INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO TRIÂNGULO MINEIRO
PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO
htttp://editora.iftm.edu.br/index.php/inova

ISSN 2447-4924 (Impressa)
ISSN 2447-598X (Digital)

CIÊNCIA DA COMPUTAÇÃO

UM COMPILADOR PARA DEFINIÇÃO E GERAÇÃO DE NORMAS EM SISTEMAS MULTIAGENTES

Luccas Felippe Oliveira[1], Eduardo Augusto Silvestre[2] e Viviane Torres da Silva[3]

RESUMO: Um agente de software é uma entidade capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores. Os agentes inteligentes podem ser classificados de acordo com a maneira que eles coletam informações e agem no ambiente. No caso de vários agentes cooperando ou disputando entre si, inseridos em um mesmo ambiente e trocando informações, chamamos esse sistema de multiagente (SMA). SMAs são sociedades autônomas, heterogêneas e podem trabalhar a fim de alcançar objetivos comuns ou diferentes. A fim de lidar com a heterogeneidade, autonomia e diversidade de interesses entre os agentes da sociedade, projetistas desses sistemas estabelecem um conjunto de normas que é usado como um mecanismo de controle social que visa possibilitar que os agentes possam trabalhar em conjunto. Dentro deste contexto, dois problemas frequentes são a representação de normas e também sua verificação e resolução de possíveis conflitos. Existem várias formas de representação de normas na literatura, mas ainda falta um consenso a respeito da sua expressividade e formalização. Os conflitos são inerentes a um sistema normativo, no qual diferentes agentes cooperam ou concorrem por um mesmo recurso. Este trabalho apresenta uma nova sintaxe para normas em SMA. Foi definida uma linguagem formal utilizando uma gramática BNF e a linguagem foi aplicada com o uso de um compilador desenvolvido neste trabalho. A linguagem pode ser utilizada como base de uma arquitetura em um SMA normativo. Como trabalhos futuros, a linguagem será utilizada para verificação de conflitos em SMA.

Palavras-chave: Gramática BNF, Linguagens Formais, Compiladores, Sistemas Multiagentes.

A COMPILER FOR DEFINING AND GENERATING NORMS IN MULTI-AGENT SYSTEMS

ABSTRACT: A software agent is an entity capable of perceiving its environment through sensors and acting upon that environment through actuators. Intelligent agents can be classified according to the way they collect information and act on the environment. In the case of multiple agents cooperating or competing with each other, inserted into the same environment and sharing information, we call this multi-agent system (MAS). MAS are autonomous and heterogeneous societies and can work to achieve common or different goals. In order to deal with the autonomy and diversity of interests among different members of society, designers of these systems provide a set of norms that is used as a social control mechanism which enables agents to work together. Within this context, two frequent problems are the representation of norms and also their verification and resolution of potential conflicts. There are several forms of representation of norms in literature, but still there is lack of consensus about its expressiveness and formalization. The conflicts are inherent in a normative system where different agents cooperate or compete for the same resource. This paper presents a new syntax for norms in MAS. A formal language was defined using a BNF grammar and the language was applied with the use of a compiler developed in this work. The language can be used as the basis of architecture in a normative MAS. As future work, the language will be used to check conflicts in MAS

Keywords: BNF Grammar. Formal Languages. Compilers. Multi-agent Systems.

[1]Graduando em Engenharia da Computação. Instituto Federal do Triângulo Mineiro, *Campus* Avançado Uberaba Parque Tecnológico, Uberaba, MG, Brasil. lfelippeoliveira@gmail.com
[2]Doutor em Computação. Instituto Federal do Triângulo Mineiro, *Campus* Avançado Uberaba Parque Tecnológico, Uberaba, MG, Brasil. eduardosilvestre@iftm.edu.br
[3]Doutora em Computação. IBM Research (licença UFF), Rio de Janeiro, RJ, Brasil. vivianet@br.ibm.com

INTRODUÇÃO

Os sistemas que usam agentes vêm ganhando uma maior importância na pesquisa e na prática para o desenvolvimento de aplicações diversas nos últimos anos. Segundo Russell e Norvig (2009), um agente de software é uma entidade capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores. Os agentes inteligentes podem ser classificados de acordo com a maneira que eles coletam informações e agem no ambiente. Sistemas nos quais vários agentes cooperam ou disputam entre si, inseridos em um mesmo ambiente e trocando informações, são chamados de sistemas multiagentes (SMA) (RUSSELL; NORVIG, 2009). SMA são sociedades autônomas, heterogêneas e podem trabalhar a fim de alcançar objetivos comuns ou diferentes (WOOLDRIDGE, 2009).

A fim de lidar com a autonomia e diversidade de interesses entre os diferentes membros, o comportamento dos agentes é governado por um conjunto de normas especificado para regular suas ações (SILVA, 2008). A introdução de normas em sistemas multiagentes tem sido considerada um fator importante para garantir a eficácia dos agentes (LÓPEZ, 2003).

