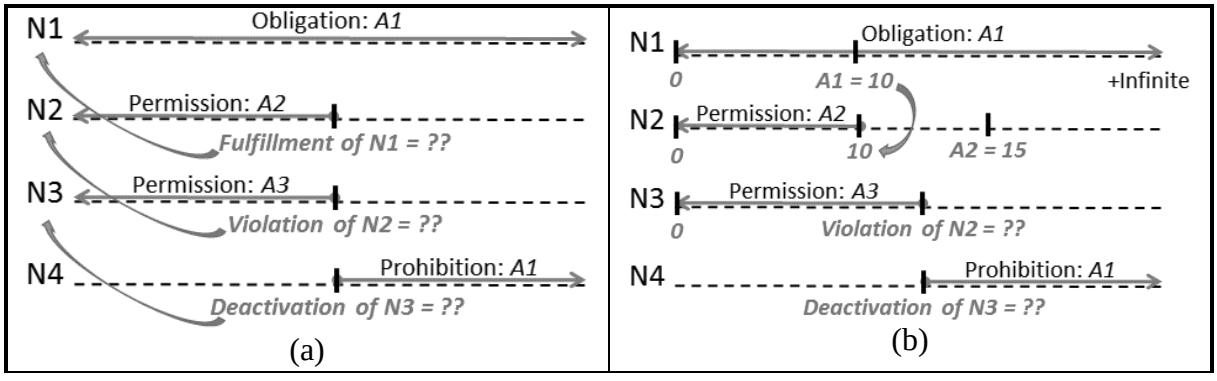
Na especificação das normas, podem haver normas definidas em cascata. As normas são definidas em cascata quando há uma dependência em cascata de parte da definição de uma norma com a ocorrência de algum evento de outra norma. O evento pode ser a ativação ou desativação de norma ou o cumprimento ou violação de uma norma. Por exemplo, sejam *N1*, *N2*, *N3* e *N4* quatro normas definidas em um mesmo contexto *C* e regulando um mesmo agente *AG*. As quatro normas estão definidas, como segue.

- *N1* obriga *AG* a executar ação *A1*;
- *N2* autoriza *AG* a executar ação *A2*, antes do cumprimento de *N1*;
- *N3* autoriza *AG* a executar ação *A3*, antes da violação de *N2*;
- *N4* proíbe *AG* de executar ação *A1*, depois da desativação de *N3*.

Como pode ser observado, existe uma dependência em cascata nas definições destas quatro normas: *N4* depende de *N3*, *N3* depende de *N2* e *N2* depende de *N1* (Figura 6.a). Além das definições das normas, vamos supor que o designer do sistema forneça o seguinte cenário de execução.

- Ação *A1* foi executada no tempo 10 por *AG*;
- Ação *A2* foi executada no tempo 15 por *AG*.

De acordo com a primeira abordagem de detecção de conflitos, o sistema normativo verifica, a partir de um conjunto de normas, se tal cenário causaria algum conflito normativo entre pares de normas caso a execução do sistema acontecesse tal como o cenário de execução passado pelo designer. Em vista disso, o sistema normativo primeiro executa a máquina de inferência OWL e SWRL para que todas as classificações OWL sejam inferidas e todas as regas SWRL sejam executadas. No exemplo, a regra do Quadro 1 é satisfeita e o tempo do cumprimento de *N1* é atualizado para 10. Após isso, o Algoritmo 1 é executado para todas as normas *N1*, *N2*, *N3* e *N4* a fim de atualizar os tempos de início e fim dos intervalos de ativação e parte da justificativa do conflito normativo. Neste momento, as normas *N1* e *N2* tem seus intervalos de ativação atualizados de zero até *limT* e de zero até 10, respectivamente (Figura 6.b). A máquina de inferência é chamada novamente para atualizar o tempo da violação de *N2* (regra do Quadro 5) e depois o Algoritmo 1 é executado para atualizar o intervalo de ativação da norma *N3* (Figura 6.c). A máquina de inferência é chamada mais uma vez para atualizar o tempo da desativação da norma *N3* (regra do Quadro 8) e depois o Algoritmo 1 é executado para atualizar o intervalo de ativação da norma *N4* (Figura 6.d). Por fim, a máquina de inferência é executada para verificar os conflitos normativos. No exemplo, foi detectado que *N1* e *N4* estão em conflito (Figura 6.e).



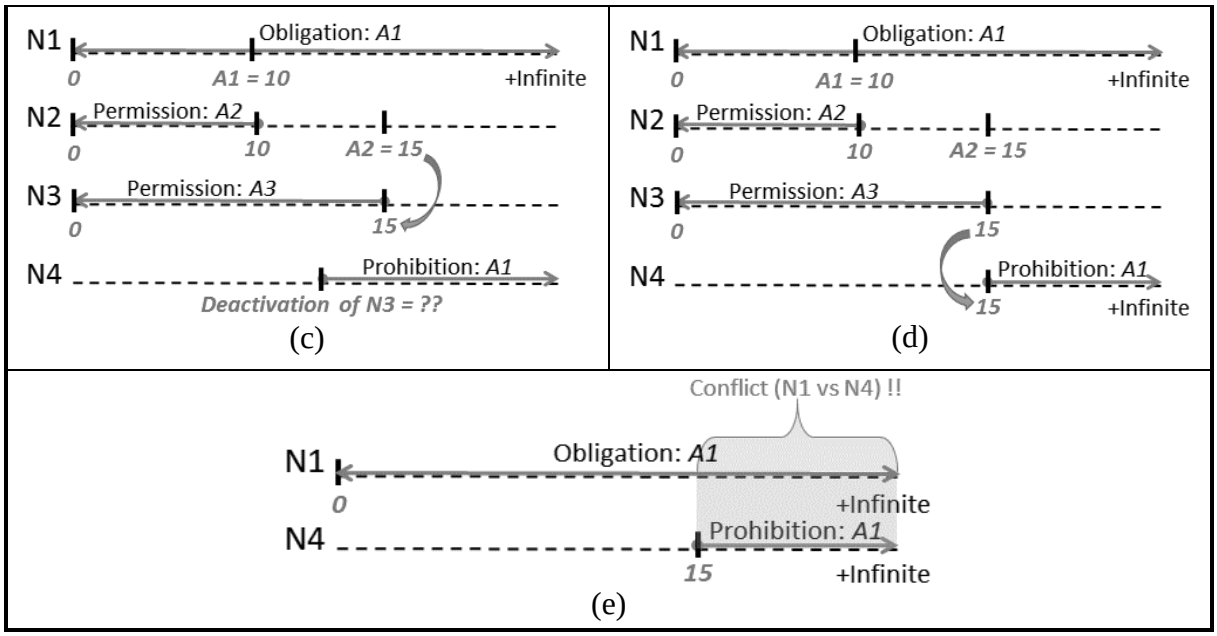


Figura 6: Normas definidas em cascata.

É importante observar que, dependendo do tamanho da dependência em cascata na definição das normas, será necessário chamar várias vezes a máquina de inferência e o algoritmo que atualiza o intervalo de ativação das normas e parte da justificativa do conflito normativo. Outra observação é que se faz necessário atualizar os intervalos de ativação de todas as normas para encontrar os conflitos normativos. No exemplo acima, para identificar o conflito entre *N1* e *N4*, foi preciso atualizar os intervalos das normas *N2* e *N3*.

O Algoritmo 2 detecta os potenciais conflitos normativos entre pares de normas dados os exemplos de cenário de execução do sistema. O algoritmo começa executando a máquina de inferência OWL e SWRL para que todas as classificações OWL sejam inferidas e todas as regas SWRL sejam executadas (linha 3). Neste momento, todas as condições relacionadas com a execução da ação ou participação de uma situação (fornecidos pelo designer) por um agente são inferidas. Após isso, a função *updateAllNormsIntervalAndJustification* é chamada para atualizar os tempos de início e fim dos intervalos de ativação e parte da justificativa do conflito normativo de todas as normas (linha 4). Ela retorna a quantidade de intervalos atualizados, cujo valor é armazenado na variável *numIntervalsBeenUpdated*. Esses passos são repetidos até que não tenham mais intervalos possíveis de serem atualizados (linha 5). É importante perceber que não necessariamente todos os intervalos de ativação de todas as normas do sistema devem ser atualizados, isto é, pode haver alguma norma cujos tempos de início ou fim de seus intervalos de ativação não foram possíveis de serem determinados porque o tempo do evento relacionado a esses tempos não foi fornecido. Por fim, a máquina de inferência OWL e SWRL é executada novamente para que as regras de detecção dos conflitos sejam executadas (linha 6).

```
1. function checkPotentialConflict()
2.     do
3.         runOWLAndSWRLReasoner()
4.         numIntervalsBeenUpdated = updateAllNormsIntervalAndJustification ()
5.     while (numIntervalsBeenUpdated != 0)
6.     runOWLAndSWRLReasoner()
7. end function
```

Algoritmo 2. Algoritmo *checkPotentialConflict* para calcular os potenciais conflitos normativos entre pares de normas dado os cenários de execução passados pelo designer.

É importante perceber também que é possível que haja situações de *deadlock* nas dependências em cascata nas definições das normas. Por exemplo, seja *N1* e *N2* duas normas definidas como segue.

- *N1 obriga* agente *AG* a executar ação *A*, depois do cumprimento de *N2*;
- *N2 proíbe* agente *AG* a executar ação *A*, depois do cumprimento de *N1*.

Como pode ser observado, norma *N1* depende do cumprimento de *N2*, e norma *N2* depende do cumprimento de *N1*. No exemplo, as normas *N1* e *N2* nunca poderão ser ativadas e cumpridas, visto que para que alguma delas seja cumprida, ela precisa estar ativada, e para que esteja ativa, é preciso que a outra seja cumprida. No entanto, situações de *deadlock* na definição das normas não afetam a detecção de conflitos. O algoritmo de detecção de conflito não conseguirá atualizar os tempos iniciais e finais dos intervalos de ativação destas normas, e retornará zero na função *updateAllNormsIntervalAndJustification* (Algoritmo 2 – linha 4) fazendo com que saia do loop e, após isso, não será identificado nenhum conflito normativo ao chamar a função *runOWLAndSWRLReasoner* (Algoritmo 2 – linha 6). O exemplo acima caracteriza normas mal formuladas pelo designer do sistema e não está dentro do escopo deste trabalho identificar normas mal formuladas pelo designer do sistema.

Outra situação que pode acontecer na detecção de conflitos são os conflitos em cascata. Conflitos em cascata ocorrem quando há conflitos encadeados entre as normas. Por exemplo, sejam *N1*, *N2* e *N3* três normas definidas em um mesmo contexto *C* e regulando uma mesma ação *AC* e um mesmo agente *AG*, como segue.

- *N1* obriga *AG* a executar ação *AC*;
- *N2* proíbe *AG* de executar ação *AC*;
- *N3* permite *AG* a executar ação *AC*.

Neste exemplo, pode-se observar que norma *N1* está em conflito com a norma *N2* e esta última está em conflito com norma *N3*, gerando um conflito em cascata entre *N1*, *N2* e *N3*,

como ilustrado na Figura 7. Conflitos em cascata também são detectados usando a abordagem de detecção de conflitos proposta neste trabalho, porém a detecção é feita entre pares de normas.

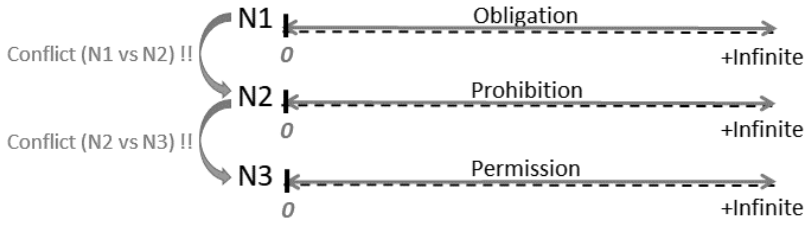


Figura 7: Conflitos em cascata.

3.3.2 DETECÇÃO BASEADA NA GERAÇÃO DOS CENÁRIOS DE EXECUÇÃO

Os sistemas multi-agentes são compostos por múltiplos agentes autônomos e heterogêneos. Por conseguinte, existe uma grande quantidade de possibilidades de cenários de execução que podem acontecer no sistema. Por conta disso, a segunda abordagem de detecção de conflitos gera para o designer do sistema os cenários de execução e faz a detecção de todos os potenciais conflitos normativos entre pares de normas. A segunda abordagem identifica os potenciais conflitos normativos entre pares de normas no sistema ao trocar as posições dos eventos mencionados nas definições destas normas. Lembrando que estes eventos podem ser quaisquer condições apresentadas na Seção 3.1.1.6, ou seja, execução de uma ação por um agente, ativação ou desativação de uma norma, cumprimento ou violação de uma norma, ou um fato no sistema que tenha se tornado verdade para um agente.

A troca das posições dos eventos é dada pela permutação simples entre os eventos mencionados nas definições do par de normas. Uma permutação simples de *n* elementos são sequências que podem se formar com esses *n* elementos, tais que essas sequências diferem uma da outra apenas pela ordem de seus elementos. A formula da permutação simples é dada como segue.

$$P_n = n.(n-1).(n-2)\ldots 1 = n! \qquad (1)$$

onde *n* é o número de eventos mencionados nas definições do par de normas, considerando apenas a quantidade de eventos distintos. Por exemplo, a permutação de três eventos *X*, *Y* e *Z* geraria seis sequências ordenadas: *XYZ*, *XZY*, *YXZ*, *YZX*, *ZXY* e *ZYX*. Porém, se houverem dois eventos repetidos, por exemplo, *X*, *Y* e *Y*, o cálculo da permutação consideraria apenas os eventos distintos, *X* e *Y*, gerando duas sequências ordenadas: *XY* e *YX*. A interpretação de uma sequência de *n* eventos pode ser feita como: <*nomeEvento_1*> "acontece antes de" <*nomeEvento_2*> ... "que acontece antes de" <*nomeEvento_n*>. Por exemplo, as sequências *XY* e *ZYXW* podem ser interpretadas, respectivamente, como: *X* acontece antes de *Y*, e Z acontece antes de *Y* que acontece antes de *X* que acontece antes de *W*.

Cada sequência resultante da permutação é um exemplo de possíveis cenários de execução do sistema. Após gerar todas as sequências da permutação, para cada uma delas, um valor hipotético de tempo é atribuído a cada evento da sequência, levando em consideração a precedência entre eles. Após isso, as regras de detecção de conflitos são executadas a fim de encontrar os potenciais conflitos normativos. A saída da segunda abordagem são todos os potenciais conflitos normativos entre duas normas juntamente com seus respectivos cenários de execução. Os cenários de execução gerados são as sequências ordenadas de eventos mencionados nas normas que causariam o conflito caso tais eventos fossem executados nessa ordem no sistema multi-agentes.

Por exemplo, sejam *N1* e *N2* duas normas com conceitos deônticos contraditórios, definidas em um mesmo contexto *C* e regulando uma mesma ação *AC* e um mesmo agente *AG*, como segue.

- Norma *N1 obriga AG* a executar *AC*, *depois* de um evento *X*, e
- Norma *N2 proíbe AG* de executar *AC*, *antes* de outro evento *Y*.

E deseja-se saber quais são os potenciais conflitos normativos entre *N1* e *N2*. De acordo com a segunda abordagem, primeiro são feitas as permutações das posições dos eventos mencionados nas definições das normas *N1* e *N2*. No caso há dois eventos distintos, *X* e *Y*, referidos por *N1* e *N2*, respectivamente. A permutação entre eles gera dois cenários de execução, como mostrado a seguir.

- Evento *X* acontece antes de evento *Y* (*X* < *Y*), e
- Evento *Y* acontece antes de evento X (*Y* < *X*).

Após gerar as permutações, um valor hipotético de tempo é atribuído a cada evento, levando em consideração a precedência entre eles. Portanto, para *X* < *Y*, pode-se atribuir os valores 1 e 2 para *X* e *Y*, respectivamente. A Figura 8 ilustra as atribuições dos valores de tempo aos eventos mencionados nas normas *N1* e *N2*.

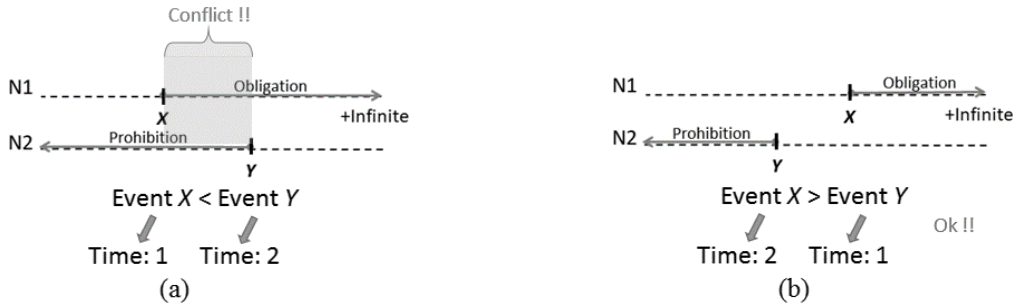


Figura 8: Atribuição de valores de tempo aos eventos mencionados nas normas *N1* e *N2*.

Após isso, as regras de detecção de conflitos são executadas e o seguinte potencial conflito normativo é identificado, extraído do valor da propriedade *conflictJustification*: "*Conflict between N1 and N2 if execution order is: 1º: X, 2º: Y*". A propriedade *conflictJustification* foi apresentada nas seções 3.1.1.1 e 3.3.1. Portanto, caso a execução do sistema multi-agentes aconteça na seguinte ordem: evento *X* da norma *N1* sendo executado antes do evento *Y* da norma *N2*, pode-se afirmar que as normas *N1* e *N2 estarão* em conflito (Figura 8.a). Caso contrário, *N1* e *N2 não* estarão em conflito, como pode ser observado na Figura 8.b.

É importante observar que os eventos *X* e *Y* do exemplo acima podem acontecer ao mesmo tempo durante a execução do sistema multi-agentes. Quando isso acontece, obviamente existirá conflito normativo, pois haverá interseção entre os períodos de ativação das normas *N1* e *N2* exatamente no momento que tais eventos forem executados (Figura 9).

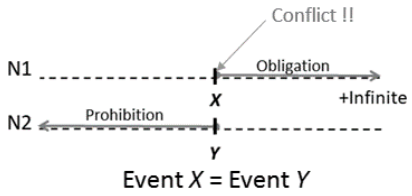


Figura 9: Conflito entre as normas *N1* e *N2* quando os eventos *X* e *Y* acontecem ao mesmo tempo.

Consideramos este caso como trivial, visto que *sempre* haverá conflito entre duas normas com conceitos deônticos contraditórios (do tipo obrigação *versus* proibição ou permissão *versus* proibição) devido a interseção entre seus períodos de ativação no exato momento do acontecimento do evento. Por esse motivo, o resultado da segunda abordagem pode ser interpretado como uma sequência ordenada de eventos do tipo $A \leq B$, quando *A* e *B* sejam eventos pertencentes a normas diferentes, onde lê-se: evento *A* "acontece antes ou ao mesmo tempo que" evento *B*.

Um par de normas pode ter até quatro eventos envolvidos em suas definições. No caso de haver três eventos, uma das normas estará associada a duas condições através dos relacionamentos *hasBefore* e *hasAfter* e a outra norma estará associada a apenas uma condição através de um destes relacionamentos. Se o par de normas se referirem a quatro eventos, as duas normas estarão associadas a duas condições através dos relacionamentos *hasBefore* e *hasAfter*. Os relacionamentos *hasBefore* e *hasAfter* foram apresentados na Seção 3.1.1.1. Por exemplo, digamos que os eventos envolvidos no par de normas sejam *X*, *Y*, *W* e *Z*, todos distintos. A permutação destes quatro eventos gera vinte e quatro sequências ordenadas representando os cenários de execução. De acordo com a segunda abordagem, após geradas as permutações, para

cada cenário gerado, um valor hipotético de tempo é atribuído a cada evento do cenário, levando em consideração a precedência entre eles e depois as regras de detecção de conflitos são executadas a fim de encontrar os potenciais conflitos normativos.

