10th Brazilian Congress on Computational Intelligence (CBIC'2011), November 8 to 11, 2011, Fortaleza, Ceará Brazil

AspectNetLogo: Uma Proposta de Linguagem Orientada a Aspectos para a Modelagem de Sistemas Multi-Agentes em Simulações Sociais

Diego de S. Braga[1], Felipe Omena M. Alves[2], Fernando Buarque de L. Neto[1] e Luis Carlos de S. Menezes[1]

[1] Universidade de Pernambuco (UPE), Escola Politécnica de Pernambuco (POLI)
Rua Benfica, 455, Madalena – 50750-410 – Recife, Pernambuco – Brasil
Telefone: (81) 3184-7555 Ramal: 7542
Fax: (81) 3184-7536
{dsb,fbln,lcsm}@ecomp.poli.br

[2] Faculdade Estácio do Recife
Av. Engenheiro Abdias de Carvalho, 1678, Madalena –50720-635 – Recife, Pernambuco – Brasil
Telefone: (81) 3226-8800
felipeomena@gmail.com

Abstract – In computing, aspect-oriented programming (AOP) is a programming paradigm which aims to increase modularity by allowing the separation of cross-cutting concerns. AspectJ is the first complete and powerful language extension for AOP has been created. With this paper we intend to extend the AspectJ approach to NetLogo. We present and discuss a proposal for a set of language extensions that we call AspectNetLogo. The objective is facilitate aspect-oriented programming with NetLogo and illustrate our prototype implementation of a compiler for this new language.

Keywords – AOP, MAS, NetLogo, Social Simulation, AspectJ.

1 Introdução

Simulação Social é uma área de pesquisa que utiliza métodos computacionais na resolução de problemas em ciências sociais (política, economia, antropologia, etc.).

Programação orientada a agentes [1] define a computação a partir de interações sociais de entidades conhecidas como agentes. Um agente é uma entidade que percebe mudanças no ambiente e age em respostas a essas mudanças visando, possivelmente, alcançar algum objetivo pré-determinado. Agentes em um sistema multi-agentes possuem propriedades interessantes tais como: autonomia, pró-atividade, aprendizado, comunicação e coordenação de tarefas distribuídas, entre outras que os habilitam a tratar problemas complexos.

Visando simplificar a execução de simulações sociais utilizando agentes, diversas ferramentas como o PAX [2], o Repast [3] e NetLogo [4] fornecem ao programador uma infra-estrutura básica com os principais elementos de um sistema multi-agente. Essas estruturas foram utilizadas com sucesso para execução de simulações sociais [5][6][7]. Entretanto, da maneira como estão construídas, exigem que o programador tenha um bom conhecimento de sua estrutura, necessária para definir novos tipos de simulações, bem como requer um maior esforço na manutenção dos seus modelos.

A programação orientada a aspectos [8] é uma modelo de programação que foi projetado para simplificar a implementação de *crosscutting concerns* em sistemas computacionais. Um *crosscutting concern* é uma propriedade de uma aplicação que, para ser implementada, exige a modificação de um grande número de módulos na aplicação, não podendo ser isolada utilizando técnicas de modularização convencional. Para implementação destes *concerns*, um aspecto define *pointcuts*, que identificam partes do código do sistema afetadas pelo *concern* (chamadas a métodos, declarações de variáveis, etc.), e *advices*, que descrevem modificações em um *pointcut* (acrescentar novas variáveis instância, redirecionar a execução de métodos, etc) . Desta forma, aspectos isolam os efeitos da implementação do *concern* no sistema.

O uso de aspectos em um sistema orientado a agentes, em especial no NetLogo, irá permitir que elementos que compõe um agente possam ser definidos e estudados isoladamente do resto da simulação. Essa característica pode facilitar a definição de novas simulações e o reuso de elementos de simulações existentes em novos contextos. Entretanto, tradicionalmente implementações de aspectos são baseadas em conceitos da programação orientada a objetos (classes, métodos, herança, etc) que podem não ser adequados para modelagem de conceitos da programação orientada a agentes (comunicações, estados mentais, aprendizado, etc).

Este artigo propõe a definição de uma linguagem orientada a aspectos para descrição de sistemas multi-agentes. Durante esse estudo será analisado problemas de modularidade que podem ser encontrados em sistemas multi-agentes e a adequação de linguagens de aspectos existentes para resolução destes problemas em simulações sociais. A partir deste estudo, novos tipos de aspectos poderão ser identificados.