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

Dado que normas regulam o comportamento de agentes autônomos, esses são, portanto, livres para decidir por cumprir ou violar cada norma. A fim de influenciar o seu cumprimento, recompensas são utilizadas como forma de promover o cumprimento das normas e punições são utilizadas como meio para inibir a violação. Recompensas e punições podem estar associadas ao cumprimento de objetivos (LÓPEZ, 2003), a realização de ações ou ao estabelecimento de outras normas (SILVA, 2008).

É necessário considerar, também, que as normas de um sistema não são isoladas uma das outras. Às vezes, a ativação, desativação, cumprimento ou violação de uma norma pode levar a ativação, desativação, cumprimento ou violação de outras normas (LÓPEZ, 2003). Na literatura, esses relacionamentos são conhecidos como *interlocking* entre normas.

As normas regulam o comportamento dos agentes através da definição de obrigações, permissões e proibições, as quais indicam que eles estão obrigados, permitidos ou proibidos, respectivamente, de realizar um determinado comportamento (SANTOS NETO, 2012).

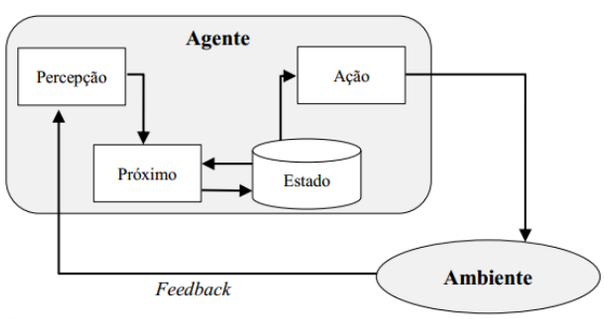
Apesar de existirem muitas pesquisas sobre sistemas multiagentes e diversas notações para representação de uma norma, sistemas multiagentes normativos ainda carecem de uma representação mais padronizada e expressiva para definição de uma norma. Essa carência se reflete em poucas linguagens e arquiteturas utilizarem normas em seu ambiente. Com o objetivo de minimizar problemas de representação e utilização de normas, este trabalho apresenta uma nova definição de norma, baseada em Figueiredo et al. (2011) que identificaram os principais constituintes de uma norma. Dessa forma, aqui a acepção denotação é estendida como mais expressiva e definida de maneira formal utilizando a sintaxe da BNF (*Backus-NaurForm*) (BACKUS et al., 1963). Assim, um compilador foi criado para geração de objetos a partir da gramática BNF.

REFERENCIAL TEÓRICO: Agentes e Normas

Os SMA são sistemas computacionais compostos por agentes de software que visam alcançar seus objetivos através da interação. Antes de descrever os SMA, é importante compreender as características dos agentes de software. Segundo Wooldridge (2009), um agente é um sistema computacional que está situado em algum ambiente e é capaz de executar ações de maneira autônoma. O autor afirma, ainda, que a execução dessas ações é para cumprir com os objetivos delegados ao agente. Em outras palavras, agentes de software são programas autônomos (ou autossuficientes), capazes de executar ações para atingir seus objetivos de acordo com a percepção que possuem sobre o ambiente.

A Figura 1 apresenta modelo de agente apresentado por Wooldridge (2009) sobre um determinado ambiente.

Figura 1. Modelo de agente.



Fonte: Wooldridge, 2009.

De acordo com a Figura 1, o agente recebe um *feedback* do ambiente, o qual é captado através de sensores que notam as mudanças que estão ocorrendo. Através da informação captada pelo agente, que está em um determinado estado de execução, o mesmo decide por executar (ou não) uma ação. Caso execute, os efeitos de sua ação podem alterar o ambiente a qual pode ser sentida pelo agente. Se o agente não executar uma ação a partir da percepção atual, ele pode aguardar por novas percepções.

Mesmo que autonomia seja uma característica predominante em agentes de software, entidades autônomas usualmente necessitam interagir com outros agentes para que objetivos de maior complexidade possam ser alcançados. Com isso, a interação entre agentes de software surge com a intenção de alcançar os objetivos propostos a cada agente, derivando então o conceito de SMA.

A autonomia e a heterogeneidade dos agentes de software exigem que o comportamento dos agentes seja regulado por um sistema normativo. Sistemas normativos são compostos por um conjunto de normas (ou regras) que visam instruir o agente sobre aquilo que ele pode fazer (permissão), sobre aquilo que não pode fazer (proibição) e sobre aquilo que deve ser feito (obrigação). No contexto sociológico, normas são regras que uma sociedade ou um grupo utiliza para definir comportamentos, atitudes, valores e crenças, classificando-as como apropriadas ou inapropriadas (DEUTSCH; GERARD, 1955).