Está fora do escopo deste trabalho a representação de normas com mais de um relacionamento *hasBefore* ou mais de um *hasAfter*. Portanto, a quantidade máxima de eventos mencionados no par de normas é quatro. A especificação de normas com repetições de relacionamento *hasBefore* ou *hasAfter* serão tratadas em trabalhos futuros. O Algoritmo 3 calcula todos os potenciais conflitos normativos entre duas normas (*n1* e *n2*). O algoritmo recebe as duas normas como entrada (linha 1) e começa calculando a permutação dos eventos mencionados em *n1* e *n2* (linha 2). Para cada sequência da permutação, é feita a atribuição de um valor hipotético de tempo a cada evento da sequência, levando em consideração a precedência entre eles (linha 4). Após isso, a função *updateNormIntervalsAndJustification* (Algoritmo 1) é chamada para atualizar na ontologia os valores dos tempos de início e fim dos intervalos de ativação de *n1* e *n2*, e parte da justificativa que contém os tempos dos eventos que causariam o conflito (linha 5 e 6). Por fim, as regras responsáveis pela verificação de conflitos são executadas a fim de encontrar os potenciais conflitos normativos (linha 7).

```
1. function checkAllPotentialConflict(Norm n1, Norm n2)
2.     perm ← getAllPermutation(n1.conditions, n2.conditions)
3.     foreach p ∈ perm do
4.         attributeTimeValue(p)
5.         updateNormIntervalsAndJustification(n1)
6.         updateNormIntervalsAndJustification(n2)
7.         runOWLAndSWRLReasoner()
8.     end for
9. end function
```

Algoritmo 3. Algoritmo *checkAllPotentialConflict* para calcular todos os potenciais conflitos normativos entre duas normas (BELCHIOR e DA SILVA, 2017c).

A detecção de todos os potenciais conflitos normativos entre pares de normas (Algoritmo 3) não gera dependências em cascata de parte da definição de uma norma com a ocorrência de algum evento de outra norma, como visto na Seção 3.3.1.2, pois todos os tempos dos eventos mencionados no par de normas são gerados previamente durante a atribuição de valores de tempo após as permutações. Portanto, diferente do Algoritmo 2, as funções *updateNormIntervalsAndJustification* e *runOWLAndSWRLReasoner* só precisam ser chamadas uma única vez para se identificar os potencias conflitos para cada sequência da permutação.

3.4 – POTENTIAL CONFLICT CHECKER

A fim de dar suporte à verificação dos potenciais conflitos normativos em um sistema multi-agente, foi desenvolvida a ferramenta *Potential Conflict Checker* (PCC). Ela fornece ao usuário um ambiente de criação das normas e de todos os elementos que compõem a norma, tais como contexto, entidades e ações. Além disso, é possível criar os cenários de execução do sistema referentes à execução de uma ação e participação de uma situação por um agente. As duas abordagens de detecção de potencias conflitos normativos descritas neste trabalho foram implementadas e os resultados são mostrados através da ferramenta.

A ferramenta PCC foi desenvolvida usando a linguagem Java e toda a manipulação das ontologias em OWL DL foi feita através da OWL API (HORRIDGE e BECHHOFER, 2011). OWL API é uma API que dá suporte à criação e manipulação de ontologias em OWL. Foi escolhida a OWL API porque ela é implementada em Java e está disponível como *open source* sob a licença LGPL. Além disso, OWL API tem sido usada em grandes projetos como, por exemplo, a ferramenta de edição de ontologias OWL chamada Protégé (KNUBLAUCH *et al.*, 2004). A máquina de inferência OWL DL usada foi o Pellet (SIRIN *et al.*, 2007). Pellet é uma máquina de inferência para ontologias escritas em OWL DL, como também para regras em SWRL. Para fazer as consultas nas ontologias OWL através da linguagem de consulta SPARQL, foi usada SPARQL-DL API (SIRIN e PARSIA, 2007).

Nas próximas subseções são apresentadas as tarefas que podem ser executas na ferramenta PCC para se criar uma norma, os componentes das normas, os cenários de execução, e para executar e visualizar os potenciais conflitos normativos usando as duas abordagens descritas neste trabalho.

3.4.1 MANIPULAÇÃO DAS NORMAS

A Figura 10 ilustra a tela de manipulação das normas do sistema, quando selecionada a aba "*Norms Detail*". Nesta tela é possível criar, editar, remover e visualizar as normas. Para criar uma norma, deve-se clicar no botão "*New Norm*" (Figura 10.a) e preencher/selecionar as informações sobre a nova norma através dos campos *Norm Name*, *Context*, *Deontic Concept*, *Action*, *Entity* e *Conditon*. Estas informações devem ser criadas anteriormente pelo usuário através da tela de criação e listagem dos elementos que compõem as normas, apresentada na Seção 3.4.2 O usuário pode adicionar até duas condições de ativação, responsáveis pela definição do período de ativação da norma, ao clicar no botão "*Add Condition*" (Figura 10.c). Após isso, o usuário deve clicar no botão "*Save Norm*" (Figura 10.e). Todas as normas criadas

são listadas no campo "*Norm*" (Figura 10.g), podendo o usuário selecionar uma norma para visualizar e/ou editar. Após concluída a criação de todas as normas, deve-se clicar no botão "*Save Ontology*" (Figura 10.f) para que todas mudanças sejam persistidas no arquivo OWL contendo as ontologias de Norma e de Cenário de Execução.

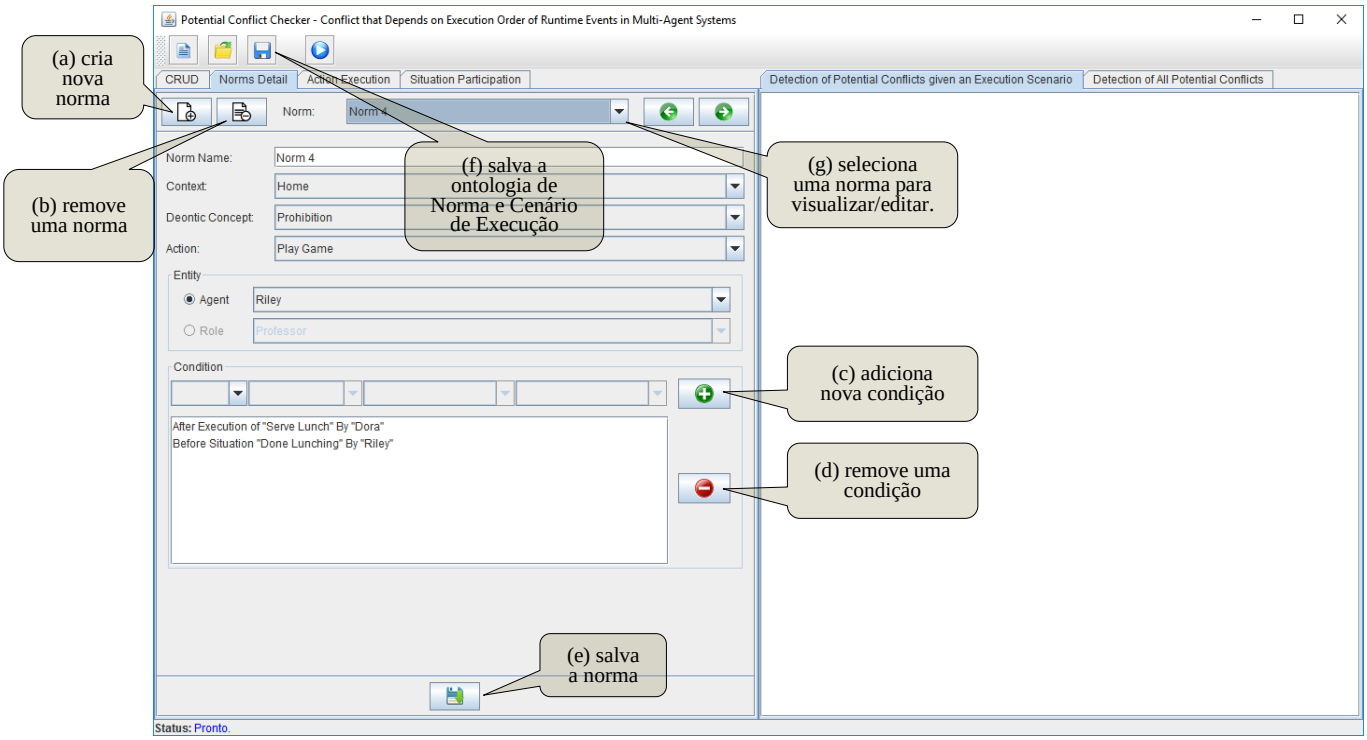


Figura 10: Tela de manipulação das Normas.

3.4.2 MANIPULAÇÃO DOS ELEMENTOS DA NORMA

A Figura 11 mostra a tela de criação e listagem dos elementos que compõem as normas, quando selecionada a aba "*CRUD*". Nesta tela é possível criar, remover e visualizar os elementos das normas, tais como contexto, entidade e ação. As situações nas quais os agentes podem participar também são criadas nesta tela. Para adicionar um elemento, deve-se digitar o nome do elemento que se deseja adicionar no campo correspondente e depois pressionar o botão "*Add*" ao lado do campo de edição correspondente como, por exemplo, o elemento contexto (Figura 11.a). O botão "*Delete*" remove um elemento selecionado na lista correspondente como, por exemplo, o elemento entidade, mostrado na Figura 11.d. Após todos os elementos das normas e todas as situações terem sido criados, deve-se clicar no botão "*Save Ontology*" (Figura 10.f) para que todas mudanças sejam persistidas no arquivo OWL. Desta forma, uma instância da classe correspondente ao elemento inserido (ou removido) é adicionada (ou removida) na ontologia.

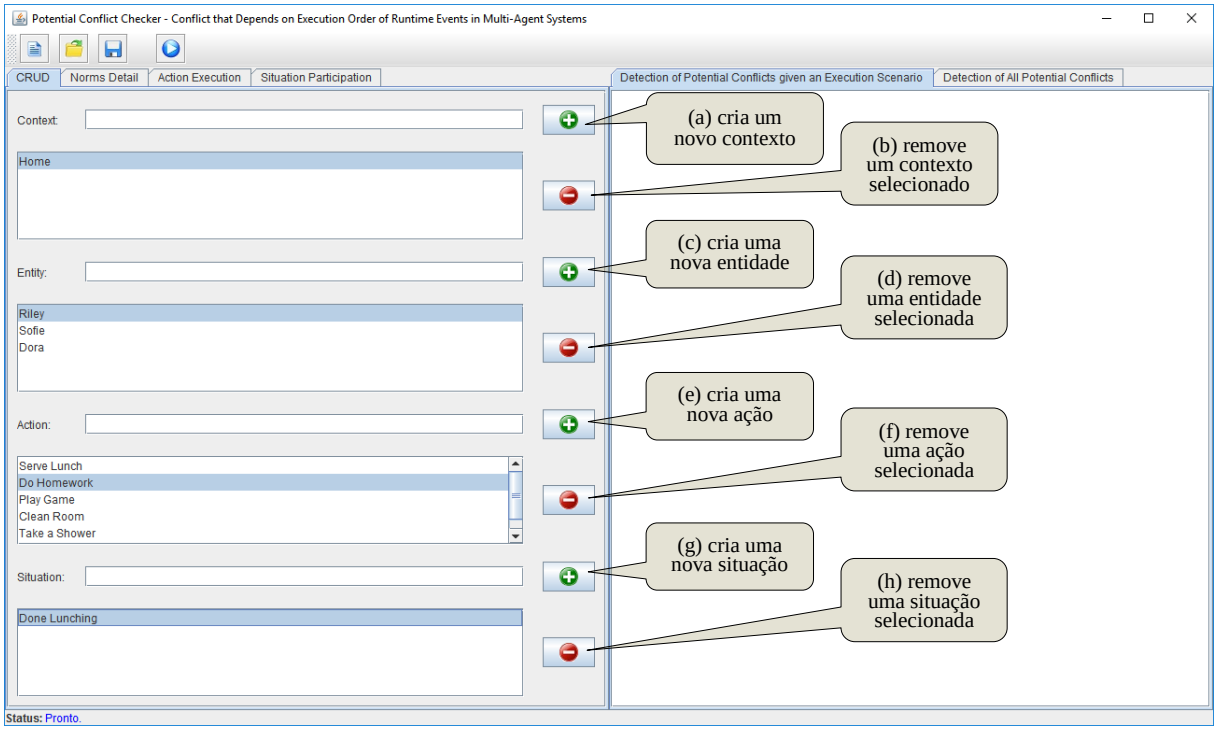


Figura 11: Tela de criação e listagem dos elementos que compõem as normas.

3.4.3 MANIPULAÇÃO DO CENÁRIO DE EXECUÇÃO

As figuras Figura 12 e Figura 13 ilustram as telas de manipulação dos cenários de execução, quando selecionadas as abas "*Action Execution*" e "*Situation Participation*", respectivamente. A manipulação dos cenários de execução compreende a criação, remoção e visualização das ações executadas por um agente (Figura 12) e criação, remoção e visualização das situações participadas por um agente (Figura 13). Para adicionar um cenário referente a execução de uma ação, o usuário deve selecionar uma ação, um agente, informar o tempo que tal ação foi executada por tal agente, e depois clicar no botão "*Add Action Execution*" (Figura 12.a). Para adicionar um cenário referente a participação de uma situação, o usuário deve selecionar uma situação, um agente, informar o tempo que tal situação aconteceu para tal agente, e depois clicar no botão "*Add Situation Participation*" (Figura 13.a). Para remover um cenário de execução, o usuário deve selecionar o cenário desejado na lista e depois clicar no botão "*Delete Action Execution*" (Figura 12.b) ou "*Delete Situation Participation*" (Figura 13.b), para remover a execução de uma ação ou participação de uma situação por um agente, respectivamente. Após todos os cenários de execução terem sido criados, deve-se clicar no botão "*Save Ontology*" (Figura 10.f) para que todas mudanças sejam persistidas no arquivo OWL contendo as ontologias de Norma e de Cenário de Execução.

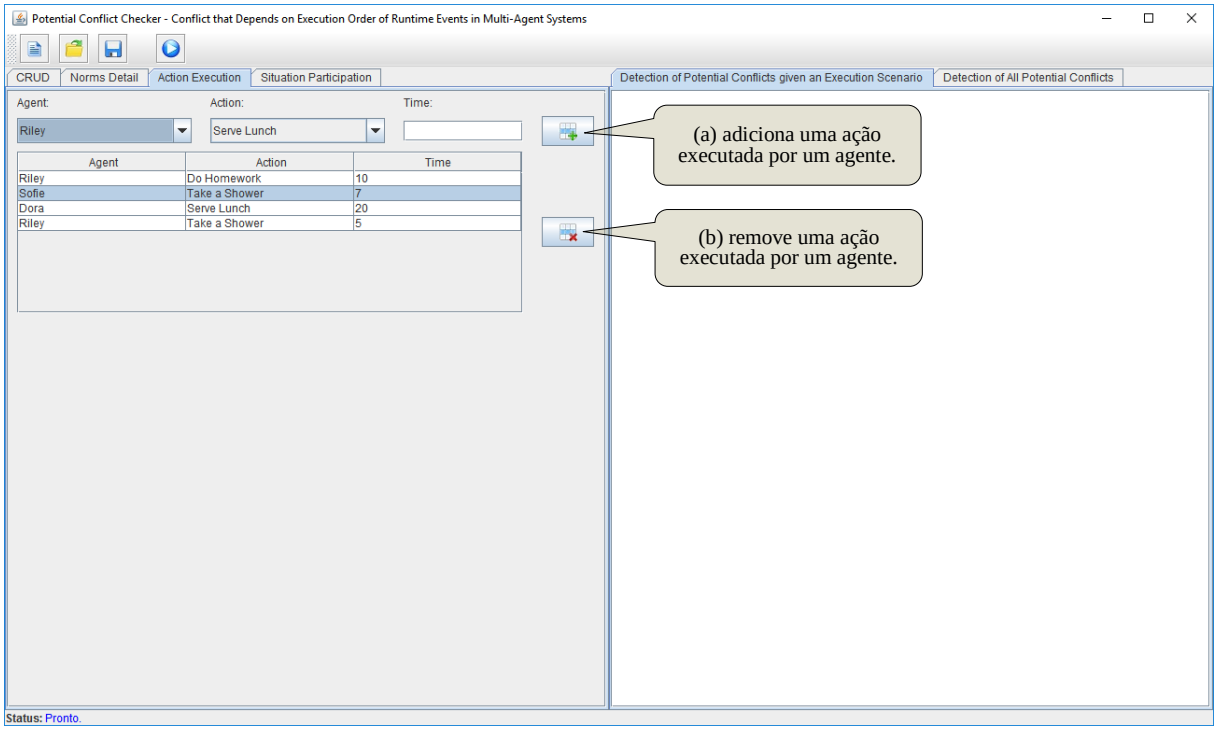


Figura 12: Tela de manipulação do cenário de execução referente às ações executadas por um agente.

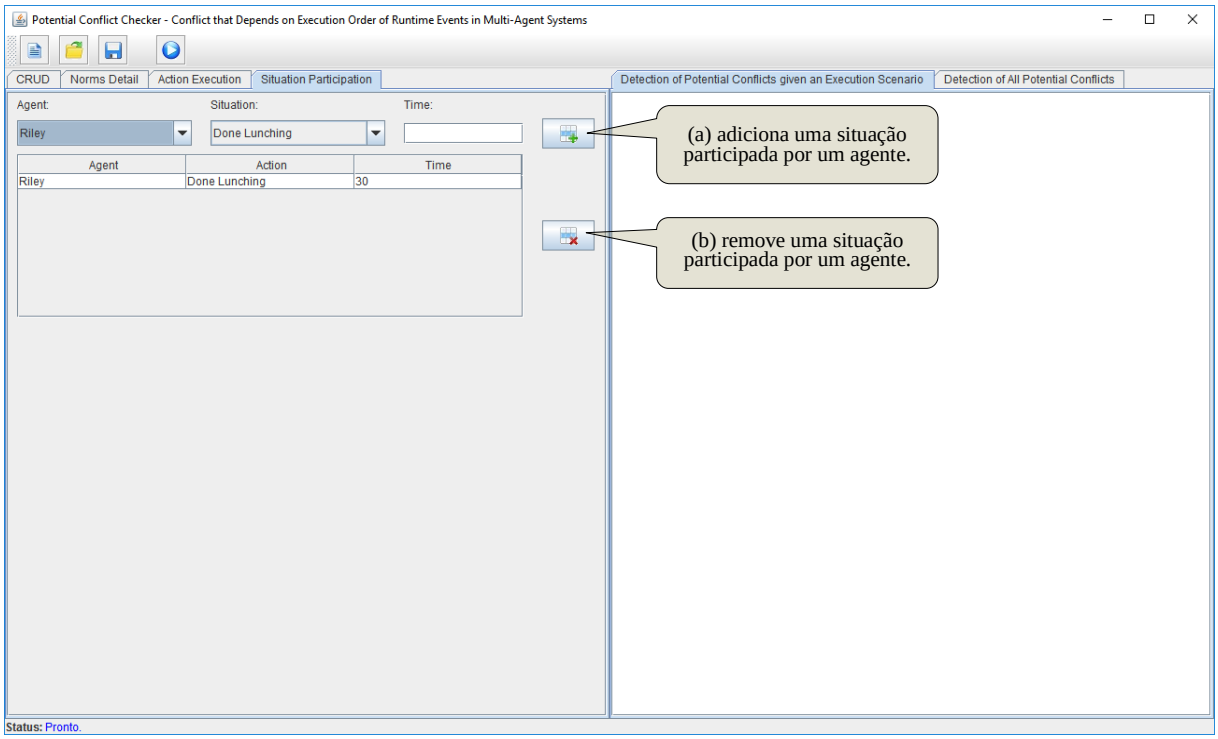


Figura 13: Tela de manipulação do cenário de execução referente às situações participadas por um agente.

3.4.4 EXECUÇÃO DO POTENTIAL CONFLICT CHECKER

A Figura 14 mostra o procedimento para executar a detecção dos potenciais conflitos normativos. O usuário deve clicar no botão "*Run*", como mostrado na Figura 14.a, e depois escolher uma das opções abaixo:

- (1) *Detect Potential Conflicts given the Execution Scenarios Provided by the Designer* (Figura 14.b)
- (2) *Detect All Potential Conflicts* (Figura 14.c)

A primeira opção executa a detecção dos potenciais conflitos normativos de acordo com a primeira abordagem, apresentada na Seção 3.3.1. O resultado da detecção é mostrado na Figura 15. No resultado são informadas as normas em potencial conflito (Figura 15.a), a justificativa do conflito mostrando qual foi a ordem dos eventos que resultou no conflito (Figura 15.b) e os tempo inferidos de cada condição de ativação das normas (Figura 15.c).