2 Referencial Teórico

Nessa seção apresentamos os conceitos e teorias que foram utilizados para a elaboração do presente trabalho.

Inicialmente foram levantados aspectos de Computação Inteligente que dão suporte ao desenvolvimento desse artigo, tais como: Sistemas Multi-Agentes Inteligentes Computação Social. Apresentamos também o NetLogo, ambiente e linguagem que foi utilizado como base para o teste de hipótese da solução proposta. Por fim foram apresentados os conceitos básicos da Programação Orientada a Aspectos.

2.1 Sistemas Multi-Agentes Inteligentes

Sistemas de alta complexidade, por exemplo, de alta dimensionalidade ou não monotônicos, são completamente inviáveis ou ineficientes se programados de maneira convencional. Por convencional, deve-se entender: busca e mapeamento de todas as condições possíveis para determinação de ações necessárias. Dessa forma, qualquer condição não mapeada poderá resultar em uma ação não tomada, inconveniente ou errada [9].

Técnicas de Inteligência Artificial, em especial Agentes Inteligentes, possibilitam que sistemas sejam construídos e funcionem de maneira robusta, paralela e eficaz. Sistemas multi-agentes permitem que o programador crie uma infra-estrutura computacional capaz de aprender como ela deve operar e tomar decisões oportunas, sem que todas as situações possíveis precisem ser discriminadas. Além disso, seu uso permite que sistemas baseados em agentes sejam modelados e simulados visando (i) analisar a dinâmica de seus comportamentos; (ii) antever o seu funcionamento; (iii) testar sua robustez e eficiência em ambientes hostis; dentre outras possibilidades.

Um agente deve interagir com o ambiente, o qual está inserido, através de seus sensores e atuadores [1]. Os sensores possibilitam sua percepção do ambiente e são utilizados como entrada para a seleção correta da ação desejada. A escolha do atuador pode ser feita através de inferências, deduções, lógicas, cálculos ou de qualquer outra maneira. A Figura 1 apresenta, esquematicamente, o ciclo da interação do agente com o ambiente.

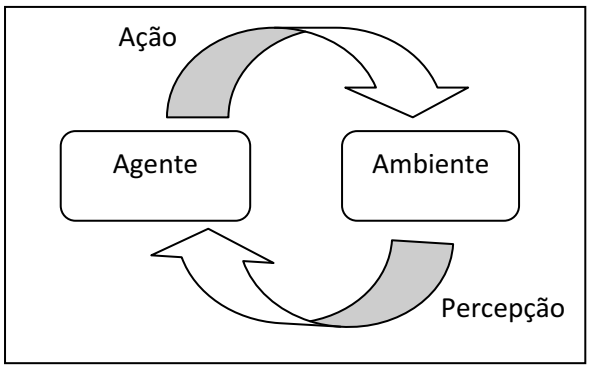

Figura 1 – A interação entre o ambiente e o agente é feita por meio de suas percepções e ações.

Agentes inteligentes incorporam diversas propriedades específicas, tais como: autonomia, pró-atividade, aprendizado, comunicação e coordenação de tarefas distribuídas, entre outras que os habilitam a tratar problemas complexos. A inclusão dessas propriedades do agente é um dos fatores que mais causam problemas na construção de sistemas multi-agentes, pois à medida que a complexidade dos agentes aumenta, essas propriedades tendem a se espalhar através dos vários módulos ou objetos do sistema tornando-o difíceis de manter e reutilizar [10].

2.2 Computação Social

A Computação Social é uma sub-área da Computação Natural que se propõe a simular a dinâmica de sistemas reais e analisar a dinâmica entre os agentes que compõem o ambiente, bem como resolver problemas complexos de forma intuitiva e com custo computacional reduzido [11].

A modelagem e implementação desse paradigma de computação são feitas utilizando-se a técnica de Sistemas Multi-Agentes. Entretanto, ainda não existe uma linguagem que permita uma maior fluência na definição dos modelos e programação dos agentes tornando cara a manutenção e o aperfeiçoamento de um modelo desenvolvido.