No contexto de SMA, normas são o meio descritivo para governar o comportamento dos agentes em um determinado contexto e período. De acordo com Jiang, et al. (2012), as normas possuem sujeitos que devem cumpri-las. Além disso, os autores também definem normas como sendo um conjunto de "o que fazer" e "o que não fazer". Definem também que normas são fatores dependentes do ambiente institucional no qual são aplicadas.

REFERENCIAL TEÓRICO: Compiladores

Um compilador pode ser definido como um programa de computador que converte um código escrito em uma linguagem de programação, também conhecido como código fonte, em outro código semanticamente equivalente, que faz a mesma coisa e gera o mesmo resultado. O termo "compilado" é usualmente empregado para designar programas que traduzem o código fonte de uma linguagem de programação de alto nível para um código objeto escrito em uma linguagem de programação de baixo nível (ex: linguagem de montagem), específica para o processador que executará o código compilado.

A transformação de um programa escrito em uma linguagem de programação de alto nível em um programa equivalente em linguagem de máquina é realizada pelo compilador. Um compilador também pode realizar outros tipos de transformações como traduzir um código objeto de uma arquitetura para outra, ou mesmo traduzir um código fonte de uma linguagem para outra.

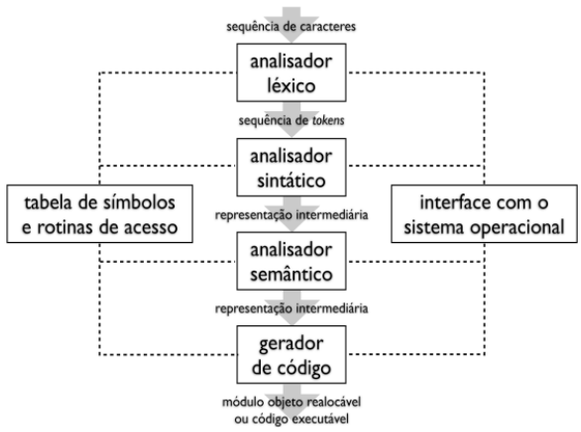
Para que essa transformação seja realizada, o processo de compilação é dividido em etapas que analisam um dado formato e o sintetizam em um novo, tipicamente partindo-se de uma sequência de caracteres do código fonte e gerando o código objeto. As etapas principais da compilação são:

- Análise léxica analisa o texto apresentado e o divide em pequenos pedaços, chamados tokens, os quais podem ser membros válidos da linguagem em que o código foi escrito. Nessa etapa, o compilador verifica a existência de inconsistências, ou seja, quando alguma parte do texto não pode ser convertida em um token válido.
- Análise sintática processa os token se gera uma representação intermediária (em inglês, Intermediate Representationou IR), que pode ser uma representação sintática sequencial ou em árvore abstrata (AST - Abstract Syntax Tree), além de uma tabela de símbolos formada pelos identificadores usados no programa e seus atributos. Nesse processo, o compilador verifica erros de sintaxe, os quais ocorrem quando alguma sequência de tokens não é reconhecida.
- Análise semântica processa a representação intermediária (IR) para verificar se o programa atende à semântica exigida pela linguagem de origem. Por exemplo, a verificação de todos os identificadores usados e suas respectivas declarações com os tipos compatíveis. Durante essa análise, o compilador verifica se o programa atende a todos os requisitos de semântica da linguagem usada.

A Geração de código transforma a representação intermediária em um código de máquina equivalente, o qual pode estar na forma de um módulo realocável ou diretamente em um programa executável.

Além dos componentes que realizam as quatro etapas principais descritas, a tabela de símbolos e de acesso a rotinas e uma interface com o sistema operacional também fazem parte do compilador, conforme demonstrado na Figura 2. A interface é usada para realizar a leitura e gravação de arquivos, fazer a comunicação com o usuário e facilitar a portabilidade de um compilador entre sistemas operacionais.

Figura 2. Estrutura em alto nível de um compilador simples.



Fonte: MUCHNICK, 1997.

TRABALHOS RELACIONADOS

Normas vêm sendo amplamente usadas em SMA para lidar com a heterogeneidade, autonomia e diversidade de interesses entre os diferentes agentes. Normas descrevem o comportamento que pode ser executado, o que deve ser executado e o que não pode ser executado. Ao longo dos anos, diversos trabalhos definiram uma sintaxe para normas. Entretanto, estes trabalhos ainda não conseguiram realizar uma transição entre a teoria e prática. Este fato é comprovado na não existência de arquiteturas para sistemas multiagentes robustas que usem normas. Um dos principais motivos está relacionado a uma notação simples, pouco expressiva e não padronizada.

Na Figura 2, a primeira parte a apresenta os principais componentes comumente utilizados para representar uma norma, a segunda parte apresenta alguns dos seus principais trabalhos.

A lógica deôntica (VON WRIGHT, 1951) é uma lógica modal com operadores para permissão, obrigação e proibição. Os conceitos da lógica deôntica vêm sendo amplamente usados em sistemas jurídicos e normativos (MEYER; WIERINGA, 1994). Na lógica deôntica, a modalidade de uma norma é chamada de conceito deôntico e representa a natureza da regulação definida pela norma: proibição, permissão ou obrigação.