A segunda opção executa a detecção de todos os potenciais conflitos normativos de acordo com a segunda abordagem, apresentada na Seção 3.3.2. O resultado da detecção é ilustrado na Figura 16. No resultado desta detecção são informadas as normas em potencial conflito (Figura 16.a) e a justificativa do conflito mostrando qual foi a ordem da execução dos eventos que resultou no conflito (Figura 16.b).

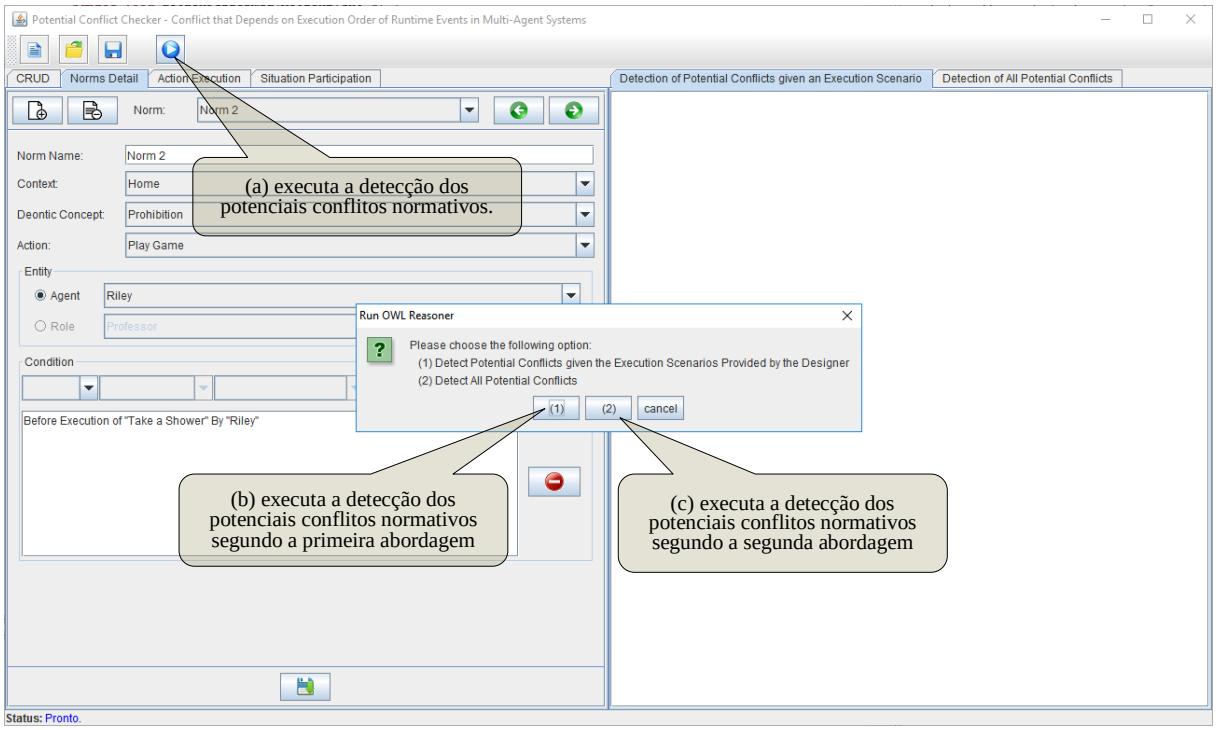


Figura 14: Execução da detecção dos potenciais conflitos normativos.

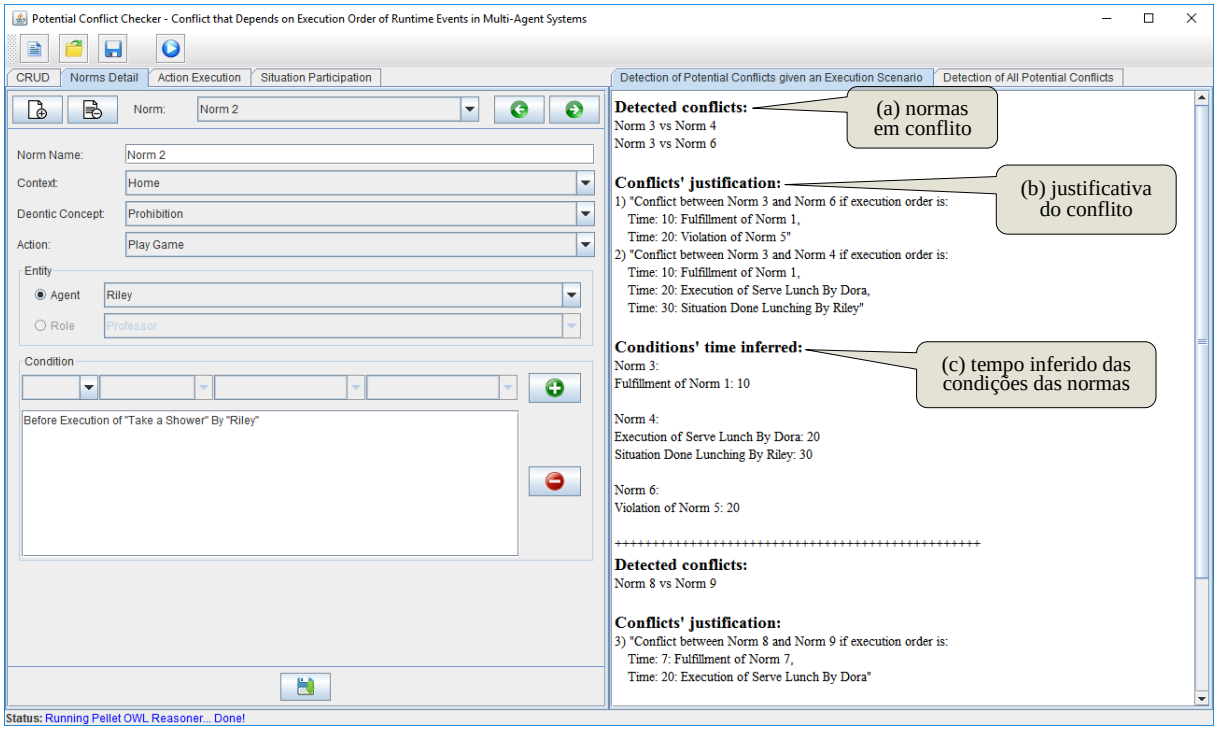


Figura 15: Resultado da detecção dos potenciais conflitos normativos de acordo com a primeira abordagem.

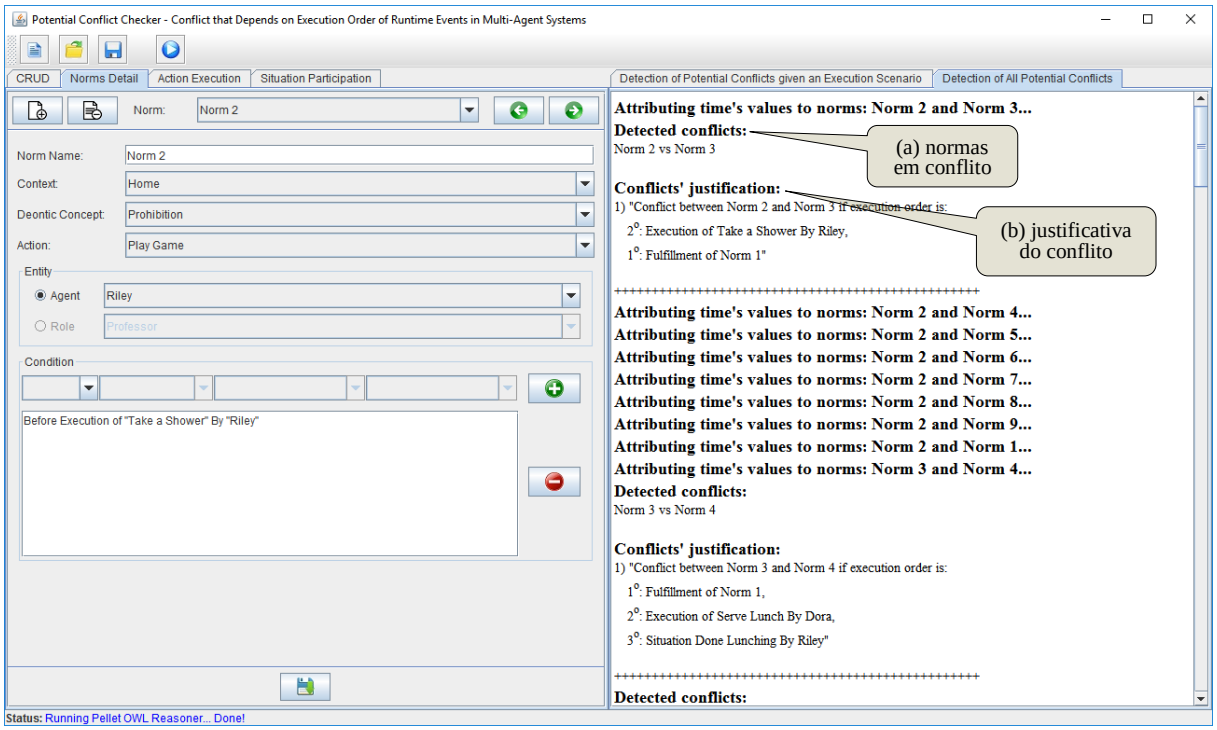


Figura 16: Resultado da detecção de todos os potenciais conflitos normativos de acordo com a segunda abordagem.

CAPÍTULO 4 – ESTUDO DE CASO E ANÁLISE DOS RESULTADOS

Este capítulo apresenta um estudo de caso para demonstrar as duas abordagens de detecção dos potenciais conflitos normativos. Além disso, são apresentadas as avaliações dos especialistas na área de normas para validar os resultados da detecção dos potenciais conflitos normativos entre pares de normas.

4.1 DESCRIÇÃO DO ESTUDO DE CASO

A fim de ilustrar os conceitos discutidos no Capítulo 3, e as abordagens de detecção de potenciais conflitos normativos, foi executado um estudo de caso no domínio de um ambiente doméstico envolvendo três agentes: *Riley, Sofie* e *Dora*, baseado no exemplo apresentado em BELCHIOR e DA SILVA, (2017c). Os agentes *Riley* e *Sofie* representam duas crianças irmãs que moram em uma mesma casa. O terceiro agente, *Dora*, desempenha o papel de uma baba. Um conjunto de nove regras diárias (*N1*, *N2*, *N3*, *N4*, *N5*, *N6*, *N7*, *N8* e *N9*) foram definidas para os três agentes, como segue.

- *N1 obriga Riley* a executar ação *doHomework*;
- *N2 proíbe Riley* de executar ação *playGame antes* dele executar *takeAShower*;
- *N3 autoriza Riley* a executar *playGame depois* do cumprimento de *N1*;
- *N4 proíbe Riley* de executar *playGame depois* que *Dora* executa *serveLunch* e *antes* da situação *doneLunching* torna-se verdade para *Riley*;
- *N5 obriga Riley* a executar ação *cleanRoom antes* de *Dora* executar *serveLunch*;
- *N6 proíbe Riley* de executar ação *playGame se* ele violar norma *N5*;
- *N7 obriga Sofie* a executar *takeAShower depois* que *Riley* executar *takeAShower*;
- *N8 autoriza Sofie* a executar *playGame depois* dela cumprir com a norma *N7*;
- *N9 proibe Sofie* de executar *playGame depois* que Dora executar *serveLunch*.

Na ontologia de Norma, as seguintes instâncias são criadas. Os agentes *Riley, Sofie* e *Dora* são definidos como instâncias da classe *Entity*. As ações *doHomework, playGame, takeAShower, serveLunch* e *cleanRoom* são definidas como instâncias da classe *Action*. A situação *doneLunching* é definida como uma instância da classe *Situation*. O contexto criado para essas normas chama-se *Home*, e é definido como uma instância da classe *Environment*.

Além disso, instâncias da classe *Condition* do tipo *ExecutionOfAction*, *FulfillmentOfNorm*, *ViolationOfNorm* e *ParticipationInSituation* são criadas, como segue. Três

instâncias da classe *ExecutionOfAction* são criadas. A primeira instância refere-se à execução da ação *takeAShower* pelo agente *Riley*. A segunda pertence à execução da ação *takeAShower* pela agente *Sofie* e a terceira instância trata da execução da ação *serveLunch* pela agente *Dora*. Uma instância da classe *FulfillmentOfNorm* é criada na ontologia de norma referindo-se ao cumprimento da norma *N1*. Uma instância da classe *ParticipationInSituation* é criada para indicar a participação do agente *Riley* na situação *doneLunching*. Uma instância da classe *ViolationOfNorm* é criada para representar a violação da norma *N5*. Por fim, nove instâncias da classe *Norm* são criadas relacionadas com suas respectivas entidades, ações, contexto, condições e conceitos deônticos, de acordo com as definições das normas apresentadas acima.

4.2 DETEÇÃO DE CONFLITOS USANDO A PRIMEIRA ABORDAGEM

A primeira abordagem de detecção de conflitos recebe como entrada o conjunto de normas e possíveis exemplos de cenários de execução fornecidos pelo designer do sistema. Em seguida, o sistema normativo verifica, a partir de um conjunto de normas, se tal cenário causaria algum potencial conflito normativo caso a execução do sistema acontecesse tal como o cenário de execução passado pelo designer. Como saída, o sistema normativo retorna os potenciais conflitos normativos, resultado da aplicação do cenário de execução fornecido pelo designer, e uma justificativa explicando a causa do conflito. Como descrito na Seção 3.3.1, no cenário de execução, o designer do sistema somente precisa fornecer informações sobre os tempos de quando uma ação é executada e/ou quando uma situação se torna verdade para um agente. Os demais tempos, tempo da ativação e desativação, cumprimento e violação de uma norma são automaticamente inferidos pelas regras SWRL, descritas na Seção 3.2. Uma vez que todos os tempos das condições das normas forem inferidos, as regras responsáveis por fazer as detecções dos conflitos são executadas.

Vamos supor que o designer do sistema forneça o seguinte cenário de execução.

- *Riley* executa a ação *takeAShower* no tempo 5;
- *Sofie* executa ação *takeAShower* no tempo 7;
- *Riley* executa a ação *doHomework* no tempo 10;
- *Dora* executa a ação *serveLunch* no tempo 20;
- Situação *doneLunching* torna-se verdade para *Riley* no tempo 30.

Ao executar o Algoritmo 2 para verificar os potencias conflitos normativos, dados o conjunto de normas e o cenário de execução fornecidos pelo designer do sistema, pode-se concluir as seguintes informações sobre os intervalos de ativação das normas. Pode-se observar

que na definição da norma *N1* não tem especificada uma condição que defina seu período de ativação. Portanto, a norma *N1* estará sempre ativa desde o começo da execução do sistema (tempo zero) até +infinito, ou seja, até o termino da execução do sistema (Figura 17.a). A Seção 3.1.1.6 apresenta como os intervalos de ativação de uma norma podem ser configurados, de acordo com seu tipo. Em relação a norma *N2*, pode-se observar que ela proíbe o agente *Riley* de executar a ação *playGame* até o tempo 5, quando ele executa a ação *takeAShower* de acordo com o cenário de execução (Figura 17.b). O tempo da condição *execução de uma ação* é inferido pela regra do Quadro 9. A norma *N1* é cumprida no tempo 10 quando a ação *doHomework* é executada por *Riley*. Então, o período de ativação da norma *N3* se inicia no tempo 10 (Figura 17.c). O tempo da condição *cumprimento de uma norma* é inferido pela regra do Quadro 1. De acordo com cenário de execução, a norma *N4* proíbe *Riley* de executar a ação *playGame* do tempo 20 ao tempo 30 (Figura 17.d), e norma *N5* obriga *Riley* a executar a ação *cleanRoom* até o tempo 20 (Figura 17.e). O tempo da condição *participação de uma situação* por um agente é inferido pela regra do Quadro 10.

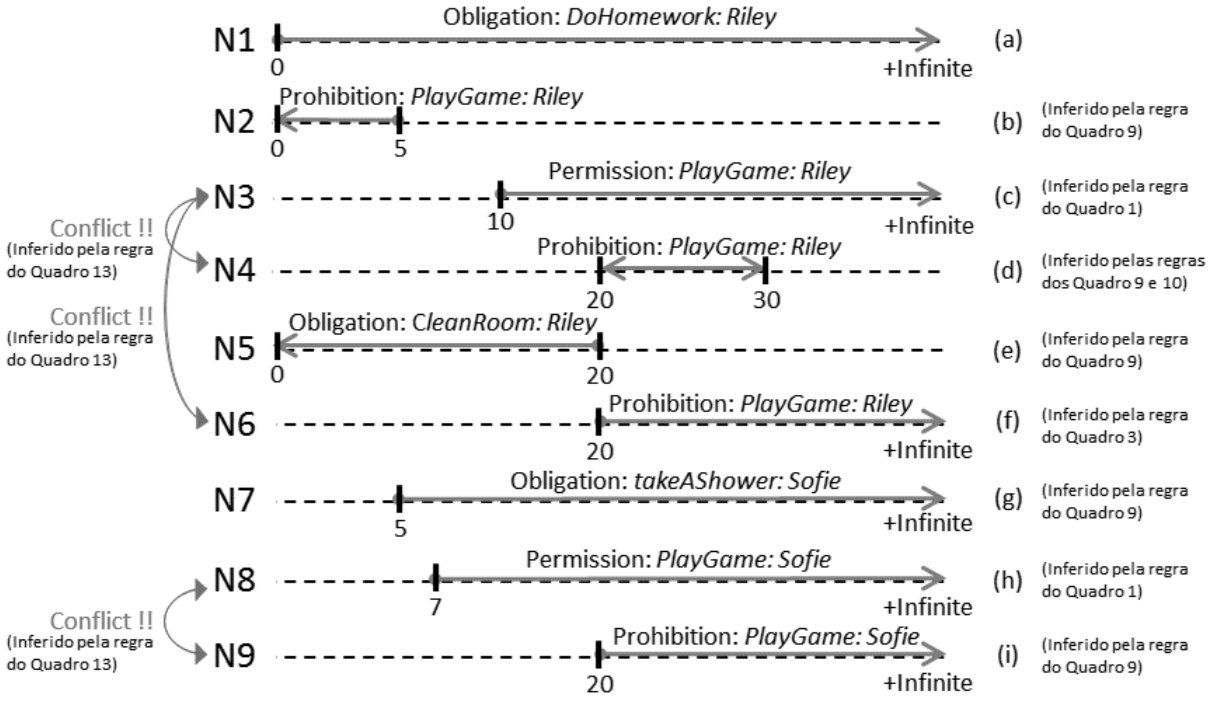


Figura 17: Intervalos de ativação das normas.

Devido a nenhuma informação sobre a execução da ação *cleanRoom* ter sido mencionada no cenário de execução durante o período em que a norma *N5* estava ativa, o estado de obediência de *N5* tornou-se violado no tempo 20 (quando a norma *N5* foi desativada) e, portanto, a ativação da norma *N6* inicia neste tempo (Figura 17.f). O tempo da condição *violação de uma norma* é inferida pela regra do Quadro 3. A norma *N7* inicia sua ativação no

tempo 5, depois do agente *Riley* ter executado a ação *takeAShower* (Figura 17.g). A norma *N7* é cumprida por *Sofie* quando ela executa a ação *takeAShower* no tempo 7, de acordo com o cenário de execução. Portanto, norma *N8* inicia sua ativação no tempo 7 (Figura 17.h). For fim, a norma *N9* é ativada no tempo 20, quando *Dora* executa a ação *serveLunch* (Figura 17.i). Figura 17 ilustra como ficaram configurados todos os períodos de ativação de cada norma.

Uma vez que todos os tempos das condições das normas foram inferidos, as regras responsáveis por fazer as detecções dos conflitos são executadas. Como descrito na Seção 3.3.1, duas normas estão em conflito quando (i) há alguma interseção entre seus períodos de ativação, (ii) os conceitos deônticos das normas são contraditórios (isto é, obrigação *versus* proibição ou permissão *versus* proibição), (iii) elas estão regulando uma mesma entidade e (iv) um mesmo comportamento, e (v) estão definidas em um mesmo contexto. Pode-se perceber na Figura 17 que, após aplicar todos os cenários de execução fornecidos pelo designer, três potenciais conflitos normativos são detectados: conflito entre (i) normas *N3* e *N4*, (ii) normas *N3* e *N6* e (iii) normas *N8* e *N9*. Todas elas têm conceitos deônticos contraditórios (no caso, entre uma permissão e uma proibição), regulam a mesma ação (*playGame*), estão relacionadas ao mesmo agente (*Riley* nas normas *N3*, *N4* e *N6*, e *Sofie* nas normas *N8* e *N9*), estão definidas em um mesmo contexto (contexto *home*), e seus períodos de ativação se interceptam em algum momento na linha do tempo. Pode-se notar que as normas *N3* e *N9* têm (quase) todos os requisitos para estarem em conflito, exceto pelo fato de não estarem regulando uma mesma entidade. Portanto, não há conflito normativo entre as normas *N3* e *N9*.