2.3 NetLogo

O NetLogo é uma linguagem de programação multi-agente e ambiente integrado de modelagem. É bastante utilizado para modelar sistemas complexos que se desenvolvem ao longo do tempo. Com ele é possível trabalhar com milhares de agentes, todos operando de forma independente. Isto torna possível analisar padrões que emergem a partir da interação de muitos indivíduos e a conexão entre o comportamento desses indivíduos no micro e no macro-nível.

Ele possui uma extensa documentação e tutoriais, além de vir acompanhado de uma biblioteca de modelos, em outras palavras, uma ampla coleção de simulações pré-escritas que podem ser utilizadas e modificadas. Estas simulações abordam conteúdos de áreas das ciências sociais e naturais. Baseado nas funcionalidades do StarLogoT [12], com acréscimo de novos

recursos e uma linguagem e interface com usuário redesenhadas, o NetLogo é a última geração da série de linguagens de modelagem multi-agentes que teve início com o StarLogo [13] Por ser desenvolvido em Java, pode facilmente ser executado na maioria das plataformas (Mac, Linux, Windows etc.).

São algumas das características oferecidas pelo NetLogo: estrutura de linguagem simples; dupla precisão matemática de ponto flutuante; visualização 2D e 3D do modelo; agentes móveis (*turtles*) caminham sobre uma grade de agentes estacionários (*patches*), criação de links entre *turtles* para construir agregados, redes e grafos de agentes; controle de velocidade de simulação; monitores que permitem inspecionar e controlar os agentes; interface gráfica com botões de controle, sistema de gráficos, exportação e importação de modelos, entre outros.

A programação em NetLogo, em sua essência, consiste em atribuir comportamentos ao observador (*observer*), aos agentes (*turtles*) e ao ambiente (*patches*). O observador é uma entidade que especifica as condições de funcionamento e controla os demais elementos. A interação no ambiente é feita pelos agentes e os *patches* (grade do ambiente).

Com a linguagem NetLogo é possível definir diferentes espécies (*breeds*) para os agentes (*turtles*) e o seu comportamento, e até mesmo atribuir diferentes formas (*shapes*) para uma "espécie" de agente, tornando o modelo mais atraente e esclarecedor. Sua linguagem é simples o suficiente para permitir que estudantes possam facilmente executar e até mesmo construir suas próprias simulações e avançado o suficiente para servir como uma poderosa ferramenta para pesquisadores de diversas áreas.

Outra característica que merece destaque no NetLogo é a denominada *Agentset*. Ela possibilita a construção de agentes com regras diferenciadas da sua "espécie". Com ela é possível selecionar um determinado subgrupo dos agentes ou do ambiente e atribuir a eles novos comportamentos.

2.4 Programação Orientada a Aspectos

Conforme afirma PIVETA [16] um sistema que utiliza orientação a aspectos é compostos pelos seguintes componentes:

a) Linguagem de Componente:
Deve permitir ao programador desenvolver as funcionalidades básicas do sistema como, por exemplo, funções e procedimentos, sem se preocupar com as implementações referentes aos aspectos [17]. A linguagem de componente pode ser estruturada ou orientada a objetos. Neste estudo foi utilizado a linguagem Logo.

b) Linguagem de Aspecto:
Deve ser genérica quanto à aplicação e suportar a implementação dos aspectos, definindo seus comportamentos e as situações que irão ocorrer de forma clara e concisa [17]. A linguagem de aspecto utilizada foi o AspectNetLogo, proposta nesse trabalho.

c) Programas de componente:
É o artefato obtido pelo uso das linguagens de componente. Contém as codificações das regras de negócio que formam as funcionalidades e módulos do sistema.

d) Programas de aspectos:
É o arquivo-fonte obtido por meio da linguagem de aspecto, neste caso o AspectNetLogo. Os programas de aspectos podem ser construídos com a utilização de mais de uma linguagem de aspecto[18].

e) Combinador de aspectos:
Responsável por combinar o código desenvolvido na linguagem de componentes (Logo) e o código desenvolvido na linguagem de aspectos (AspectNetLogo). Essa junção de código, conhecida como "*Weaving*", é feita no momento da compilação. Primeiramente se identifica onde os aspectos serão aplicados, e posteriormente as duas implementações são unidas formando código final da aplicação [19] conforme exibido na Figura 2.

3 AspectNetLogo

A programação orientada a aspectos, utilizando o AspectNetLogo, possui quatro conceitos fundamentais, que são os Pontos de Junção, os Pontos de Atuação, os Adendos e os Aspectos. Cada um deles será explicado nessa seção.

