UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DEIVIDI FERNANDO DA SILVA MOREIRA

Ferramenta para a Modelagem de Simulações baseadas em Agentes usando Linguagem Específica de Domínio

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Profª. Drª. Ingrid Oliveira de Nunes
Co-orientador: M.Sc. Fernando dos Santos

Porto Alegre
2016

RESUMO

A modelagem e simulação baseada em agentes, ou *Agent-based Modeling and Simulation* (ABMS), é uma abordagem para o desenvolvimento de sistemas compostos de agentes autônomos que interagem para simular sistemas complexos, nos quais o comportamento emergente é usualmente desconhecido. Um número crescente de modelos baseados em agentes surge com os avanços computacionais, em uma variedade de domínios de aplicação. Um exemplo são aplicações para observar o comportamento de um agente (pessoa, animal, inseto, etc.) após desastres naturais, para prever propagação de epidemias ou compreender fatores responsáveis pelo declínio de civilizações antigas. No entanto, o desenvolvimento destas aplicações ainda é muito custoso, visto que envolve especialistas no domínio em questão, em ABMS, e em programação. Assim, este trabalho apresenta ferramenta ABM Tool que permite a criação de modelos usando uma linguagem específica de domínio (DSL). Esta ferramenta, e sua DSL subjacente, abstraem detalhes de programação para dar suporte a ABMS, possibilitando ao usuário a criação de modelos de alto nível utilizando um editor gráfico. A ferramenta dispõe de uma paleta de elementos para criação do modelo de simulação, tais como entidade, atributos, parâmetros, unidades espaciais, entre outros. O foco da ferramenta é em particular ainda restrito ao ambiente simulado, mas serve de evidência dos benefícios do uso de DSL neste contexto. Em uma avaliação com usuários, a ferramenta se mostrou útil, fácil de usar e eficaz para construção de ABMS.

Palavras-chave: ABMS. Modelagem de simulação baseada em agentes. Ferramenta. Editor Gráfico. NetLogo.

Tool for Simulation Modeling based agents using Domain Specific Language

ABSTRACT

Agent-based modeling and simulation (ABMS) is an approach for the development of systems composed of autonomous agents that interact to simulate complex systems in which emerging behavior is usually unknown. There is an increasing number of agent-based models due to computational advancements in a variety of application domains. Examples are applications to observe the behavior of an agent (person, animal, insect, etc.) after natural disasters, to predict the spread of epidemics or to understand factors responsible for the declension of ancient civilizations. However, the development of these applications is requires significant effort, since it involves experts in the domain in question, in ABMS, and in programming. Thus, this work presents the ABM Tool, which allows the creation of models using a domain-specific language (DSL). This tool, and its underlying DSL, abstracts programming details to support ABMS, enabling the user to create high-level models using a graphical editor. The tool has a palette of elements to create the simulation model, such as entity, attributes, parameters, spatial units, among others. The focus of the tool, is in particular still restricted to the simulated environment, but serves as evidence of the benefits of using the DSL in this context. In an evaluation made with users, the tool has shown to be useful, easy to use and effective for building ABMS.

Keywords: ABMS, Modeling and agent-based simulation, Tool, Graph editor, NetLogo.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABMS	Agent-based Modeling and Simulation
BPM	Business Process Management
DTA	Diagram Type Agent
DSL	Domain Specific Language
EMF	Eclipse Modeling Framework
GPL	General Purpose Language
HTML	HyperText Markup Language
IDE	Ambiente de desenvolvimento integrado
MAS	Multiagent System
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
Yacc	Yet Another Compiler-Compiler
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Unified Modeling Language
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Por muitos anos, o desenvolvimento, a análise e a experimentação de modelos tem sido parte dos instrumentos de praticamente todos os domínios da ciência e da engenharia (KLÜGL; BAZZAN, 2012). As ferramentas de modelagem surgiram para facilitar a criação dos modelos. Na engenharia de software, existem ferramentas de modelagem avançadas que permitem inclusive a geração do banco de dados ou do código fonte de sistemas de software a partir de modelos.

Modelagem é o processo de construção de uma abstração de um sistema para uma finalidade específica, tendo como resultado um modelo. Um modelo é uma abstração que mantem apenas os elementos considerados relevantes (GALÁN et al., 2009). Um modelo pode ser construído como um programa de computador que geralmente usa uma representação digital simplificada de um ou mais aspectos do mundo real (CASTLE et al., 2005).

A modelagem e simulação baseada em agentes, ou *Agent-based Modeling and Simulation* (ABMS), é um paradigma de simulação que utiliza agentes autônomos e sistemas multiagentes para produzir fenômenos investigados (KLÜGL; BAZZAN, 2012). Através da ABMS é possível observar como os agentes (pessoas, bactérias, insetos, nações ou organizações) interagem entre si e com seu ambiente (BONABEAU, 2002). A ABMS está sendo aplicada em muitas áreas, abrangendo sistemas sociais, comportamentais, culturais, físicos e biológicos. As aplicações vão desde a modelagem de civilizações antigas, até a concepção de novos mercados para produtos que não existem no momento (MACAL; NORTH, 2014).

Outros modelos baseados em agentes são usados em grande escala para simular fenômenos da natureza, no qual um sistema é modelado em detalhes, aplicando dados reais ao modelo validado, gerando resultados que são destinados a informar políticas e tomada de decisão. Estas aplicações vêm sendo possíveis graças aos avanços no desenvolvimento de software especializado em ABMS, a disponibilidade de dados para aumentar os níveis de granularidade, e avanços no desempenho dos computadores (MACAL; NORTH, 2010).

A concepção, implementação e uso de uma ABMS requer colaboração entre especialistas na área de aplicação e especialistas em modelos de simulação (por exemplo, Sociologia e Ciência da Computação) (SANTOS; NUNES; BAZZAN, 2016). O papel do especialista na área se destina a produzir a primeira conceitualização do sistema alvo.

Esta tarefa envolve a definição dos objetivos, identificando os componentes críticos do sistema e suas ligações, e também descreve as relações causais mais proeminentes. A saída desta primeira etapa do processo é na maioria das vezes um modelo informal expresso em linguagem natural, e também pode incluir diagramas conceituais simples. O especialista em modelos de simulação transforma o modelo informal que o especialista no domínio pretende explorar, nas especificações de requisitos (formais) que o cientista da computação tem de formular (GALÁN et al., 2009).

Esta tarefa envolve grandes desafios, como o fato de que, na maioria dos casos, o modelo informal criado pelo especialista no domínio não está completamente especificado, já que é muitas vezes formulado usando linguagem natural. A terminologia também é um desafio, podendo haver termos idênticos para as duas áreas com conotações totalmente diferentes, ou termos desconhecidos a ciência da computação (GALÁN et al., 2009). Logo, é importante dispor de meios que permitam reduzir a lacuna representacional, possibilitando criar, no computador, modelos de simulação. Existem vários ambientes de programação especificamente projetados para desenvolver simulações baseadas em agentes, entre eles o StarLogo (RESNICK, 1996), Swarm (HIEBELER et al., 1994), RePast (COLLIER, 2003) e NetLogo (WILENSKY, 1999) que dependem de conhecimento em programação.

No contexto de Engenharia de Software, existe uma abordagem que pode potencialmente complementar esses ambientes de programação diminuindo os desafios da ABMS. As Linguagens Específicas de Domínio, ou *Domain Specific Languages (DSLs)*, permitem que as soluções sejam expressas no idioma e no nível de abstração do problema de domínio. Elas oferecem poder expressivo focado, restrito a um domínio de problema particular, por meio de notações e abstrações apropriadas (DEURSEN; KLINT; VISSER, 2000; FOWLER, 2010). Uma DSL pode ser usada para gerar os membros de uma família de sistemas de um domínio de aplicação, por exemplo ABMS. As vantagens potenciais das DSLs incluem a redução no tempo de aprendizagem e nos custos de manutenção, maior portabilidade, confiabilidade, otimização e testabilidade (DEURSEN; KLINT, 1998).

Santos, Nunes and Bazzan (2016) propuseram uma DSL para o domínio da ABMS focando no ambiente simulado, com o objetivo de elevar o nível de abstração da ABMS para o nível da área de aplicação. Entretanto, esta DSL ainda carece ferramenta que a suporte. Assim, o objetivo deste trabalho é apresentar uma ferramenta denominada ABM Tool, baseada na notação gráfica proposta na DSL de Santos, Nunes and Bazzan (2016),

para especificar o ambiente simulado.

A ABM Tool foi construída como um complemento para a IDE Eclipse, utilizando a linguagem Java e o plug-in Graphiti. Como característica principal, ABM Tool possui um editor gráfico e uma paleta de elementos, que possibilitam a criação de modelos rapidamente, proporcionando ao usuário uma interação amigável, levando em conta as experiências que o usuário possa ter em outras ferramentas de modelagem.

Como benefício, a ferramenta pode auxiliar na modelagem de cenários por pessoas sem conhecimentos em programação e plataformas de simulação. Ela também possibilita a independência de uma plataforma de simulação, uma vez que a DSL permite a transformação do modelo para diferentes plataformas de simulação, por exemplo, para a plataforma NetLogo.

Para validar a ferramenta proposta, foi realizado uma avaliação com alunos de graduação e pós-graduação de cursos da área de ciência computação. A avaliação contou com 10 alunos, todos familiarizados com ferramentas de modelagem, mas nem todos familiarizados com a modelagem baseada em agentes.

O restante do trabalho está estruturado da seguinte forma. O Capítulo 2 trata dos fundamentos em modelagem e simulação baseada em agentes e linguagens específicas de domínio. O Capítulo 3 apresenta as funcionalidades, arquitetura e tecnologias utilizadas na implementação da ferramenta e apresenta a ferramenta desenvolvida, detalhando a forma de utilização. O Capítulo 4 apresenta a avaliação da ferramenta realizada. O Capítulo 5 trata dos trabalhos relacionados. Por fim, o Capítulo 6 aborda as conclusões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, é apresentada uma visão geral sobre modelagem e simulação baseada em agentes e os principais conceitos e aplicações das linguagens específicas de domínio. São apresentadas suas características, objetivos, vantagens, áreas de aplicação e exemplos.

2.1 Modelagem e Simulação Baseada em Agentes

Podemos entender um modelo como sendo uma representação simplificada da realidade. O processo de modelar tem como pontos fundamentais: a escolha de um nível adequado de abstração, de forma a considerar informações relevantes ao objeto de estudo e desconsiderar aquelas que não são pertinentes; e a linguagem utilizada para representação do modelo.

Os modelos podem ser estáticos, se os dados de entrada e de saída correspondem a um mesmo ponto no tempo, ou dinâmicos, se os dados de saída representam um ponto posterior no tempo. Modelos estáticos fornecem indicadores que podem prover algumas previsões de impactos, sensibilidades ou vulnerabilidades. Modelos dinâmicos vão mais longe, tentando projetar impactos quantificáveis para o futuro, como por exemplo, o número de sobreviventes em uma epidemia (LONGLEY, 2005).

A tarefa de construir um modelo pressupõe um objetivo associado a este propósito, que permitirá definir quais informações são importantes e devem ser representadas e escolher a linguagem mais adequada para descrever tais representações. A simulação de um modelo consiste na execução deste modelo (KLÜGL; BAZZAN, 2012).

A modelagem e simulação de sistemas tanto pode ser útil para contribuir para o entendimento do objeto de estudo quanto como procedimento metodológico científico, através da representação de uma hipótese científica.

Modelagem e simulação baseada em agentes, ou *Agent-based Modeling and Simulation* (ABMS), é uma das metodologias (ou técnicas) disponíveis para construção de modelos, e sua utilização está se expandindo rapidamente em diversos campos da ciência, tais como ciências sociais e ambientais. A ABMS aplica o conceito de sistemas multiagentes para a estrutura básica de modelos de simulação, onde o sistema é modelado como uma coleção de entidades autônomas de tomadas de decisão chamadas agentes (NORTH; MACAL, 2007). Um sistema, que consiste em um grupo de agentes que podem potenci-

almente interagir uns com os outros, é chamado de Sistema multiagentes ou *Multiagent system* (MAS) (WOOLDRIDGE, 2001). A ideia central da ABMS é que os fenômenos podem ser gerados a partir das ações e interações do sistema multiagente (BONABEAU, 2002).

Um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores (RUSSELL et al., 2003). No entanto, os agentes são sistemas raramente autossuficientes. Em muitas situações eles coexistem e interagem com outros agentes de várias maneiras diferentes. Exemplos incluem agentes de software na Internet, robôs de futebol jogando, e muitos outros. Para Macal and North (2010) um agente deve ter as seguintes características: (1) *ser identificável, isto é, um indivíduo discreto com um conjunto de características e regras que governam seu comportamento e capacidade de tomada de decisões;* (2) *estar situado, isto é, habitar em um ambiente com o qual interage e também no qual interage com outros agentes;* (3) *ser orientado por objetivos;* (4) *ser autônomo;* (5) *ser flexível;* e (6) *possuir habilidade para aprender e adaptar seu comportamento através do tempo baseado em experiências*.

ABMS é particularmente adequada para a análise de sistemas adaptativos complexos e fenômenos emergentes em ciências sociais, tráfego, biologia e outros. Assim, uma estrutura ou comportamento emergente é gerado pela interação local das entidades, apesar de serem apenas observáveis em uma escala global, não sendo diretamente dedutível de comportamentos locais (KLÜGL; BAZZAN, 2012).

Para Macal and North (2006), o processo de se construir modelos baseados em agentes possui alguns aspectos únicos devido ao fato de se tomar por base a perspectiva de agentes ao invés da perspectiva baseada em processos da modelagem e simulação tradicional. Os autores elencam os seguintes passos gerais para a construção de modelos de agentes:

- *agentes* - identificar os tipos de agentes e seus atributos;
- *ambiente* - definir o ambiente no qual os agentes irão habitar e interagir;
- *métodos de agentes* - especificar os métodos pelos quais os atributos dos agentes serão atualizados em resposta às interações entre agentes e entre agentes e ambiente;
- *interações de agentes* - especificar os métodos de interação entre os agentes; como e quando interagem durante uma simulação; e
- *implementação* - implementar o modelo de agentes em um sistema computacional.

Para exemplificar, considere uma ABMS simples, com predadores (lobos), presa (ovelha) e, alguns recursos que servem de alimento para a presa (por exemplo, grama). Os agentes são os lobos e ovelhas. O comportamento dos agentes consiste na realização de um passeio aleatório, enquanto interações ocorrem quando dois agentes são suficientemente perto uns dos outros. Estas interações são implementadas como partes do comportamento do agente. Por exemplo, se um lobo encontra uma ovelha, ele come a ovelha e aumenta seu nível de energia. O ambiente consiste na representação espacial onde os objetos estão espalhados pela grama. O crescimento e a disponibilidade da grama estão relacionadas às variáveis globais associadas a temperatura e umidade, que mudam de acordo com alguma função, portanto, seu aparecimento e desaparecimento é um processo ambiental.

Por meio da simulação, é possível observar como agentes individuais interagem entre si e com seu ambiente. No nosso exemplo, podemos observar a quantidade de ovelhas sobreviventes, a velocidade em que elas são devoradas, a quantidade de grama disponível em cada área e o nível de energia dos lobos, e estudar os fenômenos ocorridos durante a simulação, como por exemplo, morte de todos os agentes, ou o surgimento de uma superpopulação de ovelhas.

A ABMS apresenta alguns benefícios em relação a outras técnicas de modelagem, pois permite capturar fenômenos emergentes, fornece uma descrição natural de certos tipos de sistemas e, é flexível (BONABEAU, 2002). Para Klügl and Bazzan (2012), seu poder explicativo decorre de sua natureza generativa, que permite a observação e análise da dinâmica do modelo no nível de agente local e no nível macroscópio.

2.2 Linguagens Específicas de Domínio

Uma linguagem específica de domínio, ou *Domain Specific Language* (DSL), é uma linguagem de programação ou linguagem de especificação executável que oferece, por meio de notações e abstrações apropriadas, poder expressivo focado a um domínio de aplicação particular (DEURSEN; KLINT; VISSER, 2000). Um domínio consiste de um ponto de vista de uma determinada área, tal como uma família de produtos ou um determinado assunto específico. Uma DSL deve possuir a capacidade de desenvolver aplicações completas para um domínio específico, capturar precisamente a semântica de um domínio de aplicação, uma DSL não é necessariamente genérica (HUDAK, 1998; MERNIK; HEERING; SLOANE, 2005). Exemplos comuns de DSLs incluem PERL para

manipulação de texto, VHDL para descrição de hardware, TeX e LaTeX para preparação de documentos, HTML para marcação de documentos, OpenGL para gráficos 3D, Maple para computação simbólica, e AutoCAD para desenho auxiliado por computador.

As DSLs têm algumas propriedades que as Linguagens de Propósito Geral ou *General Purpose Languages* (GPLs), como C++, Java, C# e UML, não têm. Por exemplo, com DSLs, conceitos de domínio estão diretamente representados por construções sintáticas. Isso muitas vezes permite especificações mais concisas e precisas, que até mesmo os especialistas de domínio possam entender (DEURSEN; KLINT; VISSER, 2000). Um usuário da DSL concentra seus esforços na descrição do domínio enquanto as decisões de complexidade, concepção e implementação e detalhes estão ocultas (LANGLOIS; JITIA; JOUENNE, 2007).

Uma DSL pode ser uma linguagem diagramática, uma linguagem textual ou uma combinação dessas características. Em alguns domínios, é mais adequada a implementação de linguagens diagramáticas, as quais, quase sempre, são mais expressivas em domínios relativamente não técnicos do que as linguagens textuais. Como exemplo, podemos citar a linguagem SQL como DSL textual, e linguagens de modelagem de processo como a BPM (BARZDINS et al., 2009), como DSL diagramáticas.

As DSLs permitem que as soluções sejam expressas no idioma e no nível de abstração do problema de domínio. Por consequência, os especialistas de domínio podem compreender, validar, alterar e, eventualmente desenvolver programas em uma DSL. Para Ladd and Ramming (1994), sistemas de software programados com base em uma DSL são normalmente concisos, auto documentados e podem ser reutilizados para diferentes propósitos. Para Deursen and Klint (1998) e Kieburtz et al. (1996), as DSLs podem melhorar a produtividade, clareza e a manutenção. Deursen, Klint and Visser (2000) afirmam que as DSLs incorporam conhecimento de domínio, e desta forma, permitem a conservação e reutilização deste conhecimento.

Para exemplificar a utilização de uma DSL, considere o trecho de código na linguagem Java na Figura 2.1, que instancia de objeto máquina, especifica seu processador e unidades de disco. Já na Figura 2.2 é ilustrado o mesmo trecho de código usando uma DSL. Note que a DSL simplificou o código, omitindo informações desnecessárias para a definição da máquina. Por exemplo, na Figura 2.1, linha 3, para definir o primeiro disco, temos que informar todos os parâmetros do método, mesmo que estes não sejam necessários para a definição. O mesmo não ocorre na DSL, Figura 2.2, linhas 6 e 7, que apenas define os atributos relevantes para a definição da máquina, melhorando o grau de

Figura 2.1: Computador abstrato utilizando a linguagem Java.

```
1    public Computer createComputer(){
2        Processor p = new Processor(2, 2500, Processor.Type.
             i386);
3        Disk d1 = new Disk(150, Disk.UNKNOWN_SPEED, null);
4        Disk d2 = new Disk(75, 7200, Disk.Interface.SATA);
5        return new Computer(p, d1, d2);
6    }
```

Fonte: Fowler (2010)

Figura 2.2: Computador abstrato implementado sob a forma de uma DSL.

```
1    computer()
2      .processor()
3         .cores(2)
4         .speed(2500)
5         .i386()
6      .disk()
7         .size(150)
8      .disk()
9         .size(75)
10        .speed(7200)
11        .sata()
12   .end();
```

Fonte: Fowler (2010)

flexibilidade, manutenção e evolução do código.

Hudak (1998) destaca algumas vantagens em usar DSLs, com o fato de que os programas são geralmente mais fáceis de escrever, analisar, e modificar em comparação com programas equivalentes escritos em linguagens de propósito geral. A quantidade de código que é necessário escrever é reduzida, aumentando a produtividade e decrementando os custos de manutenção.

Para Deursen, Klint and Visser (2000), a adoção de uma aproximação baseada em DSL à engenharia de software envolve oportunidades, mas também riscos. As grandes questões relativas a DSL se encontram na fase da sua concepção, devido ao custo do desenho, implementação e manutenção, assim como a dificuldade no equilíbrio entre as construções especificas de domínio e as de programação genérica da linguagem, e a potencial diminuição da eficiência quando comparada com software escrito em uma linguagem de programação genérica.

2.3 DSL para Modelagem de Ambiente em ABMS

Santos, Nunes and Bazzan (2016) propuseram uma linguagem de modelagem para ABMS fundamentada no princípio da expressividade das DSLs, a qual será referenciada como *DSL4ABMS*. Santos, Nunes and Bazzan (2016) identificaram abstrações recorrentes através de um processo de análise de domínio utilizando simulações baseadas em agentes existentes, que é a base de sua DSL. As abstrações identificadas foram modeladas de modo a formar as sintaxes abstrata e concreta da linguagem.

2.3.1 Sintaxe Abstrata

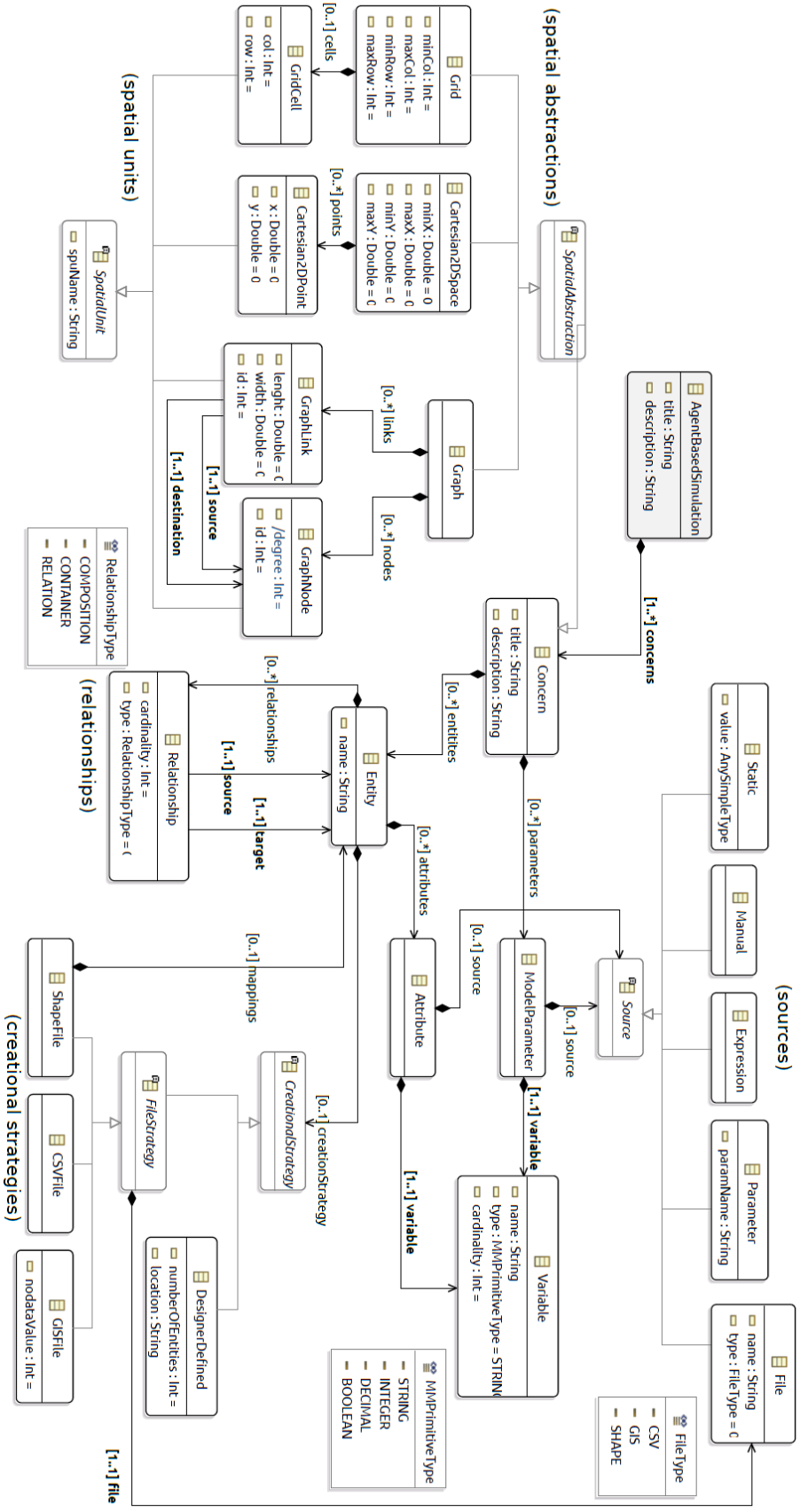
Santos, Nunes and Bazzan (2016) consideraram um conjunto de 18 simulações para definir a sintaxe abstrata da *DSL4ABMS*. Das simulações, foram identificados elementos recorrentes, que descrevem o ambiente, tais como: um ambiente em que todas as entidades situadas são localizadas; parâmetros de simulação; entidades com atributos; e relacionamentos. Na análise dos modelos, foi identificado que os elementos de um modelo podem ser agrupados de acordo com um interesse em particular. Estes elementos são detalhados na Figura 2.3, que mostra, parcialmente, a sintaxe abstrata da *DSL4ABMS*.

O elemento principal da *DSL4ABMS* é o *AgentBasedSimulation*, que agrega todos os outros elementos da especificação. É introduzido a noção de abstração espacial (*Spacial Abstraction*), utilizada para especificar o ambiente em que as outras entidades estão situadas. Uma abstração espacial é a base da simulação, e é composta de unidades espaciais (*SpatialUnit*). Uma unidade espacial é um tipo particular de entidade na qual outros objetos do ambiente podem ser situados. As abstrações espaciais podem ser de três tipos, e cada uma possui uma unidade espacial especifica: grade (*Grid*), composta de células (*GridCell*), plano cartesiano (Cartesian2DSpace) composto de pontos (*Cartesian2DPoint*) e o grafo (*Graph*) composto por arestas e nodos (*GraphLink* e *GraphEdge*).

Para evitar descrições dispersas em partes distintas do modelo, que poderia prejudicar a compreensão e a manutenção, a *DSL4ABMS* agrupa a especificação de parâmetros e entidades em *Concerns*.

Uma entidade (*Entity*) é um objeto relevante na simulação, cuja sua especificação estrutural inclui seu nome e atributos. Relacionamentos (*Relationship*) também podem ser especificados entre entidades. Uma relação pode ser de três tipos: *Composition*, quando uma entidade é composta por outras, e neste caso, o todo é responsável pela criação das

Figura 2.3: Sintaxe Abstrata da *DSL4ABMS*



Fonte: (SANTOS; NUNES; BAZZAN, 2016)

partes; *Container*, quando uma entidade pode conter outras; *Association*, quando há uma relação semântica entre duas entidades.

Os parâmetros de um *Concern* são normalmente utilizados como entrada para a simulação. Os atributos de entidades e parâmetros de *Concerns* correspondem a o mesmo elemento no modelo: *Attribute*. Esses atributos são descritos em termos de nome, tipo, cardinalidades e a origem de seus valores (*Source*). Um tipo pode ser uma entidade ou um tipo primitivo. A origem é usada para abstrair a inicialização do atributo, e esconder detalhes de como seus valores são fornecidos. Pra os parâmetros do *Concern*, são utilizados três tipos de origem: estática, manual e expressão. Já para atributos de entidades: Parâmetro, arquivo CSV, arquivo GIS e arquivo Shape.

A linguagem fornece também estratégias criacionais para configurar como as entidades são criadas e localizadas na abstração espacial. São suportadas quatro estratégias:

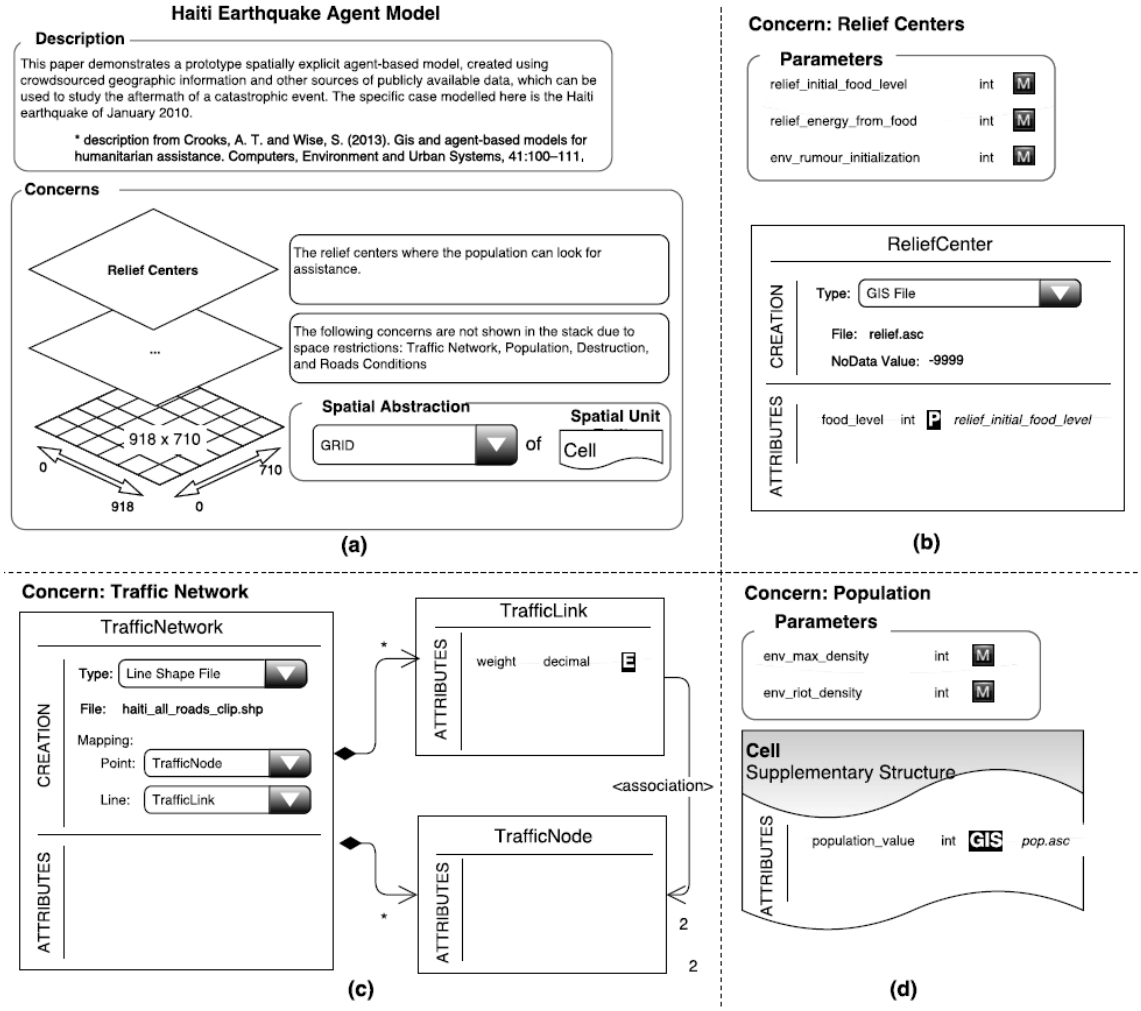
- *Designer Defined*, no qual o designer especifica o número de entidades que são criadas e suas localizações;
- *CSV File*, no qual uma entidade é criada para cada linha do arquivo CSV, e os campos separados por vírgulas estão disponíveis para uso na inicialização de seus atributos;
- *GIS File*, no qual uma entidade é criada para cada valor de dados, e sua localização é a unidade espacial nessa coordenada;
- *Line Shape File*, na qual uma entidade composta e suas entidades contidas são criadas de acordo com o mapeamento de pontos, linhas e polígonos para seus tipos de entidade correspondentes.