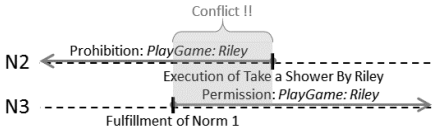
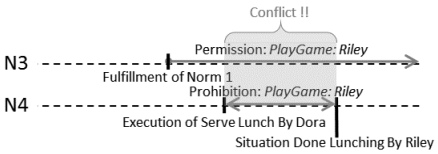
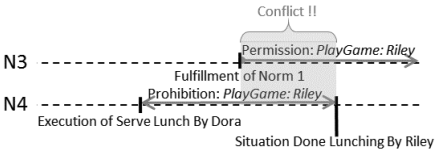
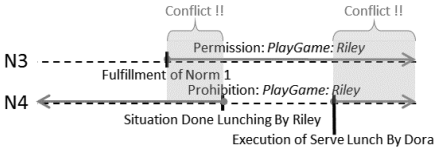
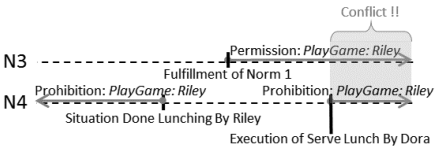
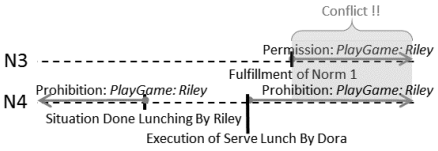
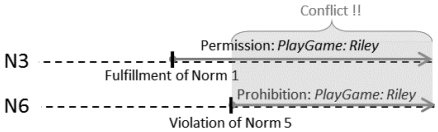
4.3 DETEÇÃO DE CONFLITOS USANDO A SEGUNDA ABORDAGEM

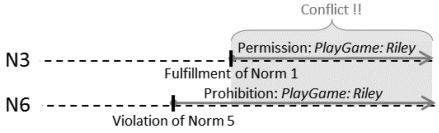
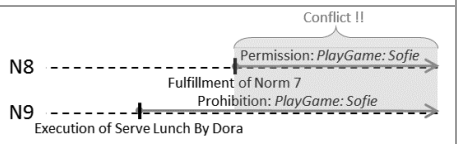
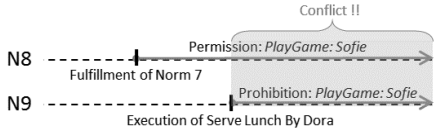
A segunda abordagem de detecção de conflitos gera para o designer do sistema os cenários de execução e faz a detecção de todos os potenciais conflitos normativos entre pares de normas no sistema. Como descrito na Seção 3.3.2, a abordagem identifica os potenciais conflitos normativos entre pares de normas pela permutação das posições dos eventos mencionados nas definições do par de normas. A saída da segunda abordagem são todos os potenciais conflitos normativos entre duas normas juntamente com seus respectivos cenários de execução. Os cenários de execução gerados são uma sequência ordenada dos eventos mencionados nas normas que causariam o conflito caso tais eventos fossem executados nessa ordem no sistema multi-agentes.

Para identificar todos os potencias conflitos normativos entre as normas do estudo de caso, o Algoritmo 3 foi chamado para cada par de normas. O resultado da segunda abordagem

é mostrado na Tabela 2, que apresenta os potenciais conflitos normativos junto com seus respectivos cenários de execução gerados pela segunda abordagem de detecção de conflitos. Ao lado de cada cenário de execução gerado há uma representação gráfica do conflito.

Tabela 2. Potenciais Conflitos Normativos e seus Respectivos Cenários de Execução Gerados pela Segunda Abordagem de Detecção de Conflitos.

Potenciais Conflitos Normativos	**Cenário de execução gerado**	**Ilustração Gráfica do Conflito**
N2* vs. *N3	1º: *Fulfillment of Norm 1* 2º: *Execution of Take a Shower By Riley*	
N3* vs. *N4	1º: *Fulfillment of Norm 1* 2º: *Execution of Serve Lunch By Dora* 3º: *Situation Done Lunching By Riley*	
	1º: *Execution of Serve Lunch By Dora* 2º: *Fulfillment of Norm 1* 3º: *Situation Done Lunching By Riley*	
	1º: *Fulfillment of Norm 1* 2º: *Situation Done Lunching By Riley* 3º: *Execution of Serve Lunch By Dora*	
	1º: *Situation Done Lunching By Riley* 2º: *Fulfillment of Norm 1* 3º: *Execution of Serve Lunch By Dora*	
	1º: *Situation Done Lunching By Riley* 2º: *Execution of Serve Lunch By Dora* 3º: *Fulfillment of Norm 1*	
N3* vs. *N6	1º: *Fulfillment of Norm 1* 2º: *Violation of Norm 5*	

	1º: *Violation of Norm 5* 2º: *Fulfillment of Norm 1*	
N8* vs. *N9	1º: *Execution of Serve Lunch By Dora* 2º: *Fulfillment of Norm 7*	
	1º: *Fulfillment of Norm 7* 2º: *Execution of Serve Lunch By Dora*	

Analisando a saída da segunda abordagem, foram identificados quatro potenciais conflitos normativos, entre as (i) normas *N2* e *N3*, (ii) normas *N3* e *N4*, (iii) normas *N3* e *N6* e (iv) entre as normas *N8* e *N9*. Pode-se observar que há um potencial conflito normativo entre as normas *N2* e *N3* caso a ordem de execução dos eventos mencionados em tais normas no sistema multi-agentes for como segue: "*Fulfillment of Norm* 1" ≤ "*Execution of Take a Shower By Riley*", ou seja, "*Fulfillment of Norm* 1" acontece antes ou ao mesmo tempo que "*Execution of Take a Shower By Riley*". De outra forma, as normas *N2* e *N3 não* estarão em conflito. Em relação as normas *N3* e *N4*, pode-se verificar que há um potencial conflito normativo em quaisquer combinações dos eventos mencionados em tais normas ("*Fulfillment of Norm 1*", "*Execution of Serve Lunch By Dora*" e "*Situation Done Lunching By Riley*"), exceto se a ordem de execução destes eventos for como segue: "*Execution of Serve Lunch By Dora*" < "*Situation Done Lunching By Riley*" < "*Fulfillment of Norm 1*", onde as normas *N3* e *N4 não* ficam em situação de interseção entre seus períodos de ativação e, consequentemente, não existiria conflito normativo entre elas. No caso das normas *N3* e *N6*, pode-se notar que tais normas sempre estarão em conflito, independente da ordem de execução dos eventos mencionados nestas normas ("*Fulfillment of Norm 1*" e "*Violation of Norm 5*"). Um comportamento similar pode ser observado entre as normas *N8* e *N9*, ou seja, não importa quando os eventos "*Execution of Serve Lunch By Dora*" e "*Fulfillment of Norm 7*" aconteçam no sistema, as normas *N8* e *N9* estarão em conflito normativo. Todas as demais normas não estão em situação de conflito.

4.4 VALIDAÇÃO

As consistências das ontologias de Norma e Cenário de Execução e das regras SWRL foram checadas pelo uso de ferramentas de inferência para OWL, como o Pellet. O Pellet faz a checagem de consistência das ontologias OWL DL e garante que elas não contenham fatos

contraditórios (SIRIN *et al.*, 2007). Quanto a validação do método de detecção dos potenciais conflitos normativos, a mesma foi realizada através da avaliação de especialistas na área de conflitos normativos em sistemas multi-agentes. A validação por especialista tem o intuito de verificar a qualidade e a acurácia do método de detecção de conflitos, segundo a opinião de especialistas. O julgamento feito por estudiosos na área torna a abordagem de detecção de potenciais conflitos normativos mais confiável, válida e decisiva para uma tomada de decisão.

Foram considerados os seguintes requisitos para a seleção dos especialistas.

- Ter experiência de pelo menos um ano na área de normas em SMA;
- Ter experiência com conflitos normativos;
- Ter formação em Ciência da Computação ou área correspondente.

Um total de quatro especialistas na área de normas foram selecionados para a realização da avaliação dos conflitos. O perfil dos quatro especialistas é mostrado na Tabela 3.

Tabela 3. Perfil dos Especialistas.

Especialista	Formação	Anos de experiência na área de normas em SMA	Experiência com conflitos normativos
Espec. 1	Formado em Ciência da Computação, com mestrado em Computação.	3 anos	Sim
Espec. 2	Formado em Ciência da Computação, com mestrado em Computação.	4 anos	Sim
Espec. 3	Formado em Ciência da Computação, com mestrado e doutorado em computação.	4 anos	Sim
Espec. 4	Formado em Matemática, com mestrado e doutorado em computação.	2 anos	Sim

Durante a etapa de avaliação dos especialistas, foi explicado o que era um potencial conflito normativo e apresentado um exemplo de detecção. Após isso, uma atividade contendo o mesmo conjunto de normas do estudo de caso apresentado na Seção 4.1 foi passada para os especialistas identificarem todos os potenciais conflitos normativos entre pares de normas. O

resultado da avaliação dos conflitos e a análise dos resultados são apresentados na Seção 4.4.1. O formulário completo contendo as instruções e a atividade de detecção de conflitos passado aos especialistas da área de normas é mostrado no Apêndice A.

4.4.1 AVALIAÇÃO DOS ESPECIALISTAS E ANÁLISE DOS RESULTADOS

As tabelas 4, 5, 6 e 7 mostram os resultados da identificação das normas em conflito junto com seus respectivos cenários de execução realizados pelos especialistas *Espec. 1*, *Espec. 2*, *Espec. 3* e *Espec. 4*, respectivamente. É possível observar que todos os quatro especialistas detectaram *exatamente* todos os quatro pares de normas com potenciais conflitos normativos do exemplo do estudo de caso, cujos resultados foram apresentados na Tabela 2, ou seja, conflitos entre as normas (i) *N2* e *N3*, normas (ii) *N3* e *N4*, normas (iii) *N3* e *N6* e entre as normas (iv) *N8* e *N9*. Portanto, todos os quatro especialistas identificaram que tais pares de normas estão em conflito devido (i) a existência de alguma interseção entre seus períodos de ativação, (ii) a presença de conceitos deônticos contraditórios (isto é, obrigação *versus* proibição ou permissão *versus* proibição) entre tais pares de normas, (iii) estarem regulando uma mesma entidade e (iv) um mesmo comportamento, e (v) estarem definidas em um mesmo contexto.

Em relação aos cenários de execução, pode-se notar que todos os quatro especialistas identificaram *exatamente* todos os cenários de execução referentes aos pares de normas *N2* e *N3*, *N3* e *N6*, e *N8* e *N9* em relação ao resultado apresentados na Tabela 2. A única exceção foi a identificação dos cenários de execução das normas *N3* e *N4*. Somente a avaliação do especialista *Espec. 1* conseguiu identificar *exatamente* todos os cenários. Os demais especialistas identificaram apenas dois cenários conflitantes para o par *N3* e *N4*: "*Fulfillment of Norm 1 < Execution of Serve Lunch By Dora < Situation Done Lunching By Riley*" e "*Execution of Serve Lunch By Dora < Fulfillment of Norm 1 < Situation Done Lunching By Riley*". De acordo com a avaliação dos especialistas *Espec. 2* e *Espec. 3* e *Espec. 4*, foi considerado que, na norma *N4*, o evento *serveLunch* acontece sempre antes de *doneLunching*. Inclusive foi destacado no campo de observações das avaliações dos especialistas *Espec. 2* e *Espec. 4* (Tabela 8). De fato, para que um agente termine de almoçar (*doneLunching*), é preciso que antes tenha-se servido o almoço (*serveLunch*). Portanto, esta análise está dentro do esperado.

É importante ressaltar que os especialistas não identificaram nenhum outro par de normas além dos quatro pares já identificados e nenhum outro cenário de execução além dos

identificados na Tabela 2, ou seja, não tiveram falsos negativos (assumindo que os especialistas são a verdade). Os resultados das avaliações dos especialistas foram satisfatórios e ajudaram a validar os resultados da detecção dos potenciais conflitos normativos entre pares de normas.

É importante observar também que a avaliação manual feita por especialistas também pode falhar, justamente por não conseguirem ver algum aspecto identificado pelo sistema dado a complexidade e/ou grande quantidade de normas. Portanto, um sistema automatizado para verificação de PNC é de grande importância para prevenir os conflitos entre normas nos SMA.

Tabela 4. Avaliação do Especialista 1.

Especialista	**Normas em conflitos identificadas**	**Cenários de execução identificados**
Espec. 1	*N2, N3*	*Take a Shower* = a, Cumprimento de *N1* = b. b<a
	N3, N4	Cumprimento *N1* = a, *serveLunch* = b, *doneLunching* = c. a<b<c; a<c<b; b<a<c; c<a<b; c<b<a
	N3, N6	Cumpri *N1* = a, Violação *N5* = b. a<b; b<a
	N8, N9	Cumprir *N7* = a, *serveLunch* = b. a<b; b<a

Tabela 5. Avaliação do Especialista 2.

Especialista	**Normas em conflitos identificadas**	**Cenários de execução identificados**
Espec. 2	*N2* e *N3*	"Cumprir com *N1* < Executar *TakeAShower*"

	N3 e *N4*	"Cumprir com *N1* < Executar *ServeLunch* < *doneLunching*" "Executar *ServeLunch* < Cumprir com *N1* < *doneLunching*"
	N3 e *N6*	"Cumprir com *N1* < Violar *N5*" "Violar *N5* < Cumprir com *N1*"
	N8 e *N9*	"Cumprir com a norma *N7* < Executar *ServeLunch*" "Executar *ServeLunch* < Cumprir com a norma *N7*"

Tabela 6. Avaliação do Especialista 3.

Especialista	**Normas em conflitos identificadas**	**Cenários de execução identificados**
Espec. 3	*N2* e *N3*	1) cumprimento de *N1* < executar *takeShower*
	N3 e *N4*	1) executar *serveLunch* < cumprimento *N1* < situação *doneLunching* 2) cumprimento *N1* < executar *serveLunch* < situação *doneLunching*
	N3 e *N6*	X = cumprimento a norma *N1*, Y = violar a norma *N5* 1) X >= Y 2) Y >= X
	N8 e *N9*	X = cumprimento a norma *N7*, Y = executar *serveLunch* 1) X >= Y 2) Y >= X

Tabela 7. Avaliação do Especialista 4.

Especialista	**Normas em conflitos identificadas**	**Cenários de execução identificados**

Espec. 4	N2 e N3	Evento Z < Evento K
	N3 e N4	Evento Z < Evento w1 < Evento w2 OU Evento w1 < Evento Z < Evento w2
	N3 e N6	Evento Z < Evento T OU Evento T < Z
	N8 e N9	Evento U < Evento S OU Evento S < Evento U

Tabela 8. Observações dos Especialistas sobre a Atividade.

Especialista	Observações sobre a atividade
Espec. 1	-
Espec. 2	Na N4, considerei que o evento "doneLunching" vem sempre após o evento "executar serveLunch".
Espec. 3	-
Espec. 4	Evento K - " executar takeShower"; Evento Z - "cumprimento de N1"; Evento w1 - "Dora executar serveLunch"; Evento w2 - "situação doneLunching verdadeira para Riley"; Evento T - " violar norma N5"; Evento U - "cumprir norma N7"; Evento S - " Dora executar serveLunch". Considerou-se nesta análise que sempre Evento w1 < Evento w2.

CAPÍTULO 5 – TRABALHOS RELACIONADOS

Vários trabalhos têm investigado mecanismos para detectar conflitos normativos em SMA. Alguns deles lidam com a identificação de conflitos normativos diretos, tais como VASCONCELOS *et al.* (2012), LI *et al.* (2014), GÜNAY E YOLUM (2013), DOS SANTOS NETO *et al.* (2013), e SILVESTRE e DA SILVA (2015, 2016). Outros trabalhos lidam também com a detecção de conflitos indiretos, tais como LAM *et al.* (2008a), SENSOY *et al.* (2010, 2012), USZOK *et al.* (2008) e BRADSHAW *et al.* (2013), DA SILVA e ZAHN (2014), DA SILVA *et al.* (2015) e SANTOS e DA SILVA (2016a, 2016b). Os conflitos normativos diretos e indiretos foram apresentados na Seção 2.1.2.

No entanto, até onde conhecemos, não existem trabalhos que proponham mecanismos em *tempo de design* para a detecção de conflitos que *podem* acontecer entre duas normas durante a execução de um sistema multi-agente, chamados aqui de *Potenciais Conflitos Normativos*, dependendo da ordem de execução dos eventos do sistema. A seguir serão apresentados alguns trabalhos relacionados e uma comparação deles com a tese.

LAM *et al.* (2008a) propuseram uma abordagem que usa regras SWRL e OWL DL para representar aspectos normativos em uma organização governada por normas, incluindo a representação de papeis e seus relacionamentos (hierarquia de papeis), normas (permissões, proibições e obrigações) com condições e deadlines, e violações e detecções de conflitos (uma ação sendo simultaneamente obrigada e proibida). Segundo os autores, uma organização baseada em normas pode ser expressada em termos de uma sociedade de agentes autônomos, um conjunto de normas para a sociedade, um conjunto de papéis que cada membro pode desempenhar, relacionamentos entre estes papéis, e uma estrutura da sociedade.

Os autores representaram os papéis e hierarquia de papéis dos agentes usando *concept inclusion* e *restrições nos papéis* (*role restrictions*, em inglês). Para mais detalhes de *concept inclusion* e *restrições nos papéis*, verificar seção 2.2.2.2. Por exemplo, para definir a seguinte hierarquia de papeis: (i) *Programmer* ou *Manager* são Staff, (ii) *DeptManager* ou *GeneralManager* são *Manager*, e (iii) *AccountingManager* é um *DeptManager*, pode-se escrever assim.

$$\texttt{Programmer} \sqcup \texttt{Manager} \sqsubseteq \texttt{Staff}$$

$$\texttt{DeptManager} \sqcup \texttt{GeneralManager} \sqsubseteq \texttt{Manager}$$

$$\texttt{AccountingManager} \sqsubseteq \texttt{DeptManager}$$

No exemplo, o papel *AccountingManager* (sub-papel) é mais específico que o papel *Manager* (super-papel). Os papéis mais específicos herdam as propriedades dos papéis mais gerais. Por exemplo, para definir que todo *Staff* trabalha em apenas um departamento, pode-se definir os seguintes axiomas.

$$Staff \sqsubseteq =1\ worksIn$$

$$range(worksIn) = Department$$

Para definir que um agente não pode ser ao mesmo tempo *DeptManager* e *GeneralManager*, simultaneamente, os autores usaram o seguinte axioma.

$$Disjoint(DeptManager, GeneralManager)$$

Para definir *separation of duty*, os autores usaram regra SWRL para definir, por exemplo, que um *staff* submetendo uma proposta de projeto não pode ser o mesmo que aprova a proposta, como mostrado abaixo.

$$Staff(x) \wedge ProjectProposal(p) \wedge submits(x,p) \wedge approves(x,p) \wedge O(x) \wedge O(p) \longrightarrow owl{:}Nothing(x)$$

A sintaxe para permissões, obrigações e proibições, todas sem condição, ou seja, a norma é ativa todo o tempo, é dada como segue:

$$Agent \sqsubseteq \exists\ isPermitted.Act$$

$$Agent \sqsubseteq \exists\ isObliged.Act$$

$$Agent \sqsubseteq \exists\ isProhibited.Act$$

onde *Agent* e *Act* são conceitos OWL representando um agente e uma ação, respectivamente, e *isPermitted*, e *isObliged* e *isProhibited* são propriedades *object properties*, cujos *domain* e *range* é *Agent* e *Act*, respectivamente. Então, por exemplo, para especificar que todo *Staff* é obrigado a trabalhar em dias da semana (*Weekdays*), das 9am até as 5pm, os seguintes axiomas podem ser declarados.

$$Staff \sqsubseteq \exists\ isObliged.(\exists\ works.(\exists\ hasDays.Weekdays \sqcap \exists\ hasHour.OfficeHour))$$

$$Weekdays \equiv \{Monday\} \sqcup \{Tuesday\} \sqcup \{Wednesday\} \sqcup \{Thursday\} \sqcup \{Friday\}$$

$$OfficeHour \equiv \exists\ starts.\{"09:00:00"\text{^^}<xsd:time>\} \sqcap \exists\ ends.\{"17:00:00"\text{^^}<xsd:time>\}$$

Os autores também definiram normas condicionais, onde a condição pode ser um fato na base ou um *deadline*. Os seguintes axiomas foram introduzidos para representar normas condicionais, cujo conceito deôntico é uma obrigação.

$$Agent \sqsubseteq \exists\ isObliged.(Act \sqcap \exists\ hasCond.Cond)$$

$$Agent \sqsubseteq \exists\ isObliged.(Act \sqcap \exists\ before.Deadline)$$

$$Agent \sqsubseteq \exists\ isObliged.(Act \sqcap \exists\ after.Deadline)$$

Os nomes *before* e *after* são propriedades *object properties*, cujos *domain* e *range* é *Act* e *xsd:dateTime*, respectivamente. Por exemplo, para definir que todo *Staff* é permitido tirar licença médica (*SickLeave*) caso receba uma aprovação médica, o seguinte axioma pode ser declarado.

$$Staff \sqsubseteq \exists\ isPermitted.(\exists\ takes.SickLeave \sqcap \exists\ hasCond.DoctorApproval)$$

O próximo exemplo mostra uma norma com condição do tipo *deadline*. Ela define que todo *ITStaff* é obrigado a atualizar o servidor web antes de um *deadline*.

$$ITStaff \sqsubseteq \exists\ isObliged.(\exists\ upgrades.WebServer \sqcap \exists\ before.Deadline)$$

Para detectar se o tempo é antes ou depois do *deadline*, os autores fizeram uso de regras SWRL com funções embutidas de tempo. Para realizar a detecção de conflitos entre uma permissão e uma proibição, e entre uma obrigação e uma proibição, os autores definiram os seguintes axiomas.

$$isPermitted(x,act) \wedge isProhibited(x,act) \wedge O(x) \wedge O(act) \longrightarrow owl{:}Nothing(x)$$

$$isObliged(x,act) \wedge isProhibited(x,act) \wedge O(x) \wedge O(act) \longrightarrow owl{:}Nothing(x)$$

Se uma ação é ao mesmo tempo permitida e proibida para um agente, então é gerado uma inconsistência na ontologia OWL. O mesmo ocorre entre uma obrigação e uma proibição. Conflitos são detectados quando a ontologia OWL tem alguma inconsistência. Os autores usaram técnicas de depuração em ontologias (LAM *et al.*, 2008b) para identificar os axiomas inconsistentes na ontologia.