3.1 Pontos de Junção

Pontos de junção (*join points*) são locais bem definidos no fluxo de execução de um programa como, por exemplo, a chamada ou a execução de uma função [20]. O objetivo deles é definir locais que um aspecto poderá atuar. É possível que pontos de junção estejam contidos em outros pontos de junção [21]. A Figura 3 apresenta alguns pontos de junção, destacados em vermelho, de um código desenvolvido no NetLogo.

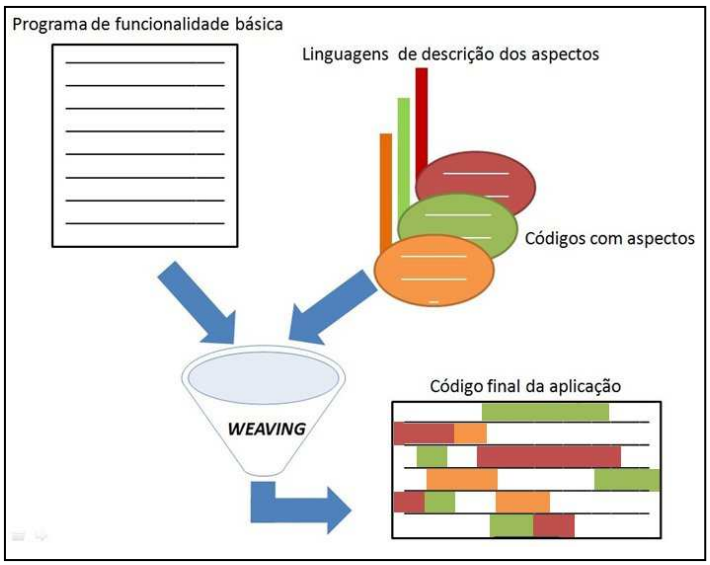

Figura 2 - Ilustra que o processo de "*Weaving*". O artefato obtido pela implementação das funcionalidades básicas do programa em tempo de compilação é unido ao código definido na linguagem de aspectos. O resultado desta combinação é o código final da aplicação.

3.2 Pontos de Atuação

Os pontos de atuação (*pointcuts*) são uma composição entre pontos de junção e operadores de interseção (&&), de união (||) e de complemento (!) [22]. A função dos pontos de atuação é criar uma espécie de regra que especifica eventos que serão adicionados aos pontos de junção ou obter dados do contexto que serão utilizados posteriormente.

Assim como nas linguagens AspectJ [23] e AspectC++ [24], o AspectNetLogo possui designadores de pontos de atuação, que auxiliarão na construção dos pontos de atuação e caracteres referente ao curinga [22], que, por exemplo, poderá ser utilizado na construção de nomes de funções parciais. A Tabela 1 apresenta um quadro comparativo entre algumas características de duas linguagens de aspectos conhecidas (AspectJ e AspectC++) e o AspectNetLogo, linguagem proposta nesse artigo.

O código apresentado em (1) cria um ponto de atuação na linguagem AspectNetLogo chamado "definirAgentes". Esta definição permite identificar o local do código (ponto de junção) ao qual o aspecto irá interferir. Nesse exemplo aspecto irá atuar no momento da execução da função "setup-globals", que é uma função do código original que não tem tipo de retorno e não recebe argumentos.

```
pointcut definirAgentes[] : execution [ setup-globals[] ]
```
(1)

Tabela 1 – Quadro comparativo entre os caractere referente ao curinga e alguns dos designadores existentes no AspectJ, AspectC++ e no AspectNetLogo.

Linguagens / *Características*	*AspectJ*	*AspectC++*	*AspectNetLogo*
Caractere Referente ao curinga	*	%	*
Designador de chamada	call	call	call
Designador de execução	execution	execution	execution
Designador de definição	set	set	-
Designador de acesso	get	get	-

A sintaxe do AspectNetLogo é bem similar a do AspectJ e a do AspectC++. O motivo dessa similaridade é tentar permitir que programadores que já utilizaram alguma dessas linguagens mais populares diminuam o tempo necessário para aprender a utilizar o AspectNetLogo.

```
1.  to go
2.    if ticks >= 200 [stop]
3.    move-turtles
4.    eat-grass
5.    check-energy
6.    regrow-grass
7.    do-plots
8.    tick
9.  end
10.
11. to check-energy
12.   ask turtles [
13.   ifelse energy > 50 [
14.   set energy energy - 50
15.   hatch 1 [set energy 50]
16.   ] ;; reproduce
17.   ]
18.   [ check-death ] ;; is to die?
19. end
```

Figura 3 – Identificação de pontos de junção em um código desenvolvido no NetLogo. O primeiro ponto de junção é a invocação da função *check-energy* (linha 5). A execução desta função é considerada outro ponto de junção (linha 11), assim como a chamada a *check-death* (linha 18).