Não existe um consenso na literatura a respeito dos componentes que uma norma deveria ter. Entretanto, alguns componentes são comuns na maioria dos trabalhos. Uma norma: é sempre associada com um conceito deôntico, regula um dado comportamento e é direcionada a uma entidade; regula uma ação atômica, uma ação complexa ou atinge um estado; é sempre associada a uma entidade que pode ser um agente, uma organização ou um papel. Adicionalmente, algumas abordagens associam a norma a um contexto que indica o escopo onde ela é definida. Outras abordagens consideram que normas podem ser associadas com condição de ativação e desativação, em que se tornam ativadas e devem ser cumpridas quando a condição de ativação é disparada e tornam-se desativas ao final desta. Para garantir o cumprimento, algumas normas especificam sanções que podem ser aplicadas aos agentes que a violam. Também podem ser definidas recompensas como forma de incentivo ao cumprimento da norma.

Alguns dos principais trabalhos encontrados na literatura são os trabalhos de Silva; Zahn (2013), Vasconcelos; Kollingbaum Norman; . (2009), Aphale; Sensoy (2013), Cholvy; Cuppens (1995), Ll, et al. (2014) e Vasconcelos, Kollingbaum; Norman (2004). Em Silva; Zahn (2013), uma norma é uma tupla da forma <*Deoc,c,e,a,ac,dc,s*>, onde *Deoc* é o conceito deôntico; *c* é o contexto onde a norma é definida; *e* é a entidade cujo comportamento é regulado; *a* é a ação atômica; *ac* é a condição que ativa a norma (uma data); *dc* é a condição que desativa a norma (uma data); *e* é o estado da norma {cumprida, violada, nenhum}.

O trabalho de Vasconcelos; Kollingbaum; Norman (2009) representa a norma como <v,td,ta,te>, onde *v* é da forma $N\alpha:\rho\circ\varphi\Gamma$; *N* é um conceito deôntico do conjunto {obrigação, proibição, permissão}; α é um agente identificador; ρ é um papel identificador; φ é fórmula atômica da lógica de primeira ordem que representa uma ação com parâmetros; Γ é um conjunto de restrições nas variáveis ocorrendo na fórmula atômica; td é o tempo onde v foi declarada; ta é o tempo quando v torna-se ativo; te é o tempo quando v expira. A associação entre a fórmula da primeira ordem φ e o conjunto de restrições Γ é representado como $\varphi\circ\Gamma$.

Em Aphale, Norman, Sensoy (2013), normas são expressas usando semântica de fórmulas conjuntivas (conjunções de asserções atômicas). Uma fórmula conjuntiva sobre uma ontologia associa variáveis usadas nas asserções (conceitos ou relações de ontologia) com um conjunto de restrições que restringem valores que eles podem assumir. Uma norma é representada como $\alpha \rightarrow N\chi: \rho(\lambda:\varphi)/e$, onde α é uma condição que ativa a norma; N é o conceito deôntico do conjunto {obrigação, permissão, proibição}; χ é a política endereçada; ρ é um papel; φ a ação ou estado regulado; e é a condição que desativa a norma. As condições de ativação e desativação são fórmulas da semântica conjuntiva.

Já em Cholvy; Cuppens (1995), normas são representadas na forma Nrp, onde N é um conceito deôntico do conjunto {obrigação, proibição, permissão}; r é um papel; p é a ação sendo regulada. Os componentes da norma são proposições.

Em Li et al. (2014), uma norma é definida como uma tupla da forma <role,deontic,action,condition,deadline>, onde role é um papel identificador; deontic é um conceito deôntico do conjunto {obrigação, permissão, proibição}; action é a ação sendo regulada; condition é definido como <ΣE>, onde Σ é um conjunto de estados onde a norma é aplicável e E é a sequência de eventos; deadline é um evento que define a condição de desativação da norma. Adicionalmente, obrigações e proibições podem ser associadas com sanções.

Por fim, segundo Vasconcelos, Kollingbaum; Norman (2004), a norma é representada como N(r,a,ac,ec) onde r é um papel; a é a atividade sendo regulada (ação ou estado); ac é a condição que desativa a norma; ec é a condição que desativa a norma. A condição de ativação e desativação são estados e os componentes da norma permitem parametrização.

UMA NOVA SINTAXE PARA NORMAS

A definição de norma utilizada neste trabalho é baseada em Figueiredo; Silva; Braga (2011) que analisaram as principais estratégias encontradas na literatura para definir uma norma. De acordo com os autores, um norma proíbe, permite ou obriga uma entidade a executar uma ação em um contexto durante certo período de tempo. A diferença entre nossa definição de norma e a definição apresentada pelos autores em questão é a representação da ação sendo regulada, para quem a ação é representada por única constante (por exemplo, alcançar, entrar,...). Nossa representação é mais sofisticada e expressiva. Além disso, nós consideramos apenas ações para o comportamento, estadas não são considerados. A partir de agora, será usado apenas o termo ação.