Comparando com a abordagem da proposta na tese, os autores representaram a entidade regulada pela norma como um papel, na tese estamos representando entidades como agentes. A abordagem proposta pelos autores possui uma série de limitações que foram tratadas na tese: (i) a representação da norma pelos autores não permite estar relacionada com ambas as propriedades *before* e *after*; (ii) não há a representação de *estado de ativação* e *estado de obediência* (detalhes nas seções 3.1.1.7 e 3.1.1.8, respectivamente), os autores removem da ontologia uma obrigação quando ela é cumprida, para informar que essa obrigação não existe

mais; (iii) não há a representação da violação de uma permissão; (iv) a detecção de conflitos especificada pelos autores é capaz de identificar *conflitos indiretos*, porém não detecta *potencias conflitos normativos*; e (v) as condições em tempo de execução, tais como execução de uma ação por um agente, cumprimento/violação de uma norma e ativação/desativação de uma norma, não são suportadas na abordagem proposta pelos autores.

SENSOY *et al.* (2010, 2012) desenvolveram uma linguagem chamada OWL-POLAR, baseada na linguagem OWL-DL, para representar políticas (ou normas, como chamadas na tese) em sistemas baseados em agentes. OWL-POLAR é um acrônimo para *OWL-based POlicy Language for Agent Reasoning*. Os autores definiram política condicionais através da expressão $\alpha \rightarrow N_{x:\rho}(a:\varphi)/e$, onde α é uma conjunção de formulas semânticas que representa a condição de ativação da política; $N \in \{O, P, F\}$ indica se a norma é uma obrigação, permissão ou proibição; x é uma variável que representa uma entidade na qual está sendo regulada pela política; ρ descreve x usando classes da ontologia, por exemplo, $?x : Student(?x) \wedge Female(?x)$; a é uma variável que representa a ação que será regulada pela norma e φ descreve a ação a usando classes e propriedades da ontologia, como por exemplo $?a : SendFileAction(?a) \wedge hasReciver(?a, John)$ onde $SendFileAction$ é uma classe que representa a ação de enviar um arquivo; por fim, e é uma conjunção de fórmulas semânticas que define uma condição de expiração. A Figura 18 ilustra um exemplo de política representada um OWL-POLAR que especifica que uma pessoa é obrigada a deixar um local caso este tenha risco de incêndio.

α	$Place(?b) \wedge hasFireRisk(?b, true) \wedge in(?x, ?b)$
N	O
$\chi : \rho$	$?x : Person(?x)$
$a : \varphi$	$?a : LeavingAction(?a) \wedge about(?a, ?b) \wedge hasActor(?a, ?x)$
e	$hasFireRisk(?b, false)$

Figura 18: Exemplo de política em OWL-POLAR (SENSOY *et al.*, 2012)

Os autores propuseram um algoritmo em tempo de design para fazer a verificação de conflitos. Dada duas normas $P_i = \alpha^i \rightarrow A_{x^i:\rho^i}(a^i:\varphi^i)/e^i$ e $P_j = \alpha^j \rightarrow B_{x^j:\rho^j}(a^j:\varphi^j)/e^j$, o algoritmo consiste em fazer as seguintes verificações.

1. $\Delta \vdash (\alpha^i \wedge \rho^i).\ \sigma_i$, mas não existe σ'_i tal que $\Delta \vdash (e^i \wedge \sigma_i).\ \sigma'_i$
2. $\Delta \vdash (\alpha^j \wedge \rho^j).\ \sigma_j$, mas não existe σ'_j tal que $\Delta \vdash (e^j \wedge \sigma_j).\ \sigma'_j$
3. $x^i.\ \sigma_i = x^j.\ \sigma_j$

4. $(\varphi^i.\sigma_i) \sqsubseteq (\varphi^j.\sigma_j)$ ou $(\varphi^j.\sigma_j) \sqsubseteq (\varphi^i.\sigma_i)$
5. $A \in \{P, O\}$, enquanto que $B \in \{F\}$, ou vice versa.

Os itens (1) e (2) verificam se as normas P_i e P_j estão ativas, respectivamente. De acordo com SENSOY *et al.* (2012), uma norma é ativada para um agente quando existe um estado de mundo tal que a condição de ativação seja satisfeita para esse agente, porém a condição de expiração não seja satisfeita. O símbolo Δ representa um estado de mundo baseado na ontologia de domínio. Para afirmar que a norma P_i está ativa, verifica-se se existe uma substituição σ_i tal que $\Delta \vdash (\alpha^i \wedge \rho^i).\ \sigma_i$ seja verdade e que não existe uma substituição σ'_i tal que $\Delta \vdash (e^i \wedge \sigma_i).\ \sigma'_i$. Os autores convertem as formulas semânticas conjuntivas em consultas SPARQL para verificar se a norma está ativada ou desativada (Figura 19). O item (3) verifica se as normas P_i e P_j referem-se a um mesmo agente e o item (4) verifica se P_i e P_j aplicam-se à mesma ação. Por fim, o item (5) verifica se as normas P_i e P_j tem conceitos deônticos conflitantes.

```
Query:

  q(?x, ?b):-
    Place(?b) ∧
    hasFireRisk(?b, true) ∧
    Person(?x) ∧
    in(?x,?b).
```

```
SPARQL SYNTAX:

  PREFIX example: <http://www.example.com/ns#>
  PREFIX rdf: <http://www.w3.org/...rdf-syntax-ns#>
  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
  SELECT ?x ?b
  WHERE {
    ?b rdf:type example:Place.
    ?b example:hasFireRisk "true"^^xsd:boolean.
    ?x rdf:type example:Person.
    ?x example:in ?b.
  }
```

Figura 19: Consulta SPARQL para verificar a ativação da norma.

Comparando o trabalho dos autores com a tese, observa-se que os autores representaram nas condições de ativação e desativação como fórmulas semânticas conjuntivas, que são fatos na base de dados. Porém, não é possível representar outras condições, tais como a execução de uma ação por um agente, ativação/desativação de uma norma ou cumprimento/violação de uma norma. Os autores não levaram em consideração normas com condições *antes* ou condições *depois*. A detecção de conflitos especificada pelos autores é capaz de identificar *conflitos indiretos*, porém não detecta *potencias conflitos normativos*. Além disso, os autores não propuseram uma representação semântica dos principais conceitos de uma norma.

USZOK *et al.* (2008) e BRADSHAW *et al.* (2013) desenvolveram um *framework*, baseado em OWL, para especificar, analisar e gerenciar normas, chamado de KAoS. Esse

framework possui um conjunto de ontologias bases para definir conceitos normativos, e suas propriedades, tais como entidade (*Actor*), ações, estados, lugares, papeis, etc. KAoS suporta dois tipos principais de normas: autorização e obrigação, e ambas podem ser classificadas como positiva ou negativa. As normas de autorização que definem ações permitidas são chamadas de *autorização positiva*, e as que definem proibições são chamadas de *autorização negativa*. Por outro lado, as normas de obrigação que especificam ações obrigatórias são classificadas como *obrigações positivas* e as que definem ações, cuja obrigatoriedade destas ações deve ser evitada, são chamadas de *obrigações negativas*.

A forma básica de políticas em KAoS é mostrada, como segue:

```
[Actor] is [constrained] to perform
[controlled action] which has [any attributes]
```

onde *[Actor]* é uma variável que se refere ao sujeito da ação sendo controlada pela política, que pode ser um simples agente ou um papel; *[constrained]* é uma variável que se refere ao tipo de política (*autorização positiva* ou *negativa* e *obrigação positiva* ou *negativa*); *[controlled action]* é uma variável que se refere a uma ação controlada pela política; e *[any attributes]* é uma variável opcional que se refere a um ou mais atributos da ação controlada.

O *framework* KAoS é capaz de fazer a detecção de conflitos indiretos, porém os autores não detalham como a detecção é realizada. KAoS possui uma estratégia de resolução de conflitos entre normas que consiste em configurar uma ordem de importância entre as normas. Todas as normas, quando criadas ou atualizadas, devem possuir uma ordem de prioridade em relação as outras. Quando duas normas estão conflito, aquela que tiver maior prioridade será executada. Caso duas normas em conflito tenham prioridades iguais, KAoS oferece sugestões ao usuário baseado no trabalho de USZOK (2003).

Comparando a abordagem dos autores com a tese, KAoS não permite definir condições que ocorrem em tempo de execução, tais como as definidas nesta tese. Além disso, KAoS é capaz de identificar *conflitos indiretos*, porém não detecta *potencias conflitos normativos*. Também, condições *antes* ou condições *depois* não são suportadas em KAoS.

DA SILVA e ZAHN (2014) e DA SILVA *et al.* (2015) apresentaram uma abordagem, baseado em ontologia OWL, para detectar conflitos normativos diretos e indiretos entre duas normas que regulam a execução de ações diferentes, porém relacionadas, e que controlam o comportamento de entidades diferentes, mas que estão relacionadas entre si. Relacionamentos entre ações e entre entidades tais como, hierarquia, participação em um papel, refinamento, composição, ortogonalidade e dependência, foram abordados pelos autores. Tais conflitos

dependem de informações do domínio da aplicação para serem detectados, ou seja, é necessário saber como os elementos que compõem uma norma estão relacionados entre si.

De acordo com os autores, uma norma n é definida, como segue:

```
{deoC, c, e, a, ac, dc, s}
```

onde $deoC$ representa o conceito deôntico da norma, podendo ser uma obrigação, proibição ou permissão; $c \in C$ representa o contexto onde a norma foi definida; $e \in E$ é a entidade cujo comportamento está sendo controlado, podendo ser um agente, um papel ou uma organização; $a \in A$ é a ação que está sendo regulada; $ac \in Cd$ define a condição que ativa a norma; $dc \in Cd$ é a condição que desativa a norma; e s indica o estado da norma podendo ser um elemento do conjunto $\{cumprido, violado, nenhum\}$. As condições de ativação e desativação da norma são representadas por números naturais, indicando um intervalo de tempo. O estado $nenhum$ especifica que a norma não foi cumprida e nem violada. O contexto representa o escopo onde a norma foi definida. Segundo os autores, o contexto pode ser definido no escopo de uma organização ou um ambiente. Se a norma não especificar um contexto, significa que ela é aplicada a todos os contextos. Uma norma deve ser cumprida somente dentro de seu contexto de aplicação.

Os autores definiram várias regras que especificam relacionamentos de transitividade entre as normas levando em consideração os relacionamentos entre as entidades e entre as ações. Tais regras foram usadas para verificar se duas normas estão em conflito. Um verificador de conflitos foi implementado usando a linguagem Java para validar a abordagem proposta usando, como exemplo, o domínio de uma universidade. O programa recebe como entrada as normas e a especificação do domínio do sistema, descritas através de uma ontologia OWL, e retorna aquelas normas que estão em conflito.

Comparando o trabalho dos autores com a tese, pode-se observar que os autores representaram a entidade regulada pela norma como um agente, papel ou uma organização (grupo de agentes); na tese estamos representando entidades apenas como agentes. No entanto, condições que podem ocorrer em tempo de execução, tais como a execução de uma ação por um agente, ativação/desativação de uma norma ou cumprimento/violação de uma norma, não são suportadas na abordagem proposta pelos autores. Além disso, o trabalho dos autores é capaz de detectar *conflitos diretos* e *indiretos*, porém não detecta *potencias conflitos normativos*. Também, a descrição das classes e propriedades da ontologia OWL que define os conceitos de uma norma não foi apresentada pelos autores.

No trabalho de SILVESTRE e DA SILVA (2015, 2016), foi proposto um algoritmo para detectar conflitos entre múltiplas normas, e não apenas entre pares de normas. A definição da norma segue uma representação similar à abordagem apresentada no trabalho de DA SILVA e ZAHN (2014) e DA SILVA *et al.*, (2015), exceto a especificação de um comportamento, onde o mesmo pode ser composto por uma ação, um objeto, e um conjunto de parâmetros e valores. O comportamento é definido pela tupla:

```
{action, object (param1=value1,. . . , paramN=valueN)}
```

onde *action* é a ação sendo regulada pela norma, *object* é o objeto da ação seguindo por uma lista de parâmetros da ação, com seus respectivos valores.

O algoritmo de detecção de conflitos identifica conflitos diretos entre múltiplas normas foi dividido em três passos. No primeiro passo, todas as normas são transformadas em permissões. Depois as normas são filtradas e colocadas em grupos por similaridade, ou seja, ficam no mesmo grupo aquelas normas que possuem contextos iguais, regulam o mesmo comportamento ou governam a mesma entidade. Somente as normas que estão no mesmo grupo podem estar em conflito. No terceiro passo, o algoritmo faz a verificação de conflito em cada grupo, verificando todas as possibilidades. O algoritmo começa verificando as normas de dois em dois e vai aumentando a quantidade até o total de normas no grupo. A complexidade do algoritmo no melhor caso é $O(1)$ e no pior caso é $O(2^n)$, onde n é o número de normas.

Comparando o trabalho dos autores com a tese, SILVESTRE e DA SILVA (2015, 2016) propuseram uma abordagem de detecção de conflitos entre múltiplas normas, enquanto que a abordagem proposta nesta tese detecta conflitos normativos apenas entre pares de normas. Além disso, as ações representadas nesta tese são atômicas, diferente da ação parametrizada proposta pelos autores. Porém, a abordagem proposta pelos autores detecta apenas *conflitos diretos* entre múltiplas normas, e não *potencias conflitos normativos*.

DOS SANTOS NETO *et al.* (2013) propuseram uma arquitetura abstrata chamada NBDI (*Norm-Belief-Desire-Intention*) para criar agentes orientados a objetivos capazes de lidar com normas sociais. A arquitetura NBDI estende a arquitetura BDI (*Belief-Desire-Intention*) incluindo funções que permitam os agentes raciocinarem sobre as normas, tais como, (i) uma função para incluir na base de crenças as normas percebidas no sistema, (ii) uma função para selecionar as normas que os agentes têm a intenção de cumprir baseada nos benefícios que elas podem produzir na realização de seus desejos e intenções, (iii) funções para identificar e resolver conflitos entre as normas que foram selecionadas e (iv) funções para selecionar os

desejos que irão se tornar intenções e os planos para alcançar tais intenções, levando em consideração as normas que os agentes querem cumprir.

A representação das normas segue a seguinte estrutura: *norma (destinatário, ativação, expiração, premiações, punições, conceito deôntico, estado)*, onde a *destinatário* é responsável pelo cumprimento da norma e pode ser representado por um agente ou um papel; *ativação* é a condição de ativação que torna a norma ativa; *expiração* é a condição de expiração da norma para deixá-la inativa; *premiações* são recompensas dadas àqueles agentes que cumpriram uma norma; *punições* são penalidades que podem ser aplicadas àqueles agentes que violaram uma norma; *conceito deôntico* indica se a norma é uma obrigação ou uma proibição (as permissões não são representadas); e *estado* descreve a ação sendo regulada pela norma.

A algoritmo de detecção de conflitos entre normas verifica se duas normas diferentes, uma sendo uma obrigação e a outra sendo uma proibição, regulam a mesma ação e o mesmo destinatário. Depois é verificado se tais normas estão no conjunto de normas que deverão ser cumpridas ou violadas pelo agente, de acordo com os benefícios trazidos por elas para a realização de seus desejos e intenções. Caso o agente tenha a intenção de cumprir uma das normas, mas violar a outra, então tais normas não estão em conflito. No entanto, caso o agente pretenda cumprir (ou violar) ambas as duas normas, então elas estão em conflito e devem ser resolvidas. O algoritmo de resolução de conflitos entre duas normas seleciona uma das normas para ser cumprida e a outra para ser violada, usando o seguinte critério: aquela norma que tiver maior contribuição na realização de seus desejos e intenções prevalece para ser cumprida, deixando a outra para ser violada. Se as contribuições das normas forem iguais, então o algoritmo seleciona qualquer uma delas.

Comparando a abordagem proposta por DOS SANTOS NETO *et al.* (2013) com a abordagem proposta na tese, as condições de ativação e desativação propostas pelos autores são fatos na base de dados. Porém, não é possível representar outras condições, tais como a execução de uma ação por um agente, ativação/desativação de uma norma ou cumprimento/violação de uma norma. Os autores propuseram um algoritmo de detecção de apenas *conflitos diretos*, e não identificam *potencias conflitos normativos*.

GÜNAY E YOLUM (2013) identificaram vários tipos de conflitos entre diferentes papeis em um sistema multi-agentes baseado em organização e apresentaram alguns métodos de como lidar com tais conflitos. Os papeis descrevem o que um agente deve ou não fazer e são elementos fundamentais em uma organização. As organizações existem para realizar tarefas e objetivos específicos e os papeis em uma organização devem ser livres de conflitos. Conflitos

podem ocorrer na especificação de um papel como também entre papéis diferentes sendo executados por um mesmo agente. Por exemplo, em uma conferência representando uma organização, o papel de um membro do comitê de programa não deve simultaneamente obrigar e proibir um agente de revisar um artigo. Isso geraria um conflito impedido o agente de cumprir com os requerimentos do papel e fazendo com que a organização falhe em atingir os seus objetivos.

A especificação dos papeis é representada por conjuntos de *compromissos*. Segundo os autores, os conflitos entre compromissos são bastante relacionados com conflitos entre normas. Um compromisso é expressado por $C(x, y, q, p)$ que é interpretado como uma obrigação de um agente *devedor x* para um agente *credor y* de produzir o *consequente p*, caso o *antecedente q* aconteça. Os agentes são representados por variáveis (ex.: *x*, *y*) e o *antecedente* e o *consequente* do compromisso são representados por proposições (ex.: *p*, *q*, *u*, *w*). O *antecedente* de um compromisso pode ser formado por uma conjunção de preposições, porém o *consequente* é formado por uma proposição simples. Se a proposição é uma condição que deve ser mantida em constante execução, ela é representara com uma linha no topo (ex.: $\bar{p}$). Quando o *antecedente* torna-se verdade, o compromisso é ativado e o antecedente é substituído pelo símbolo $\top$ (ou seja, $C(x, y, \top, p)$). Quando o *consequente* acontece, o compromisso é cumprido. Obrigações e proibições são representados por compromissos separados: $C(x, y, q, p)$ para obrigações e $C(x, y, q, \neg p)$ para proibições.