3.3 Adendos

Adendos (*advices*) definem um código adicional que será executado nos pontos de junção, com base na ocorrência de pontos de atuação [25]. Enquanto os pontos de junção e os pontos de atuação referem-se a código já existente, os adendos serão declarados em aspectos e só poderão fazer referência a código de aspectos.

O momento em que o código do adendo será inserido no ponto de junção no AspectNetLogo poderá ser antes (*before*), depois (*after*) e depois do retorno (*after returning*). Além dessas possibilidades, existe um modo que permite alteração de uma função, que é denominado *around*.

Em (2) temos o exemplo, em AspectNetLogo, de uma declaração de um adendo que especifica que após a execução do ponto de atuação, declarado na seção 3.3.2, haverá uma chamada a função "setup-people". Ou seja, um adendo depende da ocorrência do ponto de atuação no código original, para que os serviços desejados sejam inseridos.

```
after[] : definirAgentes[]
        setup-people
```
(2)

3.4 Aspectos

Aspecto é um conjunto de definições de pontos de atuação e adendos em uma unidade modular de implementação [26], que nesta utilização tem como objetivo adicionar ou alterar comportamento dos agentes na simulação de um sistema multi-agente. Esse mecanismo, disponibilizado pela programação orientada a aspectos, serve para agrupar fragmentos de código referente aos componentes não-funcionais de um sistema [27]. Os aspectos implementados utilizando o AspectNetLogo poderão interferir em definidos pontos do código original do modelo em NetLogo, sem que o código em NetLogo sofra qualquer modificação, tornando, desta forma, mais fácil a manutenção do código do modelo.

Em (3) é possível exemplificar o uso de um Aspecto na linguagem AspectNetLogo tomando como base as definições apresentadas previamente nesse artigo (ver seções 3.1; 3.2; 3.3).

```
aspect Agentes[
pointcut definirAgentes[] : execution [ setup-globals[] ]
after[] : definirAgentes[]
        setup-people ]
```
(3)

4 Estudo de Caso

Para a validar o uso do AspectNetLogo foi escolhido um modelo de simulação social disponibilizado na biblioteca de modelos do Netlogo denominado AIDS [14], que visa simular a propagação do vírus da imunodeficiência humana (HIV) especificamente através do contato sexual em uma determinada população.

Por meio desta modelagem foi possível identificar que uma simples alteração na forma em que os indivíduos tomam decisões iria impactar na necessidade de mudanças em várias partes do código. Por exemplo, para que seja modificada a tendência dos indivíduos a prática de sexo se faz necessário a manutenção em seis funções do modelo. A Figura 4(a) ilustra esse problema e a definição de um aspecto que modela o comportamento do agente no modelo do estudo de caso.