Para nossa definição de norma, considere as seguintes definições para conjuntos o conjunto Nrm é o conjunto de todas as normas, o conjunto C é o conjunto de todos os contextos, o conjunto E é o conjunto de todas as entidades, o conjunto A é conjunto de todas as ações, o conjunto Cd é o conjunto de todas as condições de ativação e desativação, o conjunto O é o conjunto de todas as organizações, o conjunto Env é o conjunto de todos os ambientes, o conjunto Ag é o conjunto de todos os agentes e o conjunto R é o conjunto de todos os papéis.

Uma norma $n \in Nrm$ é uma tupla da seguinte forma:

$$(deoC, c, e, a, ac, dc)$$

Onde C é um conceito deôntico do conjunto $\{O, F, P\}$, respectivamente, obrigado, proibido e permitido; $c \in C$ é o contexto onde a norma é definida; $e \in E$ é a entidade cujo comportando é regulado; $a \in A$ é a ação sendo representada; $ac \in Cd$ indica a condição que ativa a norma e $dc \in Cd$ é a condição que desativa a norma.

Toda norma é definida no escopo de um contexto. A entidade, cuja ação está sendo regulada deve cumprir a norma quando executa no contexto onde é definida. Neste trabalho, nós consideramos que a norma pode ser definida no contexto de uma organização $o \in O$ ou de um ambiente $env \in Env$. O conjunto de todos os possíveis contextos são definidos como $C = O \cup Env$. Uma norma regula a ação de um agente $a \in Ag$, uma organização (ou grupo de agentes) $o \in O$ ou um papel $r \in R$. Agentes, organizações e papéis são entidades do conjunto $E = Ag \cup R \cup O$.

As condições de ativação e desativação, $ac \in Cd$ e $dc \in Cd$, podem ser um evento, uma data, execução de uma ação ou o cumprimento de uma norma, etc. Neste trabalho, o foco será na especificação de dados porque é mais fácil descobrir se um evento aconteceu primeiro que outro evento. Dessa forma, nós usamos símbolos matemáticos simples como $\leq$ e $\geq$ para indicar que um evento ocorreu antes ou depois de outro ($\forall n \in N, ac \leq dc$).

Uma ação é definida pelo nome da ação e, opcionalmente, seus atributos e seus valores, um objeto onde a ação é executada e uma lista de atributos (com seus valores). Dessa forma, este artigo define quatro tipos diferentes de representações para ações:

(i) ação;
(ii) açãoobjeto;
(iii) ação(atributo1 = {valor1}, atributo2 = {valor2}, ...});
(iv) açãoobjeto (atributo1 = {value1}, atributo2 = {valor2},...}).

O projetista de um SMA pode definir qualquer dos quarto tipos diferentes de normas para representar seu domínio, esses quatro tipos diferentes representam uma grande flexibilidade na criação de um SMA. Para exemplificar estes quatro tipos de descrever uma norma, considera as quatro proibições:

(i) Na proíbe agente A de se vestir;
(ii) Nb proíbe o agente A de vestir calça;
(iii) Nc proíbe o agente A de vestir roupas vermelhas;
(iv) Nd proíbe o agente A de vestir calças vermelhas.

Os comportamentos descritos nas normas são representados como:

(i) Na: vestir;
(ii) Nb: vestir calça;
(iii) Nc: vestir (cor={vermelha});
(iv) Nd: vestir calça (cor={vermelha}).

Nós propomos uma linguagem BNF normativa com o propósito de formalmente descrever a norma. A gramática é representada pela sintaxe do GOLD Parser Builder (BUILDER, 2015).

<norm> ::= '<'<deontic_concept>'," <action>'," <context>'," <entity>'," <activation_date>'," <deactivation_date>'>'

<deontic_concept> ::= 'OBLIGED' | 'FORBIDDEN' | 'PERMITTED'

<context> ::= 'ORGANIZATION' | 'ENVIRONMENT'

<entity> ::= 'AGENT' | 'ROLE' | 'ORGANIZATION' | 'ALL'

<action> ::= <action_name> |

<action_name><information_of_the_action>

<information_of_the_action> ::= <object> |

'('<attributes_and_values>')' | <object>

'('<attributes_and_values>')'

<attributes_and_values> ::= <attribute>'={'<values>'}' | <attribute>'={'<values>'},' <attributes_and_values>

<values> ::= <value> | <value>','<values>

<action_name> ::= Identifier

<object> ::= Identifier

<attribute> ::= Identifier

<value> ::= Identifier

<activation_date> ::= <date>

<deactivation_date> ::= <date>

<date> ::= <day><month><year>

<day> ::= Numbers '/'

<month> ::= Numbers '/'