Um papel é definido pela tupla $\langle \lambda, \mathbb{L}, \mathbb{C} \rangle$, onde λ é um nome único para o papel, $\mathbb{L}$ é uma formula que define as condições que deixam o papel ativado e passivo (ou desativado), e $\mathbb{C}$ é um conjunto de compromissos que o agente deve cumprir. A Figura 20 mostra um exemplo da especificação simplificada do papel de um membro do comitê de programa em uma conferência. O nome do papel é *PCMember* que é ativado quando a submissão de artigos está encerrada e desativado quando os autores são notificados sobre o resultado. O agente que desempenha o papel de *PCMember* é obrigado a revisar os artigos alocados a ele pelo *chair* do programa (representado pelo primeiro compromisso) e proibido de revisar artigos se algum dos autores é seu colega (representado pelo segundo compromisso).

λ	PCMember
$\mathbb{L}$	*Activate*:{SubmissionClosed}, *Passivate*:{AuthorsNotified}
$\mathbb{C}$	C(PCMember, Chair, PaperAssigned, PaperReviewed) C(PCMember, Chair, AuthorIsColleague, $\overline{\neg \text{PaperReviewed}}$)

Figura 20: Especificação do papel de um membro do comitê de programa (GÜNAY E YOLUM, 2013)

Em um sistema multi-agente, o agente pode participar de vários compromissos simultaneamente, e tais compromissos podem conflitar uns com os outros. Os autores definiram formalmente um conflito entre dois compromissos como segue. Dados dois compromissos $c_i = C(x, y, w, u)$ e $c_j = C(x, y', w', u')$, c_i e c_j estão em conflito, denotado por $c_i \otimes c_j$, se: (i) $(w \rightarrow q) \wedge (w' \rightarrow \neg q)$ for falso, e (ii) $(u \rightarrow p) \wedge (u' \rightarrow \neg p)$ for verdade. Na definição, fica claro que c_i é uma obrigação e c_j é uma proibição. Em outras palavras, para que dois compromissos estejam em conflito, é preciso que seus *consequentes* sejam inconsistentes e seus *antecedentes* não sejam inconsistentes. Por exemplo, os compromissos especificados no papel *PCMember* da Figura 20 estão em conflito, pois os seus *consequentes* são inconsistentes, enquanto que seus *antecedentes* não são. O conflito também poderia existir entre compromissos especificados em papeis diferentes e o agente participando em tais papeis, simultaneamente.

Comparando a abordagem proposta por GÜNAY E YOLUM (2013) com a abordagem proposta na tese, observa-se que as condições que definem a ativação e desativação da norma são apenas fatos na base de dados. Porém, não é possível representar outras condições, tais como a execução de uma ação por um agente, ativação/desativação de uma norma ou cumprimento/violação de uma norma. Os autores representaram a entidade regulada pela norma como um papel e na tese estamos representando entidades apenas como agentes. Por fim, os autores propuseram um algoritmo de detecção de *conflitos diretos*, e não de *potencias conflitos normativos*, como proposto nesta tese.

A Tabela 9 apresenta resumidamente as abordagens de detecção de conflitos normativos em SMA utilizadas nos trabalhos relacionados apresentados neste capítulo. Pode-se observar que nenhum dos trabalhos relacionados lidam com a detecção de potenciais conflitos normativos, exceto os três últimos trabalhos que são oriundos desta tese. Os artigos BELCHIOR e DA SILVA, (2017a) e BELCHIOR e DA SILVA, (2017b) tratam da especificação da estrutura normativa e da primeira abordagem de detecção de PCN. O artigo BELCHIOR e DA SILVA, (2017c) descreve a segunda abordagem de detecção de PCN.

Tabela 9. Resumo das abordagens de detecção de conflitos normativos em SMA utilizadas nos trabalhos relacionados.

Trabalhos relacionados	Conflitos diretos	Conflitos Indiretos	Potencias Conflitos Normativos
LAM *et al.* (2008a)		X	
SENSOY *et al.* (2010, 2012)		X	

USZOK *et al.* (2008) **BRADSHAW *et al.* (2013)**		X	
DA SILVA e ZAHN (2014) **DA SILVA *et al.* (2015)**	X	X	
SILVESTRE e DA SILVA (2015, 2016)	X		
DOS SANTOS NETO *et al.* (2013)	X		
GÜNAY E YOLUM (2013)	X		
BELCHIOR e DA SILVA, (2017a)			X
BELCHIOR e DA SILVA, (2017b)			X
BELCHIOR e DA SILVA, (2017c)			X

CAPÍTULO 6 – CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo apresentada as contribuições geradas com o desenvolvimento desta tese e sugestões para trabalhos futuros.

6.1 VISÃO GERAL DO TRABALHO, CONTRIBUIÇÕES E LIMITAÇÕES

Normas têm sido usadas em SMA como um mecanismo para controlar o comportamento de agentes, sem restringir a autonomia destes agentes. As normas definem ações que devem ser executadas (obrigações), ações que podem ser executadas (permissões), e ações que não podem ser executas (proibições) por determinada entidade em certa situação. Um dos desafios no projeto e gerenciamento de sistemas governados por normas é garantir que as normas especificadas não estejam em conflito. Duas normas estão em conflito quando o cumprimento de uma causa a violação da outra, e vice-versa. A detecção de conflitos normativos não é uma tarefa trivial para o designer do sistema, visto que os SMA podem regular uma grande quantidade de normas.

Este trabalho visa detectar uma categoria de conflitos ainda não pesquisada na área de conflitos normativos em SMA: detecção de *potenciais conflitos normativos*. Potenciais conflitos normativos são conflitos entre normas que *podem* acontecer durante a execução dos SMA, dependendo da ordem de execução dos eventos do sistema. Diferente das abordagens encontradas na literatura pesquisada, a proposta da tese é identificar em *tempo de design* conflitos normativos que *podem* ocorrer em *tempo de execução*. Portanto, a proposta apresentada neste trabalho permite identificar conflitos entre pares de normas antes mesmo deles de fato acontecerem em tempo de execução. Deste modo, com o apoio da ferramenta desenvolvida, o designer do sistema poderá reformular as normas a fim de evitar que tais conflitos não aconteçam na execução do sistema multi-agentes.

O Capítulo 2 apresentou os principais fundamentos teóricos necessários para o desenvolvimento desta tese. Para identificar os potenciais conflitos normativos, foram propostas duas abordagens baseadas em cenários de execução do sistema, discutidas no Capítulo 3. Na *primeira abordagem*, o designer fornece ao sistema normativo um conjunto de normas e possíveis exemplos de cenários de execução do sistema. Como saída, o sistema normativo retorna os conflitos normativos que aconteceriam caso tais cenários fossem executados no sistema multi-agentes. A vantagem da primeira abordagem é oferecer ao

designer do sistema uma forma dele reavaliar as normas em conflito dado um exemplo esperado (ou não) de execução do sistema. Na *segunda abordagem*, o designer do sistema fornece somente um conjunto de normas e o sistema normativo gera os cenários de execução que causariam os conflitos normativos caso tais cenários aconteçam no sistema multi-agentes. Ambas as abordagens fazem a detecção do conflito entre pares de normas.

Uma ferramenta, chamada *Potential Conflict Checker*, foi desenvolvida a fim de dar suporte à verificação dos potenciais conflitos normativos em um sistema multi-agentes (Seção 3.4). No Capítulo 4 foi apresentado um estudo de caso no domínio de um ambiente doméstico, a fim de ilustrar as abordagens de detecção de conflitos propostas na tese. Além disso, foram apresentadas as avaliações de especialistas na área de normas para validar os resultados da detecção dos potenciais conflitos normativos entre pares de normas.

Este trabalho teve como principais contribuições:

- Detecção de uma nova classificação de conflitos normativos, não investigada ainda na literatura, chamados aqui de *Potenciais Conflitos Normativos*.
- Especificação da estrutura normativa usando a linguagem OWL DL para representar os principais conceitos de uma norma em um sistema multi-agente;
- Especificação de regras SWRL para determinar os tempos referentes à execução de uma ação por um agente, ativação ou desativação de uma norma, cumprimento ou violação de uma norma e fatos no sistema que tenham se tornado verdade para um agente;
- Especificação de regras SWRL para fazer a verificação dos potenciais conflitos normativos entre uma obrigação e proibição e entre uma permissão e proibição;
- Implementação de uma ferramenta chamada *Potential Conflict Checker* para dar suporte à verificação dos potenciais conflitos normativos em um sistema multi-agentes. A ferramenta fornece ao usuário um ambiente de criação das normas, dos cenários de execução e apresentar a detecção dos conflitos de acordo com as duas abordagens descritas neste trabalho;
- Demonstração das abordagens de detecção dos potenciais conflitos normativos por meio de um estudo de caso;
- Realização da validação da detecção dos potenciais conflitos normativos por meio de especialistas na área de normas em SMA.

Uma das limitações deste trabalho é a *Suposição de Mundo Aberto* (em inglês, *Open World Assumption*) na qual as linguagens OWL DL e SWRL são baseadas. Por conta disso,

nem tudo pode ser automatizado utilizando inferências baseadas em Lógica de Descrição, como foi o caso da atualização dos tempos dos intervalos de ativação das normas e parte da justificativa do conflito (Algoritmo 1).

6.2 PUBLICAÇÕES

A seguir seguem os trabalhos científicos, relacionados com a proposta da tese, publicados, submetidos e aceitos para publicação.

Trabalhos publicados em congresso:

- O artigo "Detection of Normative Conflict that Depends on Execution Order of Runtime Events in Multi-Agent Systems" foi publicado no "IEEE/WIC/ACM International Conference on Web Intelligence (WI)", no ano de 2017.
- O artigo "Detection of Runtime Normative Conflict based on Execution Scenarios" foi publicado no "International Conference on Autonomic and Autonomous Systems (ICAS)", no ano de 2017
- O artigo "Detection of Runtime Normative Conflict in Multi-Agent Systems based on Execution Scenarios" foi publicado no "International Conference on Enterprise Information Systems (ICEIS)", no ano de 2017.

Trabalho aceitos para publicação em congresso:

- O artigo "Strategies for Resolving Normative Conflict That Depends on Execution Order of Runtime Events in Multi-Agent Systems" foi aceito para publicação no "International Conference on Agents and Artificial Intelligence (ICAART)" para o ano de 2018.

Trabalhos submetidos:

- O artigo "Identifying Potential Normative Conflicts in Multi-Agent Systems" foi submetido para publicação no "Journal of Autonomous Agents and Multiagent Systems (JAAMAS)".

6.3 TRABALHOS FUTUROS

Como sugestões de trabalhos futuros, apontamos as seguintes extensões e direcionamentos para essa pesquisa.

Estender a especificação de normas para suportar a definição de papeis nas entidades. As normas muitas vezes regulam o comportamento de papéis que são desempenhados por agentes. Tal extensão permitiria a norma ser aplicada a vários agentes que

desempenhassem o papel especificado na norma. Para que esta extensão seja feita, seria preciso fazer algumas alterações na estrutura normativa. Atualmente uma norma é cumprida ou violada para um agente específico. Caso a norma seja aplicada a um papel, o papel não pode cumprir com a norma, mas sim um agente que desempenha esse papel. Portanto, seria preciso ter um atributo que indicasse quais agentes que cumpriram/violaram com a norma. Seria interessante também ter uma ontologia de papeis que descrevesse quais são os papeis desempenhados pelos agentes.

Estender a abordagem proposta para permitir a especificação de normas com repetições de relacionamento *hasBefore* ou *hasAfter*, ou seja, normas com mais de um relacionamento *hasBefore* ou mais de um relacionamento *hasAfter*. Quando isso ocorre, o período de ativação real da norma pode ser calculado pela interseção dos períodos de ativação das condições com *hasAfter* (Figura 21) ou pela união dos períodos de ativação das condições com *hasBefore* (Figura 22). Por exemplo, suponha que a norma N1 tenha três condições X, Y e Z todas com relacionamentos do tipo *hasAfter*. O período de ativação real desta norma será a interseção dos períodos de ativação das três condições, ou seja, a norma N1 só será ativada quando as três condições X, Y e Z ocorrerem, como pode ser mostrado na Figura 21.

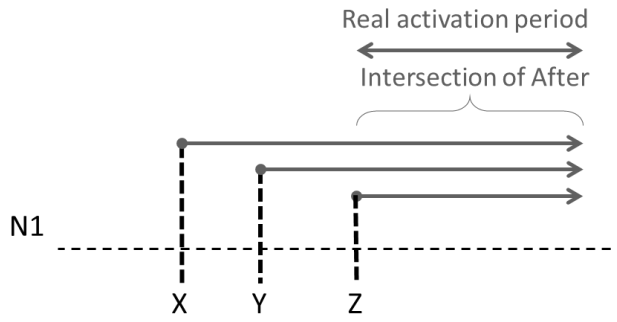


Figura 21: Norma com várias condições do tipo hasAfter

Por outro lado, a período de ativação de uma norma com vários relacionamentos do tipo *hasBefore* é dada pela união dos períodos de ativação de cada condição. Por exemplo, suponha agora que a norma N1 tenha três condições A, B e C todas com relacionamentos do tipo *hasBefore*. O período de ativação real desta norma será a união dos períodos de ativação das três condições, ou seja, a norma N1 estará ativada até que todas as três condições A, B e C tenham ocorrido no sistema, resultando na desativação da norma após isso (Figura 22).

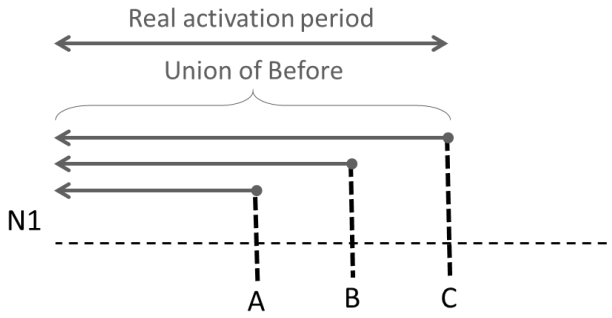


Figura 22: Norma com várias condições do tipo hasBefore

Estender a detecção de conflitos para identificar conflitos entre uma permissão e uma obrigação. Duas normas, cujos conceito deônticos são uma permissão e uma obrigação, estão em conflito quando uma obriga um agente a executar uma ação onde o agente não tem a permissão, ou seja, fora do intervalo de ativação da norma, cujo conceito deôntico é a permissão (norma não está ativa).

Investigar mecanismos para a resolução de PCN. Uma estratégia para a resolução do conflito seria eliminar as sobreposições entre os intervalos de ativação das normas com PCN. Uma sugestão seria alterar os relacionamentos *hasBefore* e *hasAfter* das condições das normas. Por exemplo, vamos supor que *n1* e *n2* sejam duas normas com PCN. Norma *n1* não tem condição associada a ela, ou seja, seu intervalo de ativação se inicia no tempo zero e dura até o termino da execução do sistema. A norma *n2* está associada a somente uma condição ("*after condition*") usando a propriedade *hasAfter*. Seu intervalo de ativação se inicia quando essa condição ocorrer no sistema e dura até o termino da execução do sistema. O mecanismo de resolução poderia remover a norma *n1* e criar uma nova norma *n1'* como uma cópia da *n1*, mas criando uma condição com relacionamento *hasBefore* ("*before condition"*) igual a condição com relacionamento *hasAfter* da norma *n2*. A Figura 23 mostra o resultado deste processo, mostrando que as normas resultantes, *n2* e *n1'*, não estão mais com PCN.

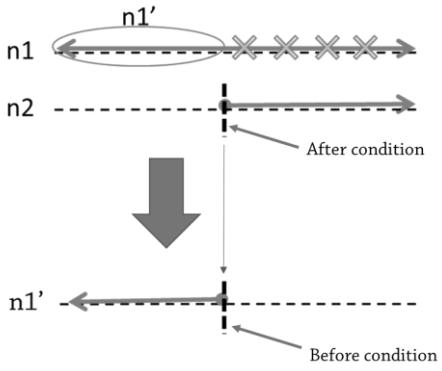


Figura 23: Eliminação das sobreposições entre os intervalos de ativação das normas com PCN.

Outra estratégia de resolução poderia resolver PNC evitando que tais conflitos aconteçam em tempo de execução. O mecanismo de resolução poderia estender um algoritmo de planejamento para que este crie planos que não produzam sequências de ações que possam causar PCN. O algoritmo de planejamento poderá testar se algum dos cenários de execução retornados pelo mecanismo de detecção está contido nas ações do plano e, caso esteja, o algoritmo de planejamento selecionaria outro plano. Neste caso, os cenários de execução devem ser formados ou representados por ações.

Estender a ferramenta *Potential Conflict Checker* para mostrar a representação gráfica dos conflitos normativos detectados. A representação gráfica do conflito, como a mostrada na coluna três da Tabela 2, auxiliaria o usuário no entendimento mais rápido do potencial conflito normativo. Atualmente a saída da ferramenta é em texto.

Aplicar a abordagem proposta em um cenário do mundo real. Uma sugestão seria testar as abordagens propostas nesta tese com um *dataset* ou estudo de caso famoso na literatura usados para testar outras abordagens. Isso permitiria revelar tipos de conflitos possivelmente complementares aos conflitos detectados por essas outras abordagens.

REFERÊNCIAS

ARTIKIS, A.; PITT, J. V. (2009) Specifying Open Agent Systems: A Survey. In A. Artikis, G. Picard, and L. Vercouter, eds. Engineering Societies in the Agents World IX. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 29-45.

BECKETT, David; BERNERS-LEE, Tim; PRUD'HOMMEAUX, Eric; & CAROTHERS, Gavin. RDF 1.1 Turtle: Terse RDF Triple Language. *W3C Recommendation*, W3C, February 2014. Available at <https://www.w3.org/TR/turtle/>.

BELCHIOR, M.; & DA SILVA, V. T. (2017a). Detection of Runtime Normative Conflict based on Execution Scenarios. In Thirteenth International Conference on Autonomic and Autonomous Systems (ICAS). May 21-25, 2017, Barcelona, Spain.

BELCHIOR, M.; & DA SILVA, V. T. (2017b) Detection of Runtime Normative Conflict in Multi-Agent Systems based on Execution Scenarios. In: 19th International Conference on Enterprise Information Systems (ICEIS), 2017, Porto, Portugal. p. 646.

BELCHIOR, M.; & DA SILVA, V. T. (2017c). Detection of Normative Conflict that Depends on Execution Order of Runtime Events in Multi-Agent Systems. In WI'17, IEEE/WIC/ACM International Conference on Web Intelligence. pp. 372-380. ACM, 2017. Leipzig, Germany.

BERNERS-LEE, T.; & CONNOLLY, D. Notation3 (N3): A readable RDF syntax. *W3C Team Submission*, March 2011. Available at <http://www.w3.org/TeamSubmission/n3/>.

BOELLA, G., & TORRE, L.V.D. A Game-Theoretic Approach to Normative Multi-Agent Systems. In Dagstuhl Seminar Proceedings 07122 - Normative Multi-Agent Systems. Schloss Dagstuhl, 2007.

BOELLA, Guido; VAN DER TORRE, Leendert; VERHAGEN, Harko. Introduction to normative multiagent systems. In: C*omputation and Mathematical Organizational Theory*, special issue on normative multiagent systems, 12(2-3):71–79, 2006.

BOELLA, Guido; VAN DER TORRE, Leendert; VERHAGEN, Harko. Introduction to normative multiagent systems. In: *Normative Multi-Agent Systems*, 2007.

BRADSHAW, J.M.; USZOK, A.; BREEDY, M.; BUNCH, L.; ESKRIDGE, T.; FELTOVICH, P.; JOHNSON, M.; LOTT, J.; & VIGNATI, M. (2013). The KAoS policy services framework. In Proc. 8th Cyber Security and Information Intelligence Research Workshop.

CONNOR, M. (2017). SWRLLanguageFAQ. Available at < https://github.com/protegeproject /swrlapi/wiki/SWRLLanguageFAQ >

DA SILVA, V. T. 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, v. 17, n. 1, p. 113-155, 2008.

DA SILVA, V. T.; BRAGA, C.; & ZAHN, J. (2015). Indirect Normative Conflict: Conflict that Depends on the Application Domain. In: *International Conference on Enterprise Information Systems* (ICEIS). Barcelona, 452-561.