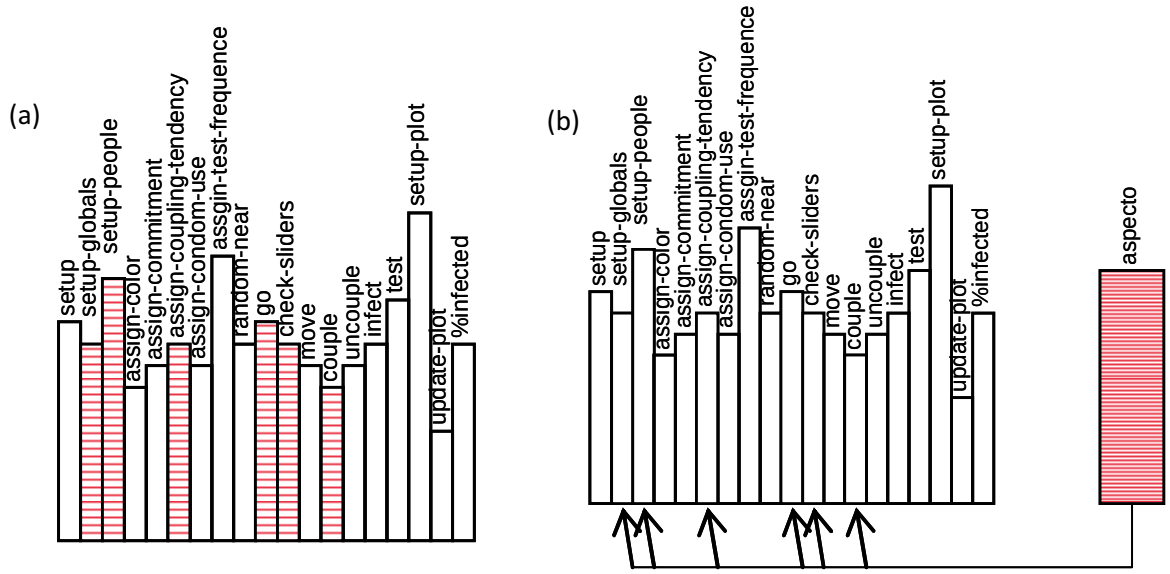

Figura 4 – (a) Cada coluna da Figura representa uma função existente no modelo de simulação social da AIDS. Uma possível alteração no modo na tendência dos indivíduos a prática do sexo impacta em 31% das funções (colunas hachuradas).

(b) O aspecto define os comportamentos que o agente terá, e por meio dos pontos de atuação são definidos locais do código aos quais o aspecto irá interferir. Dessa forma, futuras alterações no código só precisarão ser feitas no aspecto.

Uma boa prática utilizada para construir sistemas de software é a de separação de interesses ou responsabilidades distintas (*separation of concerns*) [15]. Com esse artifício o sistema se torna mais robusto, pois se houver necessidade de redefinições em algum modulo estas não irão afetar outras partes do sistema a não ser daquele em questão. Quando se consegue construir diferentes responsabilidades em módulos separados, se alcança uma maior eficiência no desenvolvimento de software.

A necessidade de se tratar interesses distintos sem impactar em partes não desejadas no sistema, fez com que a orientação a aspecto trouxesse um novo conceito que complementa a orientação a objetos: os Aspectos. Pode-se definir aspectos como sendo unidades modelares cujo objetivo é adicionar informações ao código criando um sistema único, com vários interesses, porém sem que necessariamente um interfira no outro.

A utilização do AspectNetLogo traz a possibilidade de se modelar agentes com separação total de responsabilidades. As características e os comportamentos dos agentes seriam implementados de forma isolada em aspectos, tornando a manutenção mais simples e prática. Uma solução proposta para o exemplo citado acima, seria a construção de um aspecto que implementasse as definições de tendência a sexo, e por meio dos pontos de atuação definiria as partes do código ao qual o aspecto irá interferir. Deste modo, alterações futuras iriam ser realizadas apenas no aspecto construído. A Figura 4(b) ilustra a definição de um aspecto que modela o comportamento do agente no modelo do estudo de caso.

A função "*setup-people*" do modelo AIDS é apresentada no Algoritmo 1. Ela define variáveis e faz chamada a funções com diferentes propósitos como, por exemplo, definições de tendência à pratica do sexo, tendência a exames de detecção da doença e duração do tempo de uma relação conjugal, tornando o código pouco coeso.

Com a utilização do AspectoNetLogo, essas definições poderiam ser tratadas de formas especificas, o que reduziria a quantidade de linhas do método original e o torna mais coeso, conforme apresentado no Algoritmo 2. Além disso, outras características que por ventura tenham sido declaradas em diferentes lugares do código e estejam relacionadas aos interesses listados acima, também serão removidas e unidas na declaração desses aspectos A declaração de um Aspecto é feita no Algoritmo 3. Dessa forma, caso haja necessidade futura de alteração em alguma dessas definições, somente os aspectos serão modificado e as funções do código não serão afetadas.