<year> ::= Numbers

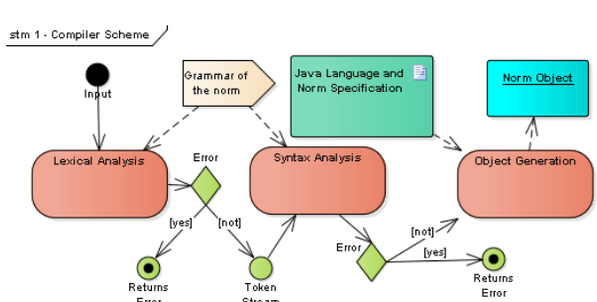
Numbers = {Number}+

Identifier = {Letter}+

COMPILADOR PROPOSTO

Nessa seção, o compilador desenvolvido é apresentado. Com a definição da gramática feita é possível construir um compilador para analisar a entrada do usuário, determinando sua validade e, caso correto, gerar um objeto. A construção do compilador proposto é dividida em análise léxica, a análise sintática e a geração do código (objeto). O compilador foi desenvolvido na linguagem Java. A Figura 3 demonstra o processo de compilação feito pelo compilador implementado.

Figura 3. Esquematização do compilador proposto.



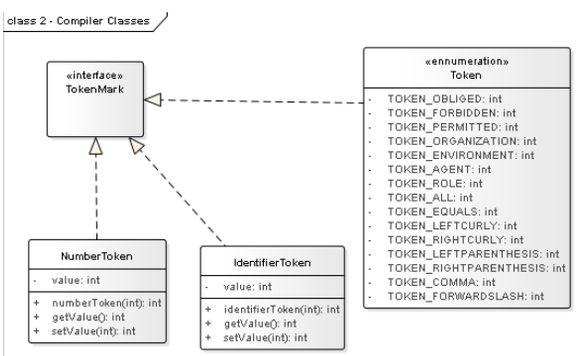
Fonte: elaborado pelos autores.

COMPILADOR PROPOSTO: o Analisador Léxico

Como descrito anteriormente, a análise léxica recebe a entrada do usuário e converte em *tokens,* nessa implementação é descrito em termos de classes os tipos de *tokens* que serão criados, a Figura 4 apresenta em UML a relação das classes criadas para implementar a análise léxica.

As classes *NumberToken*, *IdentifierToken* e a enumeração *Token* implementam a interface *TokenMark*, que existe para indicar a existência de um *token*, isso é feito para que cada classe de *token* tenha em comum algo que as identificam como *token*. A enumeração *Token* representa os *tokens* das palavras chaves existentes na linguagem como os símbolos que ela usa. A classe *NumberToken* representa entrada numérica. A classe *IdentifierToken* representa os identificadores na linguagem.

Figura 4. Descrição de token.



Fonte: elaborada pelos autores

Ao entrar com a descrição da norma, o analisador léxico verifica se não há nenhuma inconsistência na escrita, isto é, algum erro de ortografia; após a análise, são criados os respectivos *tokens*. Os *tokens* são, então, dispostos na *TokenStream* que é uma classe responsável por permitir uma leitura sequencial dos *tokens*. *TokenStream* é passada para o próximo estágio de compilação.

COMPILADOR PROPOSTO: o Analisador Sintático

Ao analisador sintático cabe a responsabilidade de verificar a coesão da entrada, isto é, verificar a sequencialidade lógica dos *tokens* recebidos. Para verificar se a ordem dos *tokens* está correta foi implementado uma máquina de estados.

Uma máquina de estados contém um ponto inicial, seus estados e um ponto final. Para a entrada ser considerada válida, todos os estados devem ser satisfeitos, isto é, não pode ocorrer uma transição de um estado para outro não especificado nessa máquina.

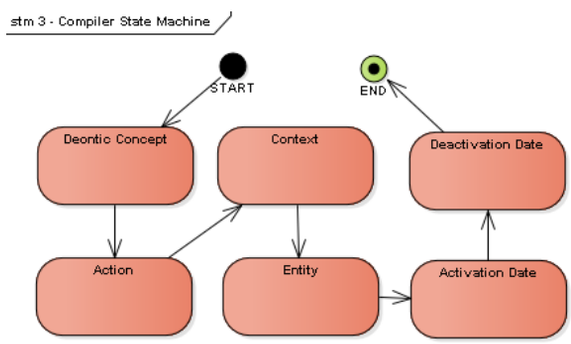
A Figura 5 representa a máquina de estado desenvolvida. Na representação o estado de "START" é a transição do estado "DEONTICCONCEPT" e o estado "END" é a transição para o estado "CONTEXT".

Como o estado de "ACTION" compreende mais estados, ele é apresentado na Figura 6.

Se a entrada conseguir ser unificada com o diagrama de estados, a entrada é válida. Assim, uma representação intermediária da norma pode ser feita. Em nível de implementação, o código intermediário é construído em cima de uma estrutura de tabela hash.