DA SILVA, V. T.; & ZAHN, J. O. (2014). Normative conflicts that depend on the domain. In Balke, T., Dignum, F., van Riemsdijk, M. B., and Chopra, A. K., editors, Coordination, Organizations, Institutions, and Norms in Agent Systems IX, volume 8386 of Lecture Notes in Computer Science, pages 311–326. Springer International Publishing.

DEAN, M., SCHREIBER, G., BECHHOFER, S., VAN HARMELEN, F., HENDLER, J., HORROCKS, I., MCGUINNESS, D.L., PATEL-SCHNEIDER, P.F. and STEIN, L.A. (2004). OWL Web Ontology Language Reference. *W3C Recommendation. W3C*, February 2004. Available at <http://www.w3.org/TR/owl-ref/>.

DOS SANTOS NETO, B. F., DA SILVA, V. T., and de LUCENA, C. J. P. (2013). Developing goal-oriented normative agents: The NBDI architecture. In Filipe, J. and Fred, A., editors, Agents and Artificial Intelligence, volume 271 of Communications in Computer and Information Science, pages 176–191. Springer Berlin Heidelberg.

ELENIUS, D.; MARTIN, D.; FORD, R.; & DENKER, G. (2009). Reasoning about Resources and Hierarchical Tasks Using OWL and SWRL. In *International Semantic Web Conference* ISWC (pp. 795-810). Springer, Berlin, Heidelberg.

FIGUEIREDO, K. S.; DA SILVA, V. T., & BRAGA, C. O. (2011). Modeling norms in multi-agent systems with NormML. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, Lecture Notes in Computer Science 6541, Springer, Berlin, 39–57.

GENNARI, J.H., MUSEN, M.A., FERGERSON, R.W., GROSSO, W.E., CRUBÉZY, M., ERIKSSON, H., NOY, N.F. & TU, S.W., (2003). The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-computer studies*, 58(1), pp.89-123.

GÜNAY, A. and YOLUM, P. (2013). Engineering conflict-free multiagent systems. In First I*nternational Workshop on Engineering Multiagent Systems* (EMAS).

HARRIS, S.; SEABORNE, A.; and PRUD'HOMMEAUX, E., SPARQL 1.1 query language. *W3C recommendation*, vol. 21, no. 10, 2013.

HEBELER, J.; FISHER, M.; BLACE, R.; PEREZ-LOPEZ, A.; and DEAN, M. Semantic Web Programming. Indianapolis, IN: *Wiley*, 2009.

HOLLANDER, Christopher D.; WU, Annie S. (2011). The current state of normative agent-based systems. *Journal of Artificial Societies and Social Simulation*, v. 14, n. 2, p. 6.

HORRIDGE, M.; & BECHHOFER, S. (2011). The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1), 11-21.

HORRIDGE, M., & BRANDT, S. A Practical Guide to Building OWL Ontologies Using Protégé 4 and CO-ODE Tools, Edition 1.3. University of Manchester, 2011.

HORRIDGE, M.; KNUBLAUCH, H.; RECTOR, A.; STEVENS, R.; & WROE, C. "A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0." University of Manchester (2004).

HORROCKS, I., PATEL-SCHNEIDER, P. F., BECHHOFER, S., & TSARKOV, D. (2005). OWL rules: A proposal and prototype implementation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1), 23-40.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOF, B.; & DEAN, M. (2004). SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member submission*, 21, 79. Available at <http://www.w3.org/Submission/SWRL/>.

HORROCKS, Ian; PATEL-SCHNEIDER, Peter F.; MCGUINNESS, Deborah L.; & WELTY, Christopher A. OWL: a Description Logic Based Ontology Language for the Semantic Web. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition), chapter 14. Cambridge University Press, 2007.

KNUBLAUCH, H.; FERGERSON, R. W.; NOY, N. F.; & MUSEN, M. A. (2004). The Protégé OWL plugin: An open development environment for semantic web applications. In *International Semantic Web Conference* (Vol. 3298, pp. 229-243).

KRÖTZSCH, M., SIMANČÍK, F., and HORROCKS, I. (2012). A description logic primer. CoRR abs/1201.4089.

KRÖTZSCH, M., SIMANČÍK, F., & HORROCKS, I. (2014). Description logics. IEEE Intelligent Systems, 29(1), 12-19.

LAM, J. S. C.; GUERIN, F.; VASCONCELOS, W.; & NORMAN, T. J. (2008a, December). Representing and reasoning about norm-governed organisations with semantic web languages. In *Sixth European Workshop on Multi-Agent Systems Bath, UK* (pp. 18-19).

LAM, J.; SLEEMAN, D.; PAN, J.; & VASCONCELOS, W. (2008b). A fine-grained approach to resolving unsatisable ontologies. Journal on Data Semantics X, 4900:62-95, 2008.

LI, T.; JIANG, J.; ALDEWERELD, H.; DE VOS, M.; DIGNUM, V.; & PADGET, J. (2014) Contextualized institutions in virtual organizations. In: Coordination, Organizations, Institutions, and Norms in Agent Systems IX. Springer International Publishing, 2014. p. 136-154.

MEYER, J-J. C.; & WIERINGA. R. J. (eds.). Deontic logic in computer science: Normative system specification. John Wiley & Sons, 1993.

RUDOLPH, S. (2011). Foundations of description logics. In *Reasoning Web. Semantic Technologies for the Web of Data* (pp. 76-136). Springer Berlin Heidelberg.

SANTOS, J. S.; & DA SILVA, V. T. (2016a). Identifying Indirect Normative Conflicts using the WordNet Database. In Proceedings of the 18th *International Conference on Enterprise Information Systems* (ICEIS) - Volume 2. 186–193.

SANTOS, J. S.; & DA SILVA, V. T. (2016b). Identifying domain-independent normative indirect conflicts. In 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI) (pp. 536-543).

SANTOS, J. S.; ZAHN, J. O.; SILVESTRE; E. A.; SILVA, V. T.; & VASCONCELOS, W. W. (2017). Detection and resolution of normative conflicts in multi-agent systems: a literature survey. *Autonomous Agents and Multi-Agent Systems*, 1-47.

SENSOY, M.; NORMAN, T. J.; VASCONCELOS, W. W.; SYCARA, K. OWL-POLAR: A framework for semantic policy representation and reasoning. Web Semantics: Science, Services and Agents on the World Wide Web, v. 12, p. 148-160, 2012.

SENSOY, M., Norman, T. J., VASCONCELOS, W. W., and SYCARA, K. Owl-polar: Semantic policies for agent reasoning. In The Semantic Web – ISWC 2010, pages 679–695. Springer, 2010.

SHEARER, R.; MOTIK, B.; & HORROCKS, I. (2008). HermiT: A Highly-Efficient OWL Reasoner. In OWLED (Vol. 432, p. 91).

SILVESTRE, Eduardo Augusto; & DA SILVA, Viviane Torres. Verifying Conflicts Between Multiple Norms in Multi-agent Systems. In Proceedings of the 2015 *International Conference on Autonomous Agents and Multiagent Systems* (AAMAS'15), Richland, SC, p. 2013-2014, 2015.

SILVESTRE, E. A.; & DA SILVA, V. T. (2016). An Approach to Verify Conflicts among Multiple Norms in Multi-agent Systems. 2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI). Omaha, NE, USA, 367-374

SIRIN, E.; & PARSIA, B. (2007, June). SPARQL-DL: SPARQL Query for OWL-DL. In *OWLED* (Vol. 258).

SIRIN, E.; PARSIA, B.; GRAU, B. C.; KALYANPUR, A.; & KATZ, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2), 51-53.

SMITH, M. K.; WELTY, C.; & MCGUINNESS, D. L. (2004). OWL Web Ontology Language Guide. *W3C Recommendation, W3C*. Available at <https://www.w3.org/TR/owl-guide/>.

SYNAK, M., DABROWSKI, M., & KRUK, S. R. (2009). Semantic web and ontologies. In Semantic Digital Libraries (pp. 41-54). Springer Berlin Heidelberg.

USZOK, A., BRADSHAW, J. M., LOTT, J., BREEDY, M., BUNCH, L., FELTOVICH, P., JOHNSON, M., & JUNG, H. New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAoS. Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on. IEEE, 2008.

USZOK, A., BRADSHAW, J., JEFFERS, R., SURI, N., HAYES, P., BREEDY, M., BUNCH, L., JOHNSON, M., KULKARNI, S. & LOTT, J. (2003). KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction and Enforcement. Proceedings of the IEEE Fourth International Workshop on Policy (Policy 2003). Los Alamitos, CA: IEEE Computer Society, pp. 93-98.

VASCONCELOS, W.; GARCÍA-CAMINO, A.; GAERTNER, D.; RODRÍGUEZ-AGUILAR, J. A., & NORIEGA, P. (2012). Distributed norm management for multi-agent systems. *Expert Systems with Applications* 39, 5 (2012), 5990–5999.

WOOLDRIDGE, Michael. An introduction to multiagent systems. *John Wiley & Sons*, 2002.

ZAHN, J. O. & SILVA, V. T. On the Checking of Indirect Normative Conflicts. In: Workshop-Escola de Sistemas de Agentes, seus Ambientes e aplicações, 2014, Porto Alegre. Anais do *Workshop-Escola de Sistemas de Agentes, seus Ambientes e aplicações*, p. 13-24, 2014.

APÊNCIDE A – FORMULÁRIO DE AVALIAÇÃO DE CONFLITOS NORMATIVOS PELO ESPECIALISTA

Instituto de Computação

Avaliação de Conflitos Normativos pelo Especialista

INSTRUÇÕES

1. Objetivo

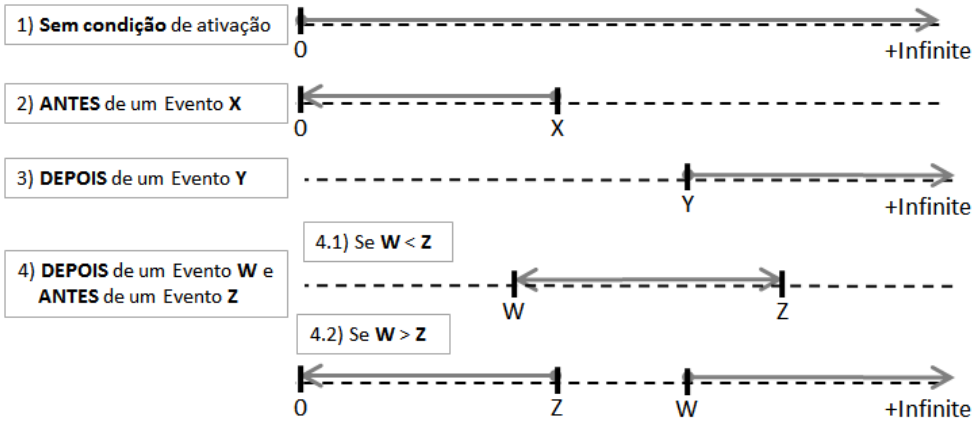
Validar a abordagem de detecção de *potenciais conflitos normativos* (PCN) por especialistas em normas em sistemas multi-agentes (SMA).

2. Instruções

O especialista em normas deve fazer a detecção de PCN em tempo de designer. PCN são conflitos que podem acontecer entre as normas durante a execução dos SMA, dependendo da ordem de execução dos eventos nos SMA.

Duas normas estão em conflito quando (i) há alguma *interseção entre seus períodos de ativação*, (ii) os conceitos deônticos das normas são contraditórios (isto é, obrigação *versus* proibição *ou permissão versus* proibição), (iii) elas estão regulando a uma mesma entidade e (iv) um mesmo comportamento, e (v) estão definidas em um mesmo contexto.

Os *períodos de ativação das normas* podem ser um dos seguintes tipos ilustrados abaixo.



Os eventos mencionados na definição das normas estarão *sublinhados*.

Serão considerados 5 tipos de eventos que ocorrem em tempo de execução: (i) execução de uma ação por um agente, (ii) ativação ou (iii) desativação de uma norma, (iv) cumprimento ou (v) violação de uma norma e um (vi) fato no sistema que tenha se tornado verdade para um agente.

A detecção deve ser feita *entre pares de normas*.

O especialista deve informar os pares de normas em conflito e, para cada par, devem ser informados *todos* os cenários de execução em conflito encontrados, ao se variar a ordem de execução dos eventos mencionados nas normas.

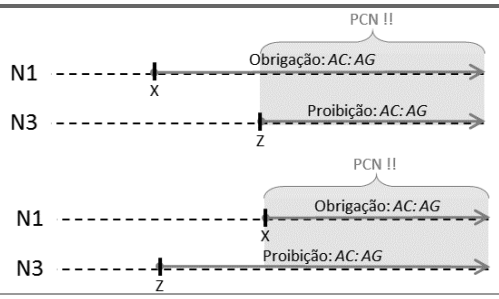
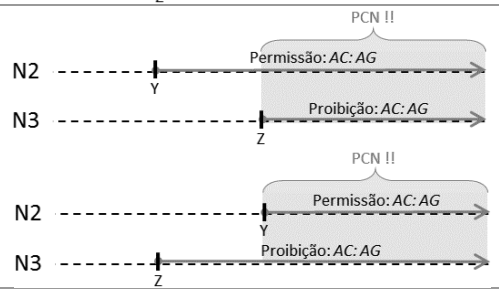
Não é necessário fazer a representação gráfica dos cenários de execução em conflito.

3. Exemplo

Sejam *N1*, *N2* e *N3* três normas definidas pelo designer do sistema, como segue:

- *N1 obriga* agente *AG* a executar ação *AC*, *depois* de evento X;
- *N2 autoriza* agente *AG* a executar ação *AC*, *depois* de evento Y;
- *N3 proíbe* agente *AG* a executar ação *AC, depois* de evento Z.

Todas as três normas estão definidas em um mesmo contexto *C*.

Par de Normas em PCN	Cenários de Execução do PCN
N1 e N3	1) Evento X <= Evento Z 2) Evento Z <= Evento X 
N2 e N3	1) Evento Y <= Evento Z 2) Evento Z <= Evento Y 

DADOS DO ESPECIALISTA

Nome:

Instituição:

Grau de Experiência com Normas em Sistemas Multi-agentes:

○ 1 ano ○ 2 anos ○ 3 anos ○ 4 anos ○ 5 anos

○ Mais de 5 anos:

ATIVIDADE – IDENTIFICAÇÃO DE POTENCIAIS CONFLITOS NORMATIVOS

Sejam *N1*, *N2*, *N3*, *N4*, *N5*, *N6*, *N7*, *N8* e *N9* um conjunto de nove normas definidas para três agentes (*Riley*, *Sofie* e *Dora*) pelo designer do sistema, como segue:

- *N1 obriga Riley* a executar ação *doHomework*;
- *N2 proíbe Riley* de executar ação *playGame antes* dele executar *takeAShower*;
- *N3 autoriza Riley* a executar *playGame depois* do cumprimento de *N1*;
- *N4 proíbe Riley* de executar *playGame depois* que *Dora* executar *serveLunch* e *antes* da situação *doneLunching* torna-se verdade para *Riley*;
- *N5 obriga Riley* a executar ação *cleanRoom antes* de *Dora* executar *serveLunch*;
- *N6 proíbe Riley* de executar ação *playGame se* ele violar norma *N5*;
- *N7 obriga Sofie* a executar *takeAShower depois* que *Riley* executar *takeAShower*;
- *N8 autoriza Sofie* a executar *playGame depois* dela cumprir com a norma *N7*;
- *N9 proibe Sofie* de executar *playGame depois* que Dora executar *serveLunch*.

Todas as normas estão definidas em um mesmo contexto *C*.

Identificar abaixo os pares de normas em conflito e, para cada par, devem ser informados *todos* os cenários de execução em conflito encontrados.

Na norma *N5*, interpretar o "*se*" como "*depois*".

Par de Normas em PCN	Cenários de Execução do PCN

OBSERVAÇÕES SOBRE A ATIVIDADE:

APÊNCIDE B – ONTOLOGIAS COMPLETAS

Este apêndice apresenta as ontologias OWL DL de Normas e de Cenário de Execução completas, descritas neste trabalho, exportadas a partir da ferramenta de edição de ontologias, chamada Protégé (KNUBLAUCH *et al.*, 2004) versão 5.0.0, usando o formato *Turtle* (BECKETT *et al.*, 2014). A linguagem *Turtle* é um subconjunto da linguagem *Notation3* (*N3*) (BERNERS-LEE e CONNOLLY, 2011) e é uma notação alternativa da sintaxe XML do RDF. Optou-se pela sintaxe *Turtle*, pois ela é uma linguagem mais compacta e legível que a sintaxe XML do RDF. *Turtle* tornou-se recomendação pela W3C em fevereiro de 2014. A Seção B.1 mostra a ontologia de Norma e a Seção B.2 apresenta a ontologia de Cenário de Execução.