Algoritmo 1 Método "setup-people" (modelo Aids) sem modificações.

```
1  to setup-people
2   crt initial-people
3     [ setxy random-xcor random-ycor
4       set known? false
5       set coupled? false
6       set partner nobody
7       ifelse random 2 = 0
8       [ set shape "person righty" ]
9       [ set shape "person lefty" ]
10      ;; 2.5% of the people start out infected, but they don't know it
11      set infected? (who < initial-people * 0.025)
12      if infected?
13        [ set infection-length random-float symptoms-show ]
14      assign-commitment
15      assign-coupling-tendency
16      assign-condom-use
17      assign-test-frequency
18      assign-color ]
19 end
```

Algoritmo 2 Método "setup-people" com o uso de AspectNetLogo.

```
1  to setup-people
2   crt initial-people
3     [ setxy random-xcor random-ycor
4       ifelse random 2 = 0
5       [ set shape "person righty" ]
6       [ set shape "person lefty" ] ]
7  end
```

Algoritmo 3 Definição do Aspecto que controla a tendência dos indivíduos a prática de sexo utilizando AspectNetLogo.

```
aspect PraticaDeSexo[

pointcut definirTendenciaPraticaSexo[] : execution [ setup-people[] ]

after[] : definirTendenciaPraticaSexo []

        set coupling-tendency random-near average-coupling-tendency

]
```

5 Conclusão e Trabalhos Futuros

Nesse artigo foi apresentada uma proposta de linguagem orientada a Aspectos para a modelagem de Sistemas Multi-Agentes em Simulações Sociais. O objetivo do AspectNetLogo é dotar o NetLogo da capacidade de utilizar Aspectos para a modelagem de sistemas multi-agentes.

Embora tenha sido necessária a realização de alterações em algumas partes da sintaxe e gramática do AspectJ e do AspectC++, a maior parte dos conceitos dessas linguagens foi preservado. Com isso espera-se que usuários experientes em AspectJ (e/ou AspectC++) e NetLogo possam se familiarizar com o AspectNetLogo sem grande esforço.

Como continuação do trabalho apresentado é proposta a realização de um estudo mais detalhado nos modelos de agentes sociais existentes, pois acreditamos que esse estudo poderá ajudar a identificar novos tipos de aspectos para sistemas multi-agentes. A continuação desse trabalho contempla a criação de uma linguagem de domínio específico e a construção de um Framework para simulações sociais.

6 Referencias

[1] RUSSELL, Stuart and NORVIG, Peter. Artificial Intelligence: A Modern Approach. Prentice Hall, 1995.

[2] LIMA NETO, Fernando Buarque de.; PITA, Marcelo R. S.; BARBOSA FILHO, Hugo Serrano. "Hybrid and Evolutionary Agent-Based Social Simulations Using the PAX Framework". In: Nineth International Conference on Intelligent Systems Design and Applications - ISDA, Pisa, Italy, 2009.

[3] M. NORTH, T. HOWE, N. COLLIER, and J.VOS. The repast simphony runtime system. In Proceedings of the Agent 2005 Conference on Generative Social Processes. Models, and Mechanisms, 2005.

[4] NETLOGO itself: Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[5] PITA, Marcelo, LIMA NETO, Fernando Buarque de. BARBOSA FILHO, Hugo Serrano. Impact of Communication on Agent-Based Social Simulations Using PAX Framework. In> IEEE International Conference on Systems,Man, and Cybernetics, 2009, Texas - USA.

[6] PAVÓN, J., Arroyo, M., Hassan, S., Sansores, S.: Agent-based modelling and simulation for the analysis of social patterns. Pattern Recogn. Lett. 29, 1039–1048 (2008).

[7] BALBINOTTO NETO, Giácomo ; Luiz Marcelo Berger ; Denis Borenstein . A Multiagent Method Applied to the Economic Analysis of Criminal Law. Economic Analysis of Law Review, v. 1, p. 161-173, 2010.

[8] PROGRAMMING, Aspect-oriented. Disponível em: < http://en.wikipedia.org/wiki/Aspect-oriented_programming/>. Acesso em: 15 de Maio de 2011.
[9] PESSOA, Luis Filipe de Araújo. UNIVERSIDADE DE PERNAMBUCO, Escola Politécnica de Pernambuco. Implementação de um Mecanismo Automático de Controle de Vizinhanças Entre Agentes de uma População Inspirada em Alcatéia, 2009. 78p, Monografia.
[10] GARCIA, Alessandro Fabricio. Objetos e Agentes: Uma Abordagem Orientada a Aspectos. Tese apresentada como requisito parcial para obtenção do título de Doutor pelo Programa de Pós-Graduação em Informática da PUC-Rio. Rio de Janeiro, 2004.
[11] BONABEAU, E.; DORIGO, M.; THERAULAZ, T. Swarm Intelligence: From Natural to Artificial Systems. New York: Oxford University Press, 1999.
[12] STARTLOGO,Disponível em: <http://education.mit.edu/starlogo/>. Acesso em: 15 de Maio de 2011.
[13] STARLOGOT. Center for Connected Learning and Computer-Based Modeling, Tufts University. Disponível em: <http://ccl.northwestern.edu/cm/starlogoT/> Acesso em: 15 de Maio de 2011.
[14] WILENSKY, U. 1997. NetLogo Aids Model. Disponível em: <http://ccl.northwestern.edu/netlogo/models/index.cgi/>. Acesso em: 15 de Maio de 2011.
[15] DEXTRA. Entendendo a Programação Orientada a Aspectos. Disponível em: <http://www.dextra.com.br/empresa/artigos/aspectprog.htm/>. Acesso em: 15 de Maio de 2011.
[16] PIVETA, Eduardo (2001). Um modelo de suporte a programação orientada a aspectos. UFSC. Dissertação submetida como parte dos requisitos para obtenção do grau de Mestre em Ciências da Computação.
[17] IRWIN, John, KICKZALES, Gregor , LAMPING, John, MENDHEKAR, Anurag, MAEDA, Chris, LOPES, Cristina Videira, LOINGTIER, Jean-Marc. Aspect - Oriented Programming. Proceeding of ECOOP'97, Finland: Springe r- Verlag, 1997.
[18] WAZLAZWICK, Raul Sidnei. Análise de Projeto de Sistemas de Informação Orientados a Objetos. Rio de Janeiro: Elsevier, 2004.
[19] CHAVES, Rafael Alves (2002). Aspectos e Middleware. UFSC. Dissertação submetida como parte dos requisitos para obtenção do grau de Mestre em Ciências da Computação.
[20] SOARES, Sergio; BORBA, Paulo. AspectJ - Programação orientada a aspectos em Java. Tutorial no SBLP 2002, 6o. Simpósio Brasileiro de Linguagens de Programação. 5 a 7 de Junho, PUC-Rio, Rio de Janeiro, Brasil, p. 39–55, 2002.
[21] FRÖHLICH, Antônio Augusto; VIEIRA, Fernanda Ariane Range; FONTANA, Heitor de Brito; BRAGA, Marcelo. Programação Orientada a Aspectos, Disponível em: <http://www.lisha.ufsc.br/teaching/sce/ine5612-2001-2/work/aop.html/>. Acesso em: 15 de Maio de 2011.
[22] LEMOS, O. A. L. ; MALDONADO, José Carlos ; MASIERO, Paulo Cesar . Data Flow Integration Testing Criteria for Aspect-Oriented Programs. In: Primeiro Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos (WASP'04), 2004, Brasília.
[23] WINK, Diogo Vinicius; Junior, Vicente Goetten. AspectJ: Programação Orientada a Aspectos com Java. São Paulo: Novatec Editora, 2006.
[24] GAL, Andreas; WOLFGANG Schröder-Preikschat, and OLAF Spinczyk: "AspectC++: Language Proposal and Prototype Implementation", Proceedings of the OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems, Tampa, Florida, 2001.
[25] LOUREIRO. João Manuel Bonita Pereira, COSTA. João Pedro Couto Soares Gonçalves da, FONSECA. Rosana Mendes Sequeira Baptista da, LOUREIRO. Vergílio Augusto Neves. Programação Orientada a Aspectos. Disponível em: <http://paginas.fe.up.pt/~aaguiar/es/artigos%20finais/es_final_15.pdf>. Acesso em: 15 de Maio de 2011.
[26] SCRIBD. A Programação Orientada a Aspectos. Disponível em: <http://pt.scribd.com/doc/2625512/Programacao-Orientada-a-Aspectos/>. Acesso em: 15 de Maio de 2011.
[27] WINCK, Diogo Vinícius; Junior, Vicente Goetten. AspectJ em 20 minutos. Disponível em: <http://javafree.uol.com.br/artigo/871488/ AspectJ-em-20-minutos.html/>. Acesso em: 15 de Maio de 2011.