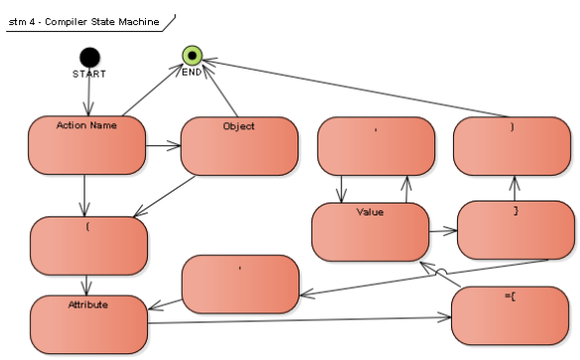
A representação contém os valores dos identificadores e *tokens*. Essa representação é posteriormente passada para a etapa de geração do código.

Figura 5. Máquina de Estados desenvolvida.



Fonte: elaborada pelos autores

Figura 6. Detalhamento do estado "ACTION".



Fonte: elaborada pelos autores

COMPILADOR PROPOSTO: Geração do Objeto

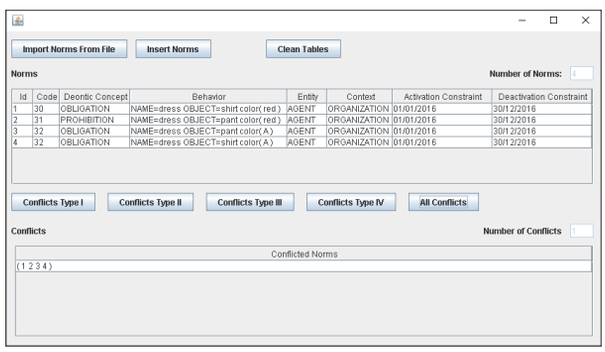
Dada a representação intermediária, realizada pela etapa de análise sintática é possível realizar a geração do código-objeto, nesse caso é realizado o instanciamento de uma classe que representa uma norma. A classe *Norm* contém todos os componentes necessário para definição das normas segundo os quatro tipos previamente definidos.

APLICAÇÃO DO COMPILADOR

Para ilustrar a aplicação dos processos desenvolvidos no compilador, o compilador foi utilizado como parte da ferramenta MuNoCC (*Multiple Norms Conflict Checker Tool*) (SILVESTRE, 2017). Esta é uma ferramenta utilizada para verificação de diversos tipos de conflitos em SMA.

A Figura 7 apresenta uma tela da aplicação MuNoCC onde o compilador desenvolvido foi utilizado. As normas, obedecendo a gramática BNF definida neste artigo, são importadas para linhas em uma tabela. Para realizar este processo de importação, o compilador desenvolvido é utilizado e, no fim, gera objetos Java.

Figura 7. Tela de inserção das normas a partir da gramática.



Fonte: elaborada pelos autores

CONCLUSÕES

Os sistemas multiagentes vêm ganhado maior importância nos últimos anos, tanto nas universidades quanto nas empresas. Apesar da significante pesquisa na área, ainda existem muitos desafios a serem considerados. No caso de normas, a definição e a utilização em arquiteturas normativas de normas é um destes desafios.

Após uma extensa pesquisa na literatura, vários artigos e notações para normas foram encontrados. Entretanto, várias destas notações são pouco expressivas ou apenas aplicáveis em uma arquitetura específica.

A principal contribuição deste trabalho é a criação de uma nova sintaxe para representação de normas em sistemas multiagentes. A notação criada é mais expressiva e flexível do que as notações encontradas em trabalhos anteriores. Além disso, a notação foi formalizada através de uma gramática BNF e sua utilidade foi generalizada através da criação de um compilador e a geração de normas Java.

Como trabalhos futuros, existem duas vertentes principais: a utilização nas normas em uma arquitetura normativa e a verificação e resolução de conflitos. Como existem poucas linguagens e arquiteturas de sistemas multiagentes que incorporam normas, um trabalho futuro desafiador é incluir a notação apresentada neste trabalho em um destes ambientes. O ambiente sugerido é o JaCaMo (BOISSIER, et al., 2013).

Outra área passível de várias pesquisas sequentes ao trabalho é utilizar a sintaxe criada para verificar e resolver conflitos em sistemas normativos. Os conflitos normativos em um SMA são inevitáveis e ocorrem porque um SMA tem diversos agentes concorrendo por um mesmo recurso ou recursos relacionados. Os conflitos acontecem quando duas normas regulando o mesmo comportamento estão ativadas, mas uma delas obriga (ou permite) a realização do comportamento enquanto a outra proíbe a realização do mesmo comportamento (VASCONCELOS; KOLLINGBAUM; NORMAN, 2009). Como trabalho futuro, pretende-se elaborar uma estratégia para verificação e resolução de conflitos em SMA.

REFERÊNCIAS

APHALE, M., NORMAN, T. J., SENSOY, M. Goal-directed policy conflict detection and prioritisation: an empirical. In: **Lecture notes in computer science**. [S.l.]: Springer, 2013. p. 87–104.