B.1 ONTOLOGIA OWL DE NORMA

```
@prefix : <http://www.example.com/ontologies/norm.owl#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix exec: <http://www.example.com/ontologies/execution.owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://www.example.com/ontologies/norm.owl> .

<http://www.example.com/ontologies/norm.owl> rdf:type owl:Ontology ;
                                rdfs:comment "The norm ontology"^^xsd:string ;
                                owl:imports <http://www.example.com/ontologies/execution.owl> .


#################################################################
#
#    Object Properties
#
#################################################################

###  http://www.example.com/ontologies/norm.owl#actsIn
:actsIn rdf:type owl:ObjectProperty ;
        rdfs:range :Context ;
        rdfs:domain :Entity .

###  http://www.example.com/ontologies/norm.owl#hasAction
:hasAction rdf:type owl:ObjectProperty ;
           rdfs:range :Action ;
           rdfs:domain :Norm .
```

```
### http://www.example.com/ontologies/norm.owl#hasActivationStatus
:hasActivationStatus rdf:type owl:ObjectProperty ;
                     rdfs:range :ActivationStatus ;
                     rdfs:domain :Norm .

### http://www.example.com/ontologies/norm.owl#hasAfter
:hasAfter rdf:type owl:ObjectProperty ;
          rdfs:range :Condition ;
          rdfs:domain :Norm .

### http://www.example.com/ontologies/norm.owl#hasBefore
:hasBefore rdf:type owl:ObjectProperty ;
           rdfs:range :Condition ;
           rdfs:domain :Norm .

### http://www.example.com/ontologies/norm.owl#hasConflict
:hasConflict rdf:type owl:ObjectProperty ,
                      owl:SymmetricProperty ;
             rdfs:range :Norm ;
             rdfs:domain :Norm .

### http://www.example.com/ontologies/norm.owl#hasContext
:hasContext rdf:type owl:ObjectProperty ;
            rdfs:range :Context ;
            rdfs:domain :Norm .

### http://www.example.com/ontologies/norm.owl#hasDeonticConcept
:hasDeonticConcept rdf:type owl:ObjectProperty ;
                   rdfs:range :DeonticConcept ;
                   rdfs:domain :Norm .

### http://www.example.com/ontologies/norm.owl#hasEntity
:hasEntity rdf:type owl:ObjectProperty ;
           rdfs:range :Entity ;
           rdfs:domain :Norm .

### http://www.example.com/ontologies/norm.owl#hasFulfillmentStatus
:hasFulfillmentStatus rdf:type owl:ObjectProperty ;
                      rdfs:range :FulfillmentStatus .

### http://www.example.com/ontologies/norm.owl#hasRelatedAction
:hasRelatedAction rdf:type owl:ObjectProperty ;
                  rdfs:range :Action ;
                  rdfs:domain :ExecutionOfAction .

### http://www.example.com/ontologies/norm.owl#hasRelatedEntity
:hasRelatedEntity rdf:type owl:ObjectProperty ;
```

```
            rdfs:range :Entity .

###  http://www.example.com/ontologies/norm.owl#hasRelatedNorm
:hasRelatedNorm rdf:type owl:ObjectProperty ;
            rdfs:range :Norm .

###  http://www.example.com/ontologies/norm.owl#hasRelatedSituation
:hasRelatedSituation rdf:type owl:ObjectProperty .

###  http://www.example.com/ontologies/norm.owl#isPerformedBy
:isPerformedBy rdf:type owl:ObjectProperty ;
            rdfs:domain :Action ;
            rdfs:range :Agent ;
            owl:inverseOf :performsAction .

###  http://www.example.com/ontologies/norm.owl#participatesIn
:participatesIn rdf:type owl:ObjectProperty ;
            rdfs:domain :Agent ;
            rdfs:range :Situation .

###  http://www.example.com/ontologies/norm.owl#performsAction
:performsAction rdf:type owl:ObjectProperty ;
            rdfs:range :Action ;
            rdfs:domain :Agent .


#################################################################
#
#    Data properties
#
#################################################################


###  http://www.example.com/ontologies/norm.owl#actionName
:actionName rdf:type owl:DatatypeProperty ;
          rdfs:range xsd:string .

###  http://www.example.com/ontologies/norm.owl#condName
:condName rdf:type owl:DatatypeProperty ;
         rdfs:range xsd:string .

###  http://www.example.com/ontologies/norm.owl#conflictJustification
:conflictJustification rdf:type owl:DatatypeProperty ;
                  rdfs:range xsd:string .

###  http://www.example.com/ontologies/norm.owl#contextName
:contextName rdf:type owl:DatatypeProperty ;
```

```
        rdfs:range xsd:string .

###  http://www.example.com/ontologies/norm.owl#entityName
:entityName rdf:type owl:DatatypeProperty ;
        rdfs:range xsd:string .

###  http://www.example.com/ontologies/norm.owl#normName
:normName rdf:type owl:DatatypeProperty ;
       rdfs:range xsd:string .

###  http://www.example.com/ontologies/norm.owl#situationName
:situationName rdf:type owl:DatatypeProperty ;
          rdfs:range xsd:string .


#################################################################
#
#    Classes
#
#################################################################

###  http://www.example.com/ontologies/norm.owl#Action
:Action rdf:type owl:Class ;
      rdfs:subClassOf [ rdf:type owl:Restriction ;
                        owl:onProperty :actionName ;
                        owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                        owl:onDataRange xsd:string
                      ] ,
                      [ rdf:type owl:Restriction ;
                        owl:onProperty :isPerformedBy ;
                        owl:allValuesFrom :Agent
                      ] .

###  http://www.example.com/ontologies/norm.owl#ActivationOfNorm
:ActivationOfNorm rdf:type owl:Class ;
             rdfs:subClassOf :Condition ,
                             [ rdf:type owl:Restriction ;
                               owl:onProperty :hasRelatedNorm ;
                               owl:onClass :Norm ;
                               owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                             ] .

###  http://www.example.com/ontologies/norm.owl#ActivationStatus
:ActivationStatus rdf:type owl:Class ;
             owl:equivalentClass [ rdf:type owl:Class ;
                                   owl:oneOf ( :Activated
                                               :None
```

```
                                    :Deactivated
                                   )
                        ] .

###  http://www.example.com/ontologies/norm.owl#Agent
:Agent rdf:type owl:Class ;
      rdfs:subClassOf :Entity ,
                  [ rdf:type owl:Restriction ;
                    owl:onProperty :performsAction ;
                    owl:allValuesFrom :Action
                  ] ,
                  [ rdf:type owl:Restriction ;
                    owl:onProperty :participatesIn ;
                    owl:allValuesFrom :Situation
                  ] ,
                  [ rdf:type owl:Restriction ;
                    owl:onProperty :hasRole ;
                    owl:allValuesFrom :Role
                  ] .

###  http://www.example.com/ontologies/norm.owl#Condition
:Condition rdf:type owl:Class ;
         rdfs:subClassOf [ rdf:type owl:Restriction ;
                      owl:onProperty :condName ;
                      owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                      owl:onDataRange xsd:string
                    ] .

###  http://www.example.com/ontologies/norm.owl#Context
:Context rdf:type owl:Class ;
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                    owl:onProperty :contextName ;
                    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                    owl:onDataRange xsd:string
                  ] .

###  http://www.example.com/ontologies/norm.owl#DeactivationOfNorm
:DeactivationOfNorm rdf:type owl:Class ;
                rdfs:subClassOf :Condition ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasRelatedNorm ;
                              owl:onClass :Norm ;
                              owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                            ] .

###  http://www.example.com/ontologies/norm.owl#DeonticConcept
:DeonticConcept rdf:type owl:Class ;
```

```
        owl:equivalentClass [ rdf:type owl:Class ;
                              owl:oneOf ( :Obligation
                                          :Permission
                                          :Prohibition
                                        )
                            ] .

###  http://www.example.com/ontologies/norm.owl#Entity
:Entity rdf:type owl:Class ;
      rdfs:subClassOf [ rdf:type owl:Restriction ;
                        owl:onProperty :entityName ;
                        owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                        owl:onDataRange xsd:string
                      ] ,
                      [ rdf:type owl:Restriction ;
                        owl:onProperty :actsIn ;
                        owl:allValuesFrom :Context
                      ] .

###  http://www.example.com/ontologies/norm.owl#Environment
:Environment rdf:type owl:Class ;
         rdfs:subClassOf :Context .

###  http://www.example.com/ontologies/norm.owl#ExecutionOfAction
:ExecutionOfAction rdf:type owl:Class ;
             rdfs:subClassOf :Condition ,
                             [ rdf:type owl:Restriction ;
                               owl:onProperty :hasRelatedAction ;
                               owl:onClass :Action ;
                               owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                             ] ,
                             [ rdf:type owl:Restriction ;
                               owl:onProperty :hasRelatedEntity ;
                               owl:onClass :Entity ;
                               owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                             ] .

###  http://www.example.com/ontologies/norm.owl#FulfilledObligationNorm
:FulfilledObligationNorm rdf:type owl:Class ;
                 owl:equivalentClass [ rdf:type owl:Class ;
                                       owl:intersectionOf ( :Norm
                                                            [ rdf:type owl:Restriction ;
                                                              owl:onProperty :hasDeonticConcept ;
                                                              owl:hasValue :Obligation
                                                            ]
                                                            [ rdf:type owl:Restriction ;
                                                              owl:onProperty :hasFulfillmentStatus ;
```

```
                                        owl:hasValue :Fulfilled
                                      ]
                                    )
                            ] ;
                rdfs:subClassOf :Norm .

###  http://www.example.com/ontologies/norm.owl#FulfilledProhibitionNorm
:FulfilledProhibitionNorm rdf:type owl:Class ;
                owl:equivalentClass [ rdf:type owl:Class ;
                                      owl:intersectionOf ( :Norm
                                                           [ rdf:type owl:Class ;
                                                             owl:complementOf :ViolatedProhibitionNorm
                                                           ]
                                                           [ rdf:type owl:Restriction ;
                                                             owl:onProperty :hasActivationStatus ;
                                                             owl:hasValue :Deactivated
                                                           ]
                                                           [ rdf:type owl:Restriction ;
                                                             owl:onProperty :hasDeonticConcept ;
                                                             owl:hasValue :Prohibition
                                                           ]
                                                         )
                                    ] ;
                rdfs:subClassOf :Norm .

###  http://www.example.com/ontologies/norm.owl#FulfillmentOfNorm
:FulfillmentOfNorm rdf:type owl:Class ;
            rdfs:subClassOf :Condition ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasRelatedNorm ;
                              owl:onClass :Norm ;
                              owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                            ] .

###  http://www.example.com/ontologies/norm.owl#FulfillmentStatus
:FulfillmentStatus rdf:type owl:Class ;
            owl:equivalentClass [ rdf:type owl:Class ;
                                  owl:oneOf ( :Violated
                                              :Fulfilled
                                              :Unknown
                                            )
                                ] .

###  http://www.example.com/ontologies/norm.owl#Norm
:Norm rdf:type owl:Class ;
    rdfs:subClassOf [ rdf:type owl:Restriction ;
                      owl:onProperty :hasConflict ;
```

```
  owl:allValuesFrom :Norm
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :hasDeonticConcept ;
  owl:onClass :DeonticConcept ;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :hasFulfillmentStatus ;
  owl:onClass :FulfillmentStatus ;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :hasActivationStatus ;
  owl:onClass :ActivationStatus ;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :conflictJustification ;
  owl:allValuesFrom xsd:string
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :hasBefore ;
  owl:onClass :Condition ;
  owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :hasAction ;
  owl:onClass :Action ;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :hasContext ;
  owl:allValuesFrom :Context
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :hasEntity ;
  owl:onClass :Entity ;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :hasAfter ;
  owl:onClass :Condition ;
  owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger
] ,
[ rdf:type owl:Restriction ;
  owl:onProperty :normName ;
```

```
                owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                owl:onDataRange xsd:string
               ] .

###  http://www.example.com/ontologies/norm.owl#Organization
:Organization rdf:type owl:Class ;
          rdfs:subClassOf :Context .

###  http://www.example.com/ontologies/norm.owl#ParticipationInSituation
:ParticipationInSituation rdf:type owl:Class ;
                  rdfs:subClassOf :Condition ,
                                  [ rdf:type owl:Restriction ;
                                    owl:onProperty :hasRelatedSituation ;
                                    owl:onClass :Situation ;
                                    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                                  ] .

###  http://www.example.com/ontologies/norm.owl#Role
:Role rdf:type owl:Class ;
    rdfs:subClassOf :Entity .

###  http://www.example.com/ontologies/norm.owl#Situation
:Situation rdf:type owl:Class ;
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                    owl:onProperty :situationName ;
                    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                    owl:onDataRange xsd:string
                  ] .

###  http://www.example.com/ontologies/norm.owl#ViolatedObligationNorm
:ViolatedObligationNorm rdf:type owl:Class ;
                owl:equivalentClass [ rdf:type owl:Class ;
                                      owl:intersectionOf ( :Norm
                                                           [ rdf:type owl:Class ;
                                                             owl:complementOf :FulfilledObligationNorm
                                                           ]
                                                           [ rdf:type owl:Restriction ;
                                                             owl:onProperty :hasActivationStatus ;
                                                             owl:hasValue :Deactivated
                                                           ]
                                                           [ rdf:type owl:Restriction ;
                                                             owl:onProperty :hasDeonticConcept ;
                                                             owl:hasValue :Obligation
                                                           ]
                                                         )
                                    ] ;
                rdfs:subClassOf :Norm .
```

```
###  http://www.example.com/ontologies/norm.owl#ViolatedPermissionNorm
:ViolatedPermissionNorm rdf:type owl:Class ;
                owl:equivalentClass [ rdf:type owl:Class ;
                                      owl:intersectionOf ( :Norm
                                                           [ rdf:type owl:Restriction ;
                                                             owl:onProperty :hasDeonticConcept ;
                                                             owl:hasValue :Permission
                                                           ]
                                                           [ rdf:type owl:Restriction ;
                                                             owl:onProperty :hasFulfillmentStatus ;
                                                             owl:hasValue :Violated
                                                           ]
                                                         )
                                    ] ;
                rdfs:subClassOf :Norm .

###  http://www.example.com/ontologies/norm.owl#ViolatedProhibitionNorm
:ViolatedProhibitionNorm rdf:type owl:Class ;
                 owl:equivalentClass [ rdf:type owl:Class ;
                                       owl:intersectionOf ( :Norm
                                                            [ rdf:type owl:Restriction ;
                                                              owl:onProperty :hasDeonticConcept ;
                                                              owl:hasValue :Prohibition
                                                            ]
                                                            [ rdf:type owl:Restriction ;
                                                              owl:onProperty :hasFulfillmentStatus ;
                                                              owl:hasValue :Violated
                                                            ]
                                                          )
                                     ] ;
                 rdfs:subClassOf :Norm .

###  http://www.example.com/ontologies/norm.owl#ViolationOfNorm
:ViolationOfNorm rdf:type owl:Class ;
            rdfs:subClassOf :Condition ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasRelatedNorm ;
                              owl:onClass :Norm ;
                              owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasRelatedEntity ;
                              owl:onClass :Entity ;
                              owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                            ] .
```

```
#################################################################
#
#    Individuals
#
#################################################################


###  http://www.example.com/ontologies/norm.owl#Activated
:Activated rdf:type :ActivationStatus ,
                owl:NamedIndividual .

###  http://www.example.com/ontologies/norm.owl#Deactivated
:Deactivated rdf:type :ActivationStatus ,
                 owl:NamedIndividual .

###  http://www.example.com/ontologies/norm.owl#Fulfilled
:Fulfilled rdf:type :FulfillmentStatus ,
                owl:NamedIndividual .

###  http://www.example.com/ontologies/norm.owl#None
:None rdf:type :ActivationStatus ,
              owl:NamedIndividual .

###  http://www.example.com/ontologies/norm.owl#Obligation
:Obligation rdf:type :DeonticConcept ,
                owl:NamedIndividual .

###  http://www.example.com/ontologies/norm.owl#Permission
:Permission rdf:type :DeonticConcept ,
                owl:NamedIndividual .

###  http://www.example.com/ontologies/norm.owl#Prohibition
:Prohibition rdf:type :DeonticConcept ,
                 owl:NamedIndividual .

###  http://www.example.com/ontologies/norm.owl#Unknown
:Unknown rdf:type :FulfillmentStatus ,
               owl:NamedIndividual .

###  http://www.example.com/ontologies/norm.owl#Violated
:Violated rdf:type :FulfillmentStatus ,
                owl:NamedIndividual .


#################################################################
#
```

```
#    General axioms
#
#################################################################


[ rdf:type owl:AllDifferent ;
  owl:distinctMembers ( :Fulfilled
                        :Unknown
                        :Violated
                      )
] .
[ rdf:type owl:AllDifferent ;
  owl:distinctMembers ( :Obligation
                        :Permission
                        :Prohibition
                      )
] .
[ rdf:type owl:AllDifferent ;
  owl:distinctMembers ( :Activated
                        :Deactivated
                       :None
                      )
] .

###  Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net
```

B.2 ONTOLOGIAS OWL DE CENÁRIO DE EXECUÇÃO

```
@prefix : <http://www.example.com/ontologies/execution.owl#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix norm: <http://www.example.com/ontologies/norm.owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://www.example.com/ontologies/execution.owl> .

<http://www.example.com/ontologies/execution.owl> rdf:type owl:Ontology .


#################################################################
#
#    Object Properties
#
#################################################################
```

```
### http://www.example.com/ontologies/execution.owl#actionTime
:actionTime rdf:type owl:ObjectProperty ;
         rdfs:range :Time ;
         rdfs:domain norm:Action .

### http://www.example.com/ontologies/execution.owl#agentTime
:agentTime rdf:type owl:ObjectProperty ;
        rdfs:range :Time ;
        rdfs:domain norm:Agent .

### http://www.example.com/ontologies/execution.owl#hasActvPrd
:hasActvPrd rdf:type owl:ObjectProperty ;
         rdfs:range :ActivationPeriod ;
         rdfs:domain norm:Norm .

### http://www.example.com/ontologies/execution.owl#situationTime
:situationTime rdf:type owl:ObjectProperty ;
           rdfs:range :Time ;
           rdfs:domain norm:Situation .

### http://www.example.com/ontologies/execution.owl#timeAction
:timeAction rdf:type owl:ObjectProperty ;
         rdfs:domain :Time ;
         rdfs:range norm:Action .

### http://www.example.com/ontologies/execution.owl#timeAgent
:timeAgent rdf:type owl:ObjectProperty ;
        rdfs:domain :Time ;
        rdfs:range norm:Agent .

### http://www.example.com/ontologies/execution.owl#timeSituation
:timeSituation rdf:type owl:ObjectProperty ;
           rdfs:domain :Time ;
           rdfs:range norm:Situation .

### http://www.example.com/ontologies/execution.owl#hasPosition
:hasPosition rdf:type owl:ObjectProperty ;
          rdfs:domain :Time ;
          rdfs:range :PositionInInterval .


#################################################################
#
#    Data properties
#
#################################################################
```

```
###  http://www.example.com/ontologies/execution.owl#actPrdName
:actPrdName rdf:type owl:DatatypeProperty ;
         rdfs:range xsd:string .

###  http://www.example.com/ontologies/execution.owl#hasConditionTime
:hasConditionTime rdf:type owl:DatatypeProperty ;
              rdfs:domain norm:Condition ;
              rdfs:range xsd:int .

###  http://www.example.com/ontologies/execution.owl#hasEnd
:hasEnd rdf:type owl:DatatypeProperty ;
       rdfs:range xsd:int .

###  http://www.example.com/ontologies/execution.owl#hasStart
:hasStart rdf:type owl:DatatypeProperty ;
        rdfs:range xsd:int .

###  http://www.example.com/ontologies/execution.owl#hasTime
:hasTime rdf:type owl:DatatypeProperty ;
        rdfs:domain :Time ;
        rdfs:range xsd:int .

###  http://www.example.com/ontologies/execution.owl#timeName
:timeName rdf:type owl:DatatypeProperty ;
        rdfs:range xsd:string .


#################################################################
#
#    Classes
#
#################################################################


###  http://www.example.com/ontologies/execution.owl#ActivationPeriod
:ActivationPeriod rdf:type owl:Class ;
              rdfs:subClassOf [ rdf:type owl:Restriction ;
                                owl:onProperty :hasStart ;
                                owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                                owl:onDataRange xsd:int
                              ] ,
                              [ rdf:type owl:Restriction ;
                                owl:onProperty :actPrdName ;
                                owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                                owl:onDataRange xsd:string
                              ] ,
```

```
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasEnd ;
                      owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                      owl:onDataRange xsd:int
                    ] .

###  http://www.example.com/ontologies/execution.owl#Time
:Time rdf:type owl:Class ;
    rdfs:subClassOf [ rdf:type owl:Restriction ;
                      owl:onProperty :timeSituation ;
                      owl:allValuesFrom norm:Situation
                    ] ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :timeAgent ;
                      owl:allValuesFrom norm:Agent
                    ] ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasTime ;
                      owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                      owl:onDataRange xsd:int
                    ] ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :timeAction ;
                      owl:allValuesFrom norm:Action
                    ] ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :hasPosition ;
                      owl:onClass :PositionInInterval ;
                      owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                    ] .

###  http://www.example.com/ontologies/norm.owl#Action
norm:Action rdf:type owl:Class ;
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                          owl:onProperty :actionTime ;
                          owl:allValuesFrom :Time
                        ] .

###  http://www.example.com/ontologies/norm.owl#Agent
norm:Agent rdf:type owl:Class ;
       rdfs:subClassOf [ rdf:type owl:Restriction ;
                         owl:onProperty :agentTime ;
                         owl:allValuesFrom :Time
                       ] .

###  http://www.example.com/ontologies/norm.owl#Condition
norm:Condition rdf:type owl:Class ;
```

```
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                          owl:onProperty :hasConditionTime ;
                          owl:allValuesFrom xsd:int
                        ] .

###  http://www.example.com/ontologies/norm.owl#Norm
norm:Norm rdf:type owl:Class ;
       rdfs:subClassOf [ rdf:type owl:Restriction ;
                         owl:onProperty :hasActvPrd ;
                         owl:allValuesFrom :ActivationPeriod
                       ] .

###  http://www.example.com/ontologies/norm.owl#Situation
norm:Situation rdf:type owl:Class ;
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                          owl:onProperty :situationTime ;
                          owl:allValuesFrom :Time
                        ] .

###  http://www.example.com/ontologies/execution.owl#PositionInInterval
:PositionInInterval rdf:type owl:Class ;
            owl:equivalentClass [ rdf:type owl:Class ;
                                  owl:oneOf ( :InsidePosition
                                              :OutsidePosition
                                              :UnknownPosition
                                            )
                                ] .

###  http://www.example.com/ontologies/execution.owl#TimeInsideInterval
:TimeInsideInterval rdf:type owl:Class ;
            owl:equivalentClass [ rdf:type owl:Class ;
                                  owl:intersectionOf ( :Time
                                                       [ rdf:type owl:Restriction ;
                                                         owl:onProperty :hasPosition ;
                                                         owl:hasValue :InsidePosition
                                                       ]
                                                     )
                                ] ;
            rdfs:subClassOf :Time .

###  http://www.example.com/ontologies/execution.owl#TimeOutsideInterval
:TimeOutsideInterval rdf:type owl:Class ;
             owl:equivalentClass [ rdf:type owl:Class ;
                                   owl:intersectionOf ( :Time
                                                        [ rdf:type owl:Class ;
                                                          owl:complementOf :TimeInsideInterval
                                                        ]
```

```
                                        )
                                ] ;
                rdfs:subClassOf :Time .


#################################################################
#
#    Individuals
#
#################################################################


###  http://www.example.com/ontologies/execution.owl#OutsidePosition
:OutsidePosition rdf:type :PositionInInterval ,
                          owl:NamedIndividual .

###  http://www.example.com/ontologies/execution.owl#InsidePosition
:InsidePosition rdf:type :PositionInInterval ,
                         owl:NamedIndividual .

###  http://www.example.com/ontologies/execution.owl#UnknownPosition
:UnknownPosition rdf:type :PositionInInterval ,
                          owl:NamedIndividual .


#################################################################
#
#    General axioms
#
#################################################################


[ rdf:type owl:AllDifferent ;
  owl:distinctMembers ( :InsidePosition
                        :OutsidePosition
                        :UnknownPosition
                       )
] .

###  Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net
```