2.3.2 Sintaxe Concreta

Santos, Nunes and Bazzan (2016) especificaram símbolos textuais e gráficos para representar elementos da sintaxe abstrata. Por uma questão de escalabilidade do modelo, os *Concerns* de simulação são separados em diagramas diferentes, onde cada diagrama representa um interesse em particular. Além disso, especificar *Concerns* individualmente permite que especialistas se concentrem em conceitos relacionados a sua área. A Figura 2.4 mostra uma visão geral da sintaxe concreta da *DSL4ABMS*.

A Figura 2.4(a) mostra o elemento de visão geral, cujo objetivo é apontar o propósito da simulação e fornecer uma visão ampla da simulação que está sendo modelada.

Figura 2.4: Sintaxe Concreta da *DSL4ABMS*



Fonte: (SANTOS; NUNES; BAZZAN, 2016)

É composto do título e da descrição da simulação, e seus *Concerns*. Os *Concerns* são representados como um losango, composto por um título e uma descrição. Os *Concerns* são representados como uma pilha, enfatizando que elas representam a decomposição da simulação em camadas conceituais.

A camada inferior representa sempre a abstração espacial do ambiente em que as entidades estão situadas. A abstração espacial pode ser uma grade, um espaço cartesiano ou um grafo. Para cada tipo de abstração espacial deve-se informar o nome da unidade espacial associada e algumas propriedades adicionais, como descritos na Tabela 2.1.

As entidades são representadas com um retângulo dividido em três seções: nome da entidade, estratégia de criação e atributos. A Tabela 2.2 contém os tipos de estratégias de criação suportadas pela *DSL4ABMS* e suas propriedades adicionais. Um atributo é representado pelo seu nome, tipo, cardinalidade e origem de seus valores iniciais. A Tabela 2.3 contém os tipos de origens suportadas pela *DSL4ABMS* e suas propriedades particulares.

Tabela 2.1: Tipos de abstração espacial e suas propriedades adicionais

Tipo	***Propriedades***
Grid	*minrow*: indice da menor linha da *grid* ; *maxrow*: indice da maior linha da *grid* ; *mincol*: indice da menor coluna da *grid* ; *maxcol*: indice da maior coluna da *grid* ; *spatialUnitNameOfCell*: nome da entidade que será criada para representar a célula no *grid*;
Cartesian Space	*minx*: valor mínimo da coordenada x no plano cartesiano; *maxx*: maior coordenada x no plano cartesiano; *miny*: menor coordenada y no plano cartesiano; *maxy*: maior coordenada y no plano cartesiano *spatialUnitNameOfPoint*: nome da entidade que será criada para representar o ponto no plano cartesiano;
Graph	*quantityOfNodes*: quantidade de nodos do grafo; *spatialUnitNameOfNode*: nome da entidade que será criada para representar o nodo no grafo; *spatialUnitNameOfLink*: nome da entidade que será criada para representar a aresta no grafo;

Tabela 2.2: Estratégias de criação e suas propriedades adicionais

Tipo	***Propriedades***
Designer defined	*quantity*: quantidade de entidades a criar; *location*: onde cada entidade será posicionada;
CSV File	*file name*: nome/caminho do arquivo CSV;
GIS File	*file name*: nome/caminho do arquivo GIS;
Line Shape File	*file name*: nome/caminho do arquivo SHAPE; *point mapping*: uma entidade criada para cada ponto; *line mapping*: uma entidade criada para cada linha; *polygon mapping*: uma entidade criada para cada polígono;

Concerns podem especificar estruturas suplementares compostas de atributos adicionais, e essas estruturas estão embutidas na entidade da unidade espacial, que é herdada da abstração espacial (grade, espaço carteiro ou grafo) escolhida no elemento de visão geral. Uma estrutura suplementar é representada por um retângulo com a linha de fundo curvada para dar a ideia de complemento. A Figura 2.4(d) apresenta um exemplo de tal estrutura suplementar, na qual um atributo adicional é definido para a unidade espacial da célula.

Além de entidades e estruturas suplementares, os *Concerns* podem conter parâmetros, que são agrupados no *Concern* em um único elemento, chamado *Parameters*, como visto na Figura 2.4(b). Este elemento é composto de parâmetros, que seguem a mesma notação de atributos em uma entidade.

A Figura 2.4(c) mostra que os relacionamentos entre entidades são representados

Tabela 2.3: Origens e suas propriedades adicionais

Símbolo	***Descrição***	***Propriedades***
P	Parâmetro	*parameter*: um dos parâmetros do concern;
E	Expressão	*expression*: uma expressão;
V	Valor estático	*value*: valor direto que será atribuído;
MI	Entrada manual livre	
MYN	Entrada manual YES/NO	
MB	Entrada manual limitada	*min*: valor mínimo aceitável da entrada; *max*: valor máximo aceitável da entrada; *step*: valor do incremento;
MLV	Entrada manual de uma lista valores	*values*: lista de valores para entrada;
FCSV	Arquivo CSV	*file*: nome/caminho do arquivo CSV;
FGIS	Arquivo GIS	*file*: nome/caminho do arquivo GIS;
FCHP	Arquivo SHAPE	*file*: nome/caminho do arquivo SHAPE;

por linhas. Uma relação de composição é representada por uma linha terminada com um losango cheio. Uma relação de associação é representada por uma linha terminada com uma seta simples. Uma relação de contêiner é representada por uma linha terminada com uma seta cheia.

2.4 Considerações Finais

Neste capítulo foram discutidos os principais conceitos sobre ABMS e DSL. Discutimos o conceito de modelagem, simulação, agente, ambiente e interações entre agentes para ABMS. Vimos que as DSLs permitem que as soluções sejam expressas no idioma e no nível de abstração do domínio e, consequentemente, especialistas no domínio conseguem entender, validar, modificar e mesmo criar as mesmas. Discutimos as desvantagens no uso de DSLs, tais como custo para projetá-la, implementá-la e mantê-la, e a perda de eficiência quando comparada com o código escrito em GPL.

Descrevemos a *DSL4ABMS* proposta por Santos, Nunes and Bazzan (2016), explorando os detalhes da sua sintaxe abstrata e concreta. A criação de uma ferramenta que implemente tal sintaxe concreta é o objetivo deste trabalho. No capítulo seguinte, apresentaremos essa ferramenta, que possibilita a construção de ABMS em alto nível, sem a necessidade do usuário se preocupar com questões técnicas da ferramenta de simulação.

3 ABM TOOL

A ABM Tool é uma implementação da *DSL4ABMS* que apresentamos na Seção 2.3, focada no domínio ABMS, mais especificamente no aspecto ambiente simulado. Construída como um *plug-in* para a IDE (*Integrated Development Environment*) Eclipse que permite organizar e projetar modelos baseados em agentes de acordo com especificação da *DSL4ABMS*. A ferramenta utiliza a interface de edição de diagramas da IDE Eclipse, disponibilizando ao usuário uma gama de recursos, que normalmente são encontradas em IDEs de desenvolvimento de softwares, editores de texto e editores de imagem. Recursos, como por exemplo, uma aba que contêm uma árvore que mostra os arquivos e diretórios do projeto, que permite que o usuário crie, exclua, renomeie, abra, mova, arquivos do projeto etc. O ambiente também disponibiliza uma paleta de elementos que podem ser inseridos no diagrama. Outros recursos nativos da plataforma, como a exportação do diagrama em diferentes formatos, ações de copiar, colar, refazer e desfazer, impressão diagramas e configuração de preferencias de visualização também são disponibilizadas ao usuário.

No restante do capítulo serão apresentadas as principais funcionalidades do ABM Tool, sua implementação, arquitetura e tecnologias utilizadas na implementação.

3.1 Funcionalidades

As principais funcionalidades da ferramenta compreendem a criação de diagramas para especificar modelos de agentes conforme a sintaxe concreta da *DSL4ABMS*, bem como a manipulação dos elementos destes diagramas. A seguir, apresentamos os principais recursos da ferramenta com um nível maior de detalhes.

- *Inclusão/Remoção de modelos*: Usuários podem adicionar e remover modelos em um projeto. Na ferramenta, cada diagrama no projeto representa um modelo. Ao adicionar um novo diagrama, a ferramenta associa o diagrama a um elemento de visão geral, que representa graficamente o modelo. Para remover o modelo, basta excluir o diagrama do projeto, que seu modelo também será removido, assim como todos os objetos associados a ele.
- *Edição do modelo*: Usuários podem editar propriedades do modelo diretamente no elemento de visão geral. O usuário pode definir o título do modelo, sua descrição

e especificar as propriedades da abstração espacial. É possível também, editar o título e a descrição de cada extitConcern individualmente.

- *Inclusão/Remoção de Concerns*: Usuários podem adicionar e remover *Concerns* ao modelo. Para adicionar um *Concern*, é necessário que o diagrama associado ao modelo esteja ativo no editor. A ferramenta, internamente, cria um subdiagrama, e o associa ao *Concern* criado. O subdiagrama é utilizado para apresentar uma visão parcial do modelo, focando apenas no detalhamento do *Concern*. A ferramenta armazena as definições do subdiagrama no diagrama principal que está associado ao modelo. Só é permitido remover o *Concern* que não contiver estruturas suplementares associadas. Caso possua, para remoção, é necessário primeiro excluir as estruturas suplementares do *Concern*.
- *Detalhamento de Concerns*: Usuários podem especificar detalhadamente um *Concern* do modelo. Cada *Concern* pode conter elementos como entidades, parâmetros, estruturas suplementares e relacionamentos entre entidades. Através de uma visão parcial do modelo, em um diagrama diferente, o usuário poderá criar, modificar e remover estes elementos.
- *Inclusão/Remoção de entidades*: Os usuários podem adicionar e remover entidades a um *Concern*. A Adição de entidades só é permitida nos subdiagramas associados aos *Concerns*. Cada entidade deve possuir um nome único em cada *Concern*. Podem haver entidades com mesmo nome, desde que, em *Concerns* distintos. A remoção de entidade implica na exclusão de todos os relacionamentos que a tem como origem ou destino. A ferramenta se encarrega de manter a consistência das entidades que eventualmente a referenciam.
- *Edição de propriedades de entidades*: Usuários podem editar propriedades das entidades, tais como nome e estratégia de criação. O usuário pode optar por quatro estratégias de criação suportadas, *Designer defined*, *CSV File*, *GIS File* e *Line Shape File*. Cada um deles exige que o usuário especifique suas propriedades particulares.
- *Inserção/Remoção de atributos*: Usuários podem inserir e remover atributos nas entidades e nas estruturas suplementares. O nome do atributo é único em cada elemento. O usuário é livre para adicionar a quantidade de atributos de desejar. O usuário remove um atributo por vez, não sendo permitida a exclusão de múltiplos atributos de uma única ação.
- *Edição de propriedades de atributos*: Usuários podem editar atributos de entidades e estruturas suplementares diretamente no elemento. Eles podem especificar, por

exemplo, o nome do atributo, seu tipo e sua forma de inicialização. Dependendo da forma de inicialização, são exigidos que o usuário defina suas propriedades particulares.

- *Inclusão/Remoção de estruturas suplementares*: Os usuários podem adicionar e remover estruturas suplementares a um *Concern*. A inclusão de estruturas suplementares só é permitida nos subdiagramas associados aos *Concerns*, e restrita às definições de abstração espacial do modelo.
- *Edição de propriedades de estruturas suplementares*: Usuários podem editar propriedades das estruturas suplementares. No momento, a única propriedade disponível é o nome da estrutura, que também pode ser modificada alterando as definições de abstração espacial do modelo a qual está diretamente associada.
- *Inclusão/Remoção de parâmetros em Concerns*: Usuários podem inserir e remover parâmetros nos *Concerns*. O nome do parâmetro é único em cada *Concern*, e a inclusão de parâmetros só é permitida nos subdiagramas associados aos *Concerns*. O usuário é livre para adicionar a quantidade de parâmetros que desejar ao *Concern*. O usuário remove um parâmetro por vez, não sendo permitida a exclusão de múltiplos parâmetros de uma única ação. Quando não há parâmetros associados ao *Concern*, o elemento *Parameters* não é apresentado no subdiagrama.
- *Edição de propriedades de parâmetros*: Usuários podem editar propriedades dos parâmetros do *Concern* diretamente no elemento *Parameters*. Eles podem especificar as mesmas propriedades que estão disponíveis nos atributos de entidades, tais como nome, tipo e forma inicialização.
- *Inclusão/Remoção de relacionamento entre entidades*: Usuários podem adicionar e remover relacionamentos entre duas entidades. O usuário pode também especificar se relação é bidirecional e qual sua cardinalidade. Quando um relacionamento é removido, a ferramenta atualiza todas as entidades envolvidas, removendo eventuais referencias que existam.
- *Inclusão de relacionamento entre entidades em outros Concerns*: Usuários podem adicionar relacionamentos entre duas entidades de *Concerns* diferentes. Neste caso, é criado um novo elemento no subdiagrama que contém a entidade origem da relação. Este elemento representa a entidade contida no *Concern* detalhado em outro subdiagrama.
- *Exportar dados do modelo*: Usuários podem exportar os dados do modelo em um

arquivo no formato XML (*eXtensible Markup Language*). O usuário pode definir o nome do arquivo e escolher o local onde o arquivo será gravado.

3.2 Arquitetura e Tecnologias Utilizadas

A arquitetura da ferramenta está disposta seguindo as práticas recomendadas para o desenvolvimento de complementos para a IDE Eclipse. Por este motivo, nos restringimos a utilização tecnologias homologadas pela Eclipse Foundation, organização mantenedor da IDE Eclipse. Tais tecnologias são de código aberto e estão publicadas no repositório de complementos da IDE Eclipse[1].

3.2.1 Arquitetura

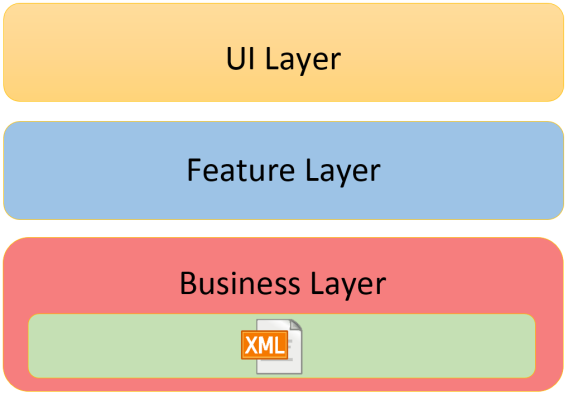
A arquitetura da ferramenta está dividida em três camadas, como ilustrado na Figura 3.1. A primeira camada representa a interface de interação com o usuário. A segunda é camada que implementa as funcionalidades que provem os recursos para renderização pela camada de apresentação. A terceira é camada de negócio, que implementa a *DSL4ABMS*, responsável pelo armazenamento e persistência dos dados do modelo.

Como consequência da opção pelo desenvolvimento da ferramenta como complemento para a IDE Eclipse, a camada de apresentação (*UI Layer*) herda muitas de suas características, tais como manipulação de arquivos, configuração de ambiente, recursos de acessibilidades como zoom e tamanho de fontes, entre outros. Por um lado, as características herdadas já estão testadas pela comunidade, podendo-se focar no desenvolvimento das novas funcionalidades requeridas. Por outro lado, é demandado um esforço significativo para entendimento do código de integração com a IDE e sua extensão para incorporar as novas funcionalidades.

A *Feature Layer* é composta por um conjunto de classes que estendem as funcionalidades nativas da IDE Eclipse, possibilitando a criação de funcionalidades customizadas. Com as customizações é possível definir, por exemplo, a forma do objeto a ser renderizado, regras de interação no diagrama, comunicação com recursos da IDE Eclipse, manipulação de arquivos, entre outros. Nesta camada, é possível adicionar objetos na paleta de objetos da IDE Eclipse, manipular os arquivos do projeto, manipular diagramas,

[1] http://marketplace.eclipse.org

Figura 3.1: Arquitetura da ABM Tool



adicionar e modificar pictogramas dentro de um diagrama.

A última camada é a de negócio (*Business Layer*), no caso deste trabalho, é a implementação da *DSL4ABMS*. Esta camada é responsável pela inicialização dos objetos, persistência e consistência dos dados do modelo. Essa persistência é feita em um arquivo XML, que além dos dados do modelo, também contém o posicionamento e as dimensões dos objetos no diagrama. A camada de negócio é fundamental para a ferramenta, tendo em vista que as camadas UI e Feature não persistem os dados, apenas provem uma interface de visualização e regras para interação.

Na ABM Tool, a *UI Layer* é o diagrama Eclipse, com as abas, paleta, menus e árvores de diretório. A *Feature Layer* são as funcionalidades adicionadas e estendidas pelo *framwork Graphiti*. A *Business Layer* é o conjunto de classe, criadas a partir do modelo Ecore, que definem a *DSL4ABMS*.

3.2.2 Tecnologias Utilizadas

Para a implementação da ferramenta, optamos pela utilização do *Eclipse Modeling Framework*[2], que permite a especificação de metamodelos e geração de código. Também utilizamos o *Graphiti Framework*[3], uma coleção de classes que permitem a criação de diagramas de alta qualidade com facilidade, além de prover uma completa integração com a IDE Eclipse. Os dois *frameworks* também possuem um guia de referência para desenvolvedores, com exemplos e tutoriais. Como consequência da utilização da *Graphiti Framework* adotamos o Java (GOSLING, 2000) como linguagem de desenvolvimento da ferramenta.

[2]http://www.eclipse.org/modeling/emf/
[3]http://eclipse.org/graphiti/

3.2.2.1 Eclipse Modeling Framework

O Framework de Modelagem do Eclipse ou *Eclipse Modeling Framework (EMF)* (ECLIPSE, 2016) fornece uma estrutura para geração de modelos visuais e código para aplicações Eclipse. Para tanto, o EMF requer a especificação de um metamodelo. Um modelo criado visualmente a partir do EMF é, portanto, uma instância de um metamodelo. Para especificação de metamodelos, o EMF disponibiliza um meta-metamodelo chamado Ecore [4].

A partir de um metamodelo Ecore, um conjunto de classes Java para o modelo e um editor básico baseado em árvore podem ser gerados. As classes geradas fornecem suporte básico para criar, modificar e excluir elementos de modelo e as operações de persistência, como carregar e salvar. As relações entre classes de modelo EMF são tratadas por listas EMF especiais, estendendo as classes de lista Java. Além disso, os modelos EMF podem ser usados como modelos subjacentes em novos plug-ins de aplicação.

3.2.2.2 Graphiti

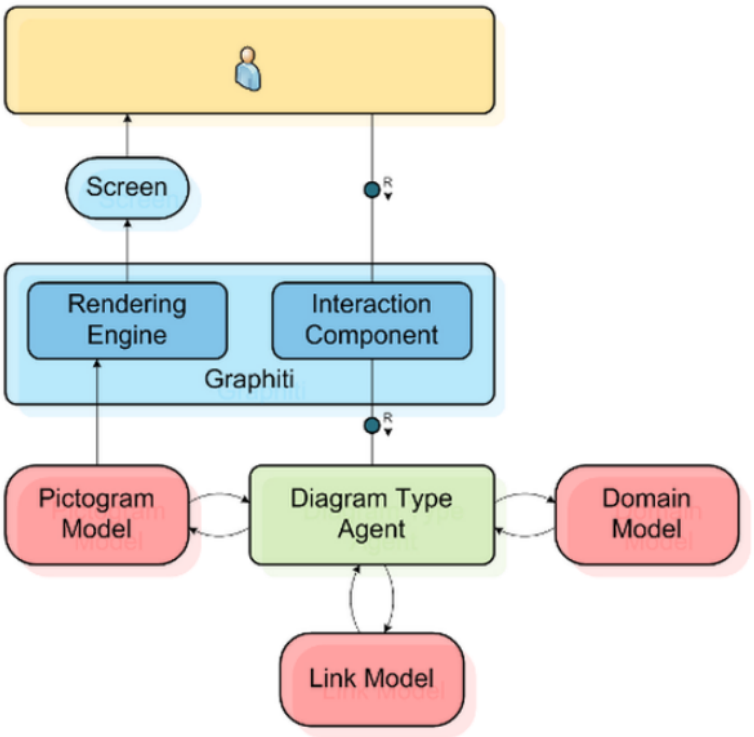
O *Graphiti Framework* (ECLIPSE, 2016) é uma biblioteca que permite o desenvolvimento de editores gráficos de uma forma mais estruturada e simples. Editores gráficos criados a partir do Graphiti já contam com barras laterais para os tipos de objetos que se podem utilizar, bem como uma grade para melhor organização de elementos gráficos. Graphiti requer que o metamodelo da linguagem gráfica esteja definido em Ecore. A estrutura do Graphiti pode ser vista na Figura 3.2.

O objetivo do *Rendering Engine* é exibir as representações gráficas atuais na tela. *Rendering Engine* e *Interaction Component* juntos formam o *Graphiti Runtime*, que fornece ao usuário um editor gráfico que permite a interação e manipulação dos objetos renderizados. Estes dois componentes são providos pelo próprio Graphiti, e portanto não exigem esforço de desenvolvimento.

O bloco *Pictogram Model* contém a representação gráfica dos elementos do modelo, enquanto o bloco *Domain Model* contém o modelo gerado com base no metamodelo Ecore. O bloco *Link Model* é responsável por fazer a integração entre os elementos gráficos e suas respectivas instâncias criadas a partir do metamodelo. É possível diversas representações visuais para o mesmo elemento do metamodelo. O bloco *Diagram Type Agent - (DTA)* contém toda a lógica para responder às ações do utilizador, sendo necessá-

[4] http://www.eclipse.org/ecoretools/

Figura 3.2: Arquitetura do Framework Graphiti



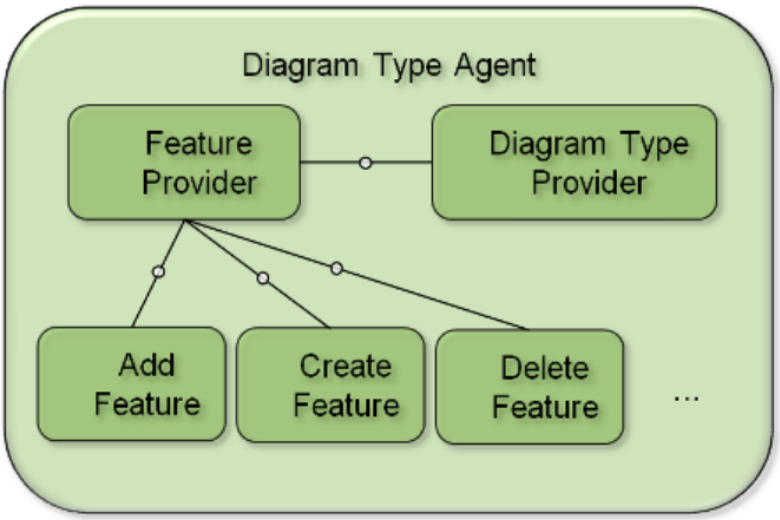
Fonte: Eclipse (2016)

rio construí-lo. A partir do DTA, o desenvolvedor pode criar e eliminar elementos gráficos e elementos do modelo, mover e alterar a forma de elementos gráficos e atualizá-los. A Figura 3.3 representa a estrutura do DTA.

O DTA contém a classe *Diagram Type Provider*, a qual define o diagrama que se está a desenvolver. O DTA é formado por blocos de classes que constituem as funcionalidades a que o utilizador pode recorrer. É necessário criar funcionalidades para cada objeto que se pretende representar (ex.: um Step para ter a funcionalidade de adicionar cria a classe AddStep que irá pertencer ao bloco *Add Feature* e para criar o objeto precisa de uma classe CreateStep que irá pertencer ao bloco *Create Feature*). A classe *Feature Provider* é responsável por identificar qual o bloco de funcionalidades pretendido pelo editor (devido ao *context* que contextualiza a *Feature Provider*) e reencaminha a informação para o respectivo bloco de *Features*.

O DTA é o orquestrador do diagrama. Para cada novo tipo de diagrama, uma classe DTA deve ser implementada. Ele contém a lógica para integrar o *Domain Model* com o *Pictogram Model* e Link Model. Um DTA é composto pelos elementos apresentados na Figura 3.3. O *Diagram Type Provider* é a classe responsável por disponibilizar o tipo de diagrama no IDE Eclipse. A classe *Feature Provider* é responsável por prover as funcionalidades disponibilizados pelo diagrama, tais como adicionar ou remover elementos.

Figura 3.3: Estrutura do Diagram Type Agent



Fonte: Eclipse (2016)

Para cada funcionalidade desejada, deve-se criar uma classe que irá provê-la, estendendo as classes disponíveis no Graphiti, tais como *Add Feature*, *Delete Feature*.

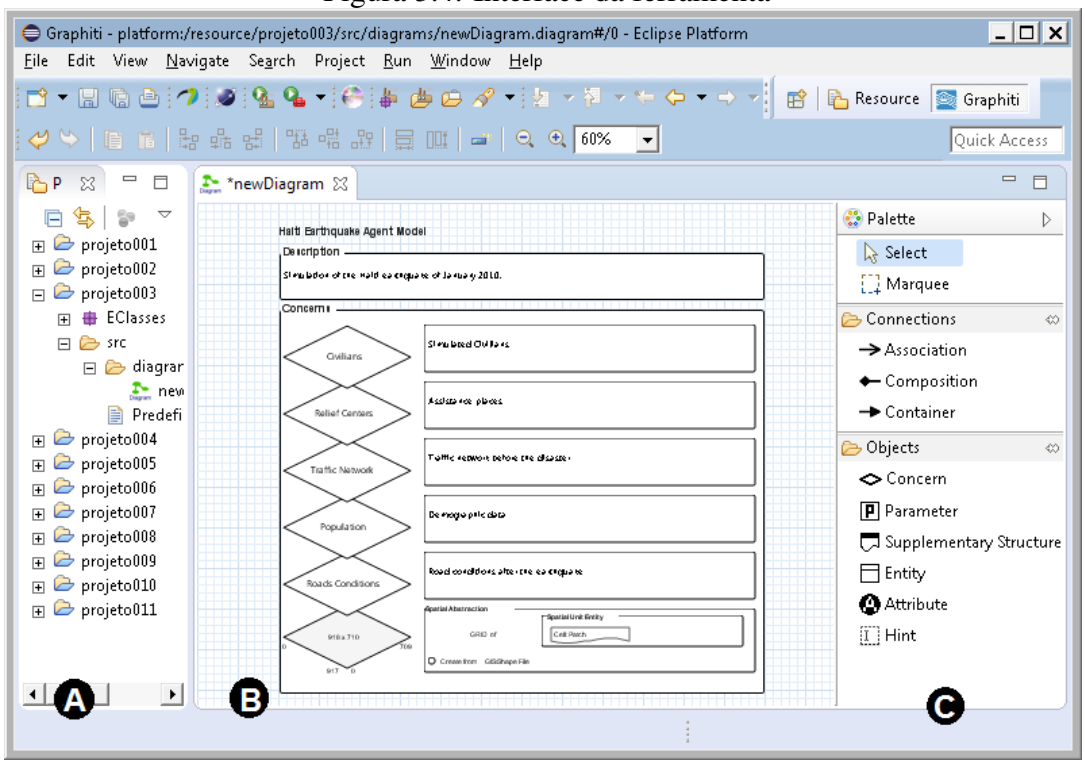
3.3 Apresentação da Ferramenta

A interface da ferramenta é dividida em três componentes,mostrados na Figura 3.4: (A) o explorador do projeto, (B) o editor de diagrama e (C) a paleta. No explorador do projeto, o usuário pode criar, remover, renomear, copiar e organizar em diretórios arquivos de diagramas. O editor de diagrama permite que o usuário manipule os elementos do diagrama, como por exemplo, posicionar um elemento em uma determinada posição, adicionar e remover elementos do próprio diagrama. Através do editor de diagrama, o usuário pode editar diretamente as propriedades dos elementos, desde que o elemento permita a edição direta. Na paleta estão todos os elementos suportados pelo diagrama. O usuário escolhe qualquer elemento da paleta e pode arrastar para o dentro do editor de diagrama para renderização.

Os exemplos apresentados nesta seção modelam a simulação do terremoto no Haiti (CROOKS; WISE, 2013), que investiga como as pessoas reagem à distribuição de alimentos, e como os boatos relativos à sua disponibilidade se propagam através da população. A simulação replica as condições do terreno após o terremoto de 2010 no Haiti, e inclui a modelagem de civis, que podem se mover e propagar rumores, e os centros de comunitários podem distribuir alimentos.

Ainda na Figura 3.4 podemos visualizar, o *elemento de visão geral*. Ele é adicio-

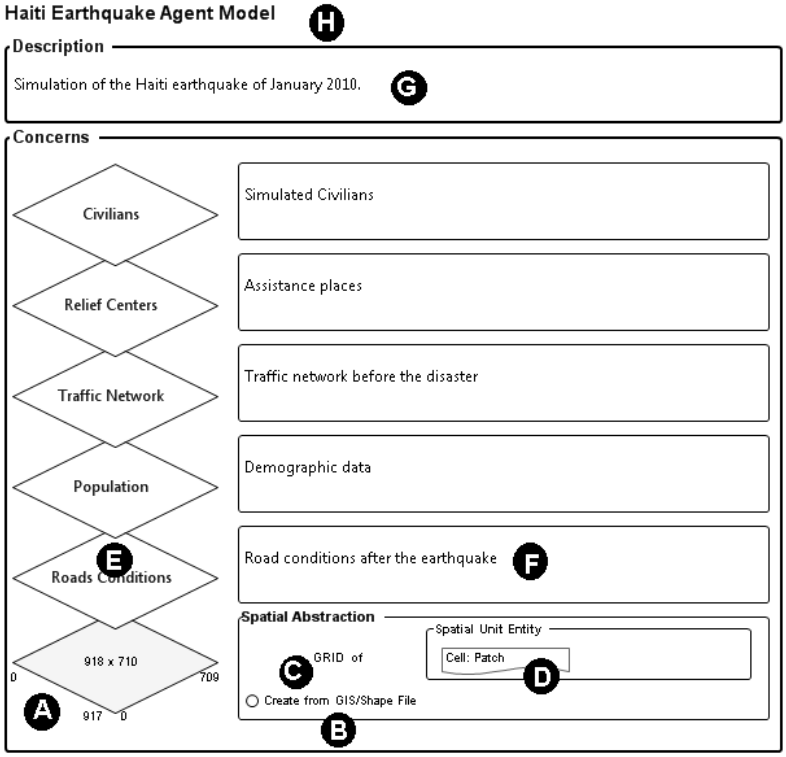
Figura 3.4: Interface da ferramenta



nado automaticamente no diagrama no quando um novo arquivo é adicionado ao projeto. A Figura 3.5 mostra em detalhes todos os elementos que o compõe. No topo, temos o nome do modelo (H), seguido por uma descrição (G) sucinta do modelo. Para modificar o nome ou a descrição do modelo, o usuário deve dar um clique com o botão esquerdo do mouse sobre o texto da descrição. Um campo de edição será aberto, permitindo ao usuário digitar um texto livre. Logo abaixo da descrição do modelo temos a seção *Concerns* ou *Concerns*, que contém a representação dos *Concerns* que compõe o modelo. Para cada *Concern*, é possível definir o título (E) e sua descrição (F), editáveis da mesma forma que o título e descrição do modelo. O último *Concern* desta seção define as propriedades da abstração espacial do modelo. Nela é possível definir e alterar o tipo de abstração espacial, propriedades e os nomes das unidades espaciais.

Para alteração do tipo de abstração espacial, o usuário deve dar um clique com o botão esquerdo do mouse sobre o tipo de abstração espacial (C) e selecionar um dos tipos disponíveis. Após selecionado, a ferramenta efetua a atualização do modelo e disponibiliza as propriedades do tipo para edição pelo usuário. A edição das propriedades segue o mesmo padrão já descrito anteriormente. Além dos parâmetros descritos na Tabela 2.1, o usuário pode especificar para todos os tipos, se as unidades espaciais serão criadas a partir de uma estratégia de criação por arquivo. Para isso, ele deve marcar a opção "*Create from GIS/Shape File*", selecionar *GIS File* ou *Line Shape File* (B) como estratégia de criação e

Figura 3.5: Elemento de visão geral em detalhes

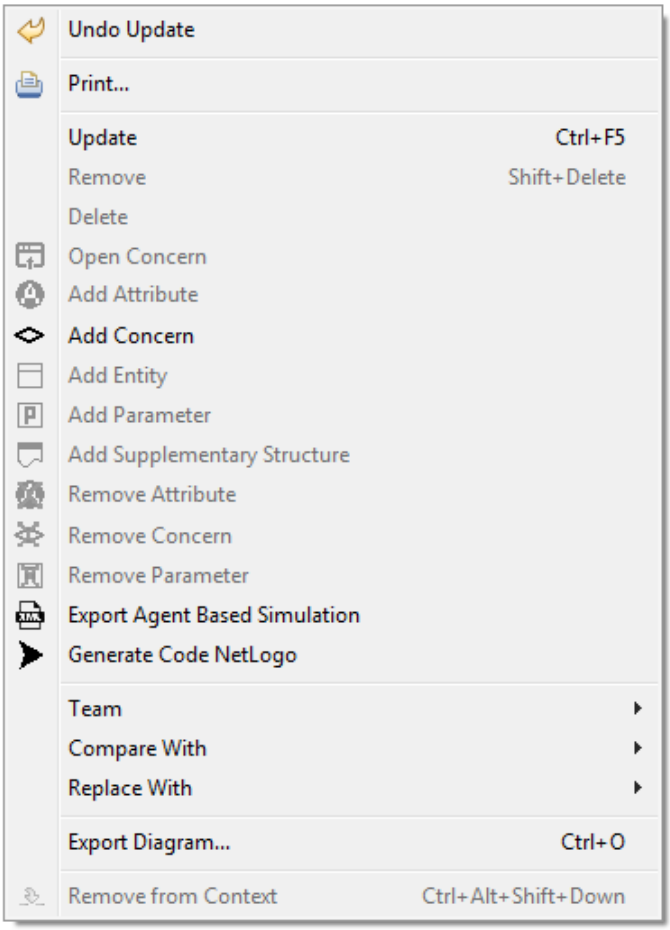


informar o caminho para o arquivo ou clicar no botão *File Choose*, para abrir a caixa de diálogo para escolha de arquivo. É também possível informar o nome da unidade espacial (D) e as dimensões de abstração espacial (A). Por exemplo, na Figura 3.5 (C), está definido que o tipo de abstração é do *Grid*, em (D) o nome da entidade que será criada para representar a célula no *Grid*, e em (A) estão definidos os parâmetros para a abstração espacial do tipo *Grid*, tais como índices da menor e maior linha, índices da maior e menor coluna. Os parâmetros de cada tipo de abstração suportados na ferramenta são aqueles previamente descritos na Tabela 2.1.

Além das funcionalidades de edição direta no elemento de visão geral, o usuário pode ter acesso a outras funcionalidades disponíveis através do menu contextual, como visto na Figura 3.6, acessível acionando o botão direito do mouse sobre o elemento de visão geral. Por exemplo, o usuário pode adicionar ou remover *Concerns* no modelo, exportar o modelo como arquivo XML e salvar o diagrama em diversos formatos, como JPG e PDF.

O usuário também pode adicionar *Concerns* utilizando a paleta de elementos, selecionando o objeto *Concern* e em seguida clicando sobre o elemento de visão geral. A ferramenta cria um subdiagrama para cada *Concern* adicionado, que pode ser aberto dando um duplo clique sobre o *Concern* que se deseja detalhar. Já para remover um *Concern*, o usuário deve clicar com o botão direito do mouse sobre o *Concern* (Figura 3.5-E/F) que se

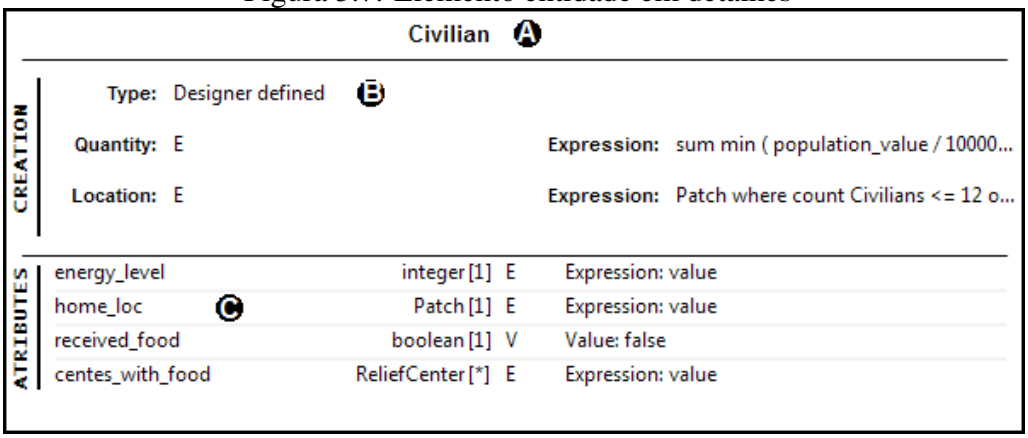
Figura 3.6: Opções no menu contextual



deseja excluir do modelo. Quando a ação exclusão é realizada, a ferramenta remove todos os objetos pertencentes ao *Concern* do modelo, assim como e eventual diagrama onde o *Concern* foi detalhado. Na Figura 3.6, também podemos ver que alguns itens estão desabilitados para utilização, isso porque a ferramenta identifica o elemento sob a areá clicada e habilita somente as funcionalidades adequadas a ele. Por exemplo, se o elemento sob a área for um *Concern*, os itens *Open Concern*, *Add Concern* e *Remove Concern* estarão habilitados, enquanto que os demais não.

O segundo elemento a analisarmos mais aprofundadamente é a entidade ilustrada na Figura 3.7. Uma entidade só pode ser adicionada em um subdiagrama associado a um *Concern* através da paleta ou do menu contextual. Pela paleta, o usuário deve selecionar o objeto *Entity* e em seguida clicar em um local vazio dentro do subdiagrama. Já pelo menu contextual, o usuário deve primeiramente escolher uma região sem elementos no subdiagrama, clicar com o botão direito e selecionar *Add Entity*. A remoção de uma entidade também pode ser de duas formas, a primeira selecionando a entidade e pressionando o botão deletar do teclado. A segunda forma é pressionando um botão com ícone lixeira

Figura 3.7: Elemento entidade em detalhes



sobre a entidade. Ambas as formas exibem uma mensagem de advertência ao usuário exigindo que o usuário confirme a ação. Essa funcionalidade evita que o usuário acabe por apagar uma entidade por descuido.

Conforme a sintaxe concreta da *DSL4ABMS*, a representação de uma entidade é dividida em três seções, a primeira contém o nome da entidade; a segunda contém a estratégia de criação da entidade; e a terceira contém a lista de atributos da entidade. O usuário pode editar diretamente o nome da entidade, clicando com o botão esquerdo do mouse em (A). Para segunda seção, o usuário deve optar entre uma das estratégias disponíveis, conforme apresentado previamente na Tabela 2.2, e especificar suas propriedades. Para definir a estratégia, o usuário deve clicar com o botão esquerdo do mouse em (B), selecionar uma das estratégias disponíveis e definir as suas propriedades. Na terceira seção, dos atributos, o usuário pode adicionar, editar e remover atributos da entidade. Para adicionar, o usuário dispõe de duas formas: a primeira é selecionando o objeto *Attribute* na paleta de objetos e em seguida clicando na entidade que se deseja adicionar o atributo. A segunda forma é através do menu contextual, clicando com o botão direito do mouse sobre a entidade e selecionando o item *Add Attribute*.

Um atributo (C) de uma entidade possui vários parâmetros, entre eles o nome, o tipo, e a origem do atributo. O nome é um texto livre sem espaços, já o tipo é uma lista de valores, podendo ser um *integer*, *string*, *decimal*, *boolean* ou uma entidade do mesmo *Concern*. Uma origem é usada para abstrair a inicialização do atributo, escondendo detalhes de como seus valores são fornecidos. Em um atributo, são suportadas dez tipos de origens. As origem suportadas e seus detalhes foram descritos previamente na Tabela 2.3. O usuário deve optar por uma das origens para cada atributo adicionado e especificar as propriedades desta origem. Por exemplo, se o usuário optar por especificar a origem como *parameter*, ele deverá também informar qual parâmetro entre os disponíveis

Figura 3.8: Elemento parâmetros em detalhes

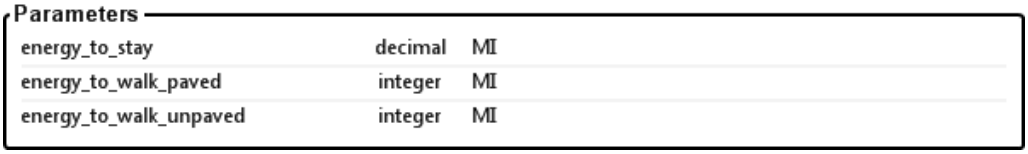
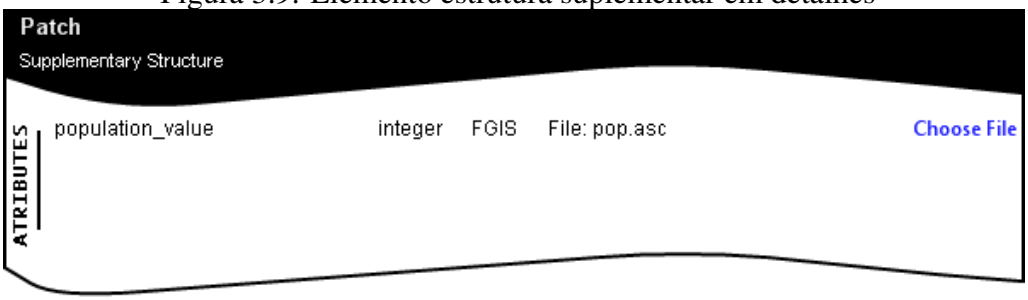


Figura 3.9: Elemento estrutura suplementar em detalhes



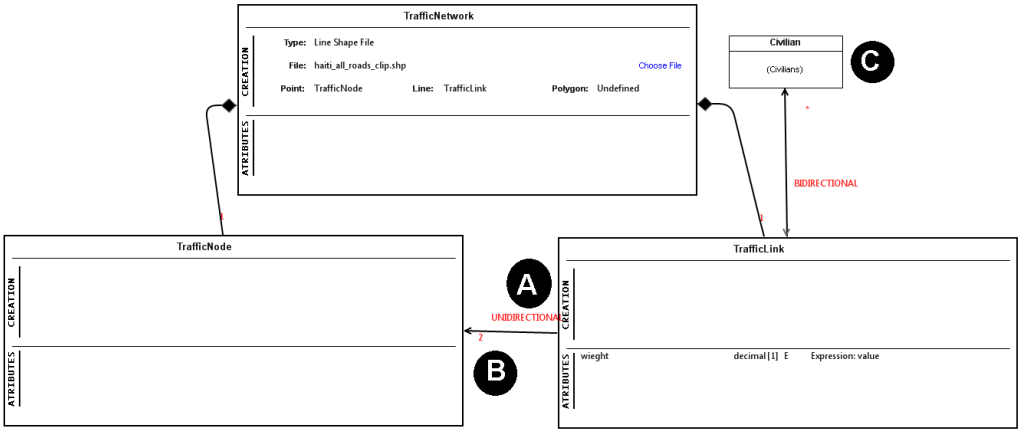
definidos para o *Concern*.

O próximo elemento atualizado é a lista de parâmetros ou *Parameters* do *Concern* ilustrado na Figura 3.8, que agrupa em um único elemento todos os parâmetros definidos para o *Concern*. Ao adicionar um parâmetro ao *Concern*, a ferramenta cria automaticamente o bloco de parâmetros. A ferramenta também remove o bloco parâmetros do subdiagrama se todos os parâmetros do *Concern* forem removidos.

O usuário pode adicionar parâmetros de duas formas, através da paleta de objetos ou pelo menu contextual, semelhante a mostrada anteriormente para adição de atributos em entidades. Pela paleta, o usuário seleciona o objeto *Parameter*, e em seguida arrasta para dentro do subdiagrama ou sobre o elemento parâmetros. Para remover um parâmetro, o usuário deve clicar com o botão direito sobre um dos parâmetros que se deseja remover no elemento *Parameters*, e selecionar o item *Remove Parameter*. É possível também editar as propriedades de um parâmetro, de forma semelhante a edição de um atributo de uma entidade. Podem ser editados o nome, tipo e origem do parâmetro como ocorre nas entidades.

Por fim, a Figura 3.9 apresenta a o componente para modelagem das estruturas suplementares, que especificam as unidades espaciais definidas no elemento de visão geral, no diagrama principal. As estruturas suplementares são divididas em duas seções como ilustrado na Figura 3.9, sendo que a primeira contém o nome da unidade espacial que a estrutura suplementar representa, e segunda contém uma lista com os atributos da estrutura suplementar. As funcionalidades de adicionar, remover e editar atributos de uma estrutura suplementar são semelhantes às funcionalidades de uma entidade, assim como as propriedades dos atributos disponíveis para edição.

Figura 3.10: Exemplo de um relacionamento entre entidades do mesmo *Concern*



A adição de estruturas suplementares no subdiagrama está condicionada ao tipo de abstração espacial definida no elemento de visão geral definido no diagrama principal. Por exemplo, se o usuário define que a abstração espacial é do tipo *Grid*, a ferramenta permitirá que o usuário defina apenas uma estrutura suplementar por subdiagrama. No entanto, se a opção for pela abstração espacial do tipo *Graph*, o usuário poderá definir até duas estruturas suplementares, uma para especificar o vértice (*Nodo*) e outro para a aresta (*Link*).

Além dos elementos entidade, propriedades e estruturas suplementares que definimos acima, é possível adicionar relacionamentos entre entidades de mesmo *Concern*, como ilustrado na Figura 3.10. Os relacionamentos são disponibilizados na paleta de objetos, sendo possível optar por três tipos de relacionamento: *Composition* quando uma entidade é composta de outras; *Container* quando uma entidade pode conter outras e *Association* quando uma entidade está também abstraindo um relacionamento entre entidades. Para adicionar um relacionamento entre duas entidades, o usuário deve escolher uma das relações na paleta, selecionar a entidade de origem do relacionamento, e depois, a entidade destino do relacionamento.

A ferramenta possibilita especificar propriedades de cardinalidade e direção de uma relação. A direção de uma relação pode ser direcional ou bidirecional. Para alterar as propriedades da relação, o usuário deve selecionar o relacionamento e clicar sobre a propriedade que deseja modificar. Por exemplo, se a propriedade for direção, ao clicar sobre o texto (A), será disponibilizado ao usuário as opções *UNIDIRECTIONAL* e *BIDIRECTIONAL*, dos quais ele deve optar por um. Na Figura 3.10 podemos visualizar quatro relações, uma relacionamento do tipo *Association* que associa a entidade *TrafficNode* a entidade *TrafficLink*, e dois relacionamentos do tipo *Composition* entre as entidades *Traf-*

ficNetwork e *TrafficLink*, e *TrafficNetwork* e *TrafficNode*. É possivel modificar a cardinalidade de uma relação, selecionando a relação, e em seguida, clicando com botão esquerdo do mouse sobre a cardinalidade (B).

Outra possibilidade é criar uma relação entre entidades de *Concerns* distintos. Para tal, o usuário deve selecionar a entidade origem da relação, clicar com o botão direito do mouse sobre a entidade, e em seguida selecionar o item *Add Relationship with*, escolher um *Concern* e uma entidade para relacionar. A ferramenta adicionará automaticamente um novo elemento no subdiagrama que contem a entidade origem. A Figura 3.10(C) mostra um relacionamento entre as entidades *TrafficLink* e *Civilian* dos *Concerns Traffic Network* e *Civilians* respectivamente.

3.4 Limitações da Ferramenta

A ABM Tool possui algumas limitações, a maioria relacionada ao designer dos elementos. Por exemplo, a ferramenta não possibilita o redimensionamento dos elementos dentro do diagrama. Também não é possível selecionar isoladamente um atributo dentro de uma entidade. Por causa disso, o usuário pode não identificar, com precisão, qual atributo irá manipular. A edição dos atributos exige do usuário múltiplos cliques, o que pode causar incômodo durante sua utilização.

Como a ferramenta trabalha como multivisões do mesmo arquivo, foi implementado um mecanismo que salva o diagrama automaticamente, toda a vez que o usuário troca de diagrama, mantendo a consistência do modelo para todos as multivisões. No entanto, o usuário pode desejar navegar entre os diagramas sem salvar suas modificações, o que não será possível, porém, caso seja necessário voltar a um estágio anterior, ele pode desafazer operações.

3.5 Considerações Finais

No desenvolvimento da ferramenta, procurou-se seguir à risca a sintaxe concreta da *DSL4ABMS*, embora alguns detalhes nos omitimos de implementar, por serem questões estéticas da sintaxe concreta que não afetam a usabilidade. Por exemplo, na definição de prioridades dos atributos da entidade, a ferramenta mostra a cardinalidade independente do tipo, mas na sintaxe concreta original da *DSL4ABMS* a cardinalidade é definida apenas

quando o atributo é do tipo complexo. Na especificação das propriedades de atributos e parâmetros, Santos, Nunes and Bazzan (2016) representam a origem de um atributo ou parâmetro como um símbolo, mas em nossa implementação, definimos como siglas, o que facilitou a implementação e renderização pela ferramenta. Buscou-se explorar ao máximo os recursos disponibilizados pela IDE Eclipse, padronizando a forma como os usuários interagem com a ferramenta, seguindo a proposta de outras ferramentas de modelagem, como por exemplo o Bizagi (2016).

Uma vantagem de ser uma ferramenta projetada para atuar como um complemento para o Eclipse é a reutilização de recursos disponíveis no IDE, tais como criar e salvar projetos, imprimir e exportar o diagrama para outros formatos. O usuário também conta com configurações de acessibilidade, como zoom no diagrama e alteração no tamanho e tipo das fontes da ferramenta. Funções de desfazer e refazer também são nativas da IDE, e estão habilitadas para algumas funcionalidades.

Neste capítulo, listamos as funcionalidades que a ferramenta implementa para utilização da sintaxe concreta da *DSL4ABMS* e as tecnologias que foram utilizadas