BACKUS, J. W. et al. Revised report on the algorithm language ALGOL 60. **ACM digital library**. [S.l.], v. 6, n. 1, 1963. p. 1–17. Disponível em: <http://doi.acm.org/10.1145/366193.366201>. Acesso em: 10 jun. 2016

BOISSIER, O. et al. Multi-agent oriented programming with JaCaMo. In: **Science Computer Programming**, v. 78, n. 6, 2013. p. 747–761. Disponível em: <http://dx.doi.org/10.1016/j.scico.2011.10.004>. Acesso em: 10 jun. 2016

BUILDER, Gold Parser. **Gold Parser Builder**. [S.l: s.n.], 2015.

CHOLVY, L; CUPPENS, F. **Solving normative conflicts by merging roles**: ICAIL '95, New York, NY, USA: ACM digital library, 1995. p.201–209. Disponível em: <http://doi.acm.org/10.1145/222092.222241>. Acesso em: 10 de jun. 2016

DEUTSCH, Morton; GERARD, Harold B. A study of normative and informational social influences upon individual judgement. In: **Journal of Abnormal & Social Psychology**, v. 51, n. 3, 1995. p. 629–636. Disponível em: <http://dx.doi.org.arugula.cc.columbia.edu:2048/10.1037/h0046408>. Acesso em: 10 jun. 2016

FIGUEIREDO, Karen da Silva; SILVA, Viviane Torres da; BRAGA, Christiano de Oliveira. Modeling Norms in Multi-agent Systems with NormML. In: **COIN@AAMAS'10**, Berlin, Heidelberg: Springer-Verlag, 2011. p.39–57. Disponível em: <http://dl.acm.org/citation.cfm?id=2018118.2018122>. Acesso em: 10 jun. 2016

JIANG, J. et al. **Norm Contextualization**. [S.l: s.n.], 2012. p.141–157. Disponível em: <http://dx.doi.org/10.1007/978-3-642-37756-3_9>. Acesso em: 10 jun. 2016

KOLLINGBAUM, M; NORMAN, T. Strategies for resolving norm conflict in practical reasoning. In: **ECAI Workshop Coordination in Emergent Agent Societies**, 2004. Disponível em: <http://www.csd.abdn.ac.uk/~mkolling/publications/KollingbaumNorman_ECAI2004_WS.pdf>. Acesso em: 10 de jun. 2016

LI, T. et al. Contextualized Institutions in virtual organizations. In: BALKE, Tina et al. (Orgs.). . **Coordination, Organizations, Institutions, and Norms in Agent Systems IX**. Lecture Notes in Computer Science. [S.l.]: Springer International Publishing, v. 8386, 2014. p. 136–154. Disponível em: <http://dx.doi.org/10.1007/978-3-319-07314-9_8>. Acesso em: 10 jun. 2016

LÓPEZ. F. L. Y. **Social Power and Norms**: impact on Agent Behaviour. [S.l: s.n.], 2003.

MEYER, J. J. C; WIERINGA, R. J. **Deontic logic in computer science**: normative system specification. [S.l.]: John Wiley and Sons Ltd., 1994.

MUCHNICK, S. S. **Advanced compiler design and implementation**. [S.l.]: Morgan Kaufmann, 1997.

RUSSELL, S; NORVIG, P. **Artificial Intelligence**: a modern approach. 3. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

SANTOS NETO, B. F. dos. **Desenvolvimento de agentes normativos**. Rio de Janeiro: PUC-Rio, 2012.

SILVA, Viviane Torres da. From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. In: **Journal Autonomous agents and multi-Agent systems**, v. 17, n. 1, 2008. p. 113–155. Disponível em: <http://dx.doi.org/10.1007/s10458-008-9039-8>. Acesso em: 10 jun. 2016

SILVA, V. T. da; ZAHN, J. **Normative conflicts that depend on the domain**. [S.l: s.n.], 2013. p.311–326. Disponível em: <http://dx.doi.org/10.1007/978-3-319-07314-9_17>. Acesso em: 10 jun. 2016

SILVESTRE, Eduardo Augusto. **Verificação de conflitos entre múltiplas normas em sistemas multiagentes**. Rio de Janeiro: Universidade Federal Fluminense, 2017.

VASCONCELOS, W. W.; KOLLINGBAUM, M. J; NORMAN, T. J. Normative conflict resolution in multi-agent systems. In: **Autonomous agents and multi-agent systems**, v. 19, n. 2, 2009. p. 124–152 . Disponível em: <http://dx.doi.org/10.1007/s10458-008-9070-9>. Acesso em: 10 jun. 2016

VON WRIGHT, G. H. Deontic Logic. In: **Mind**. Oxford University Press, v. 60, n. 237, 1951. p. 1–15 .

WOOLDRIDGE, Michael. **An introduction to multiAgent systems**. 2. ed. [S.l.]: Wiley Publishing, 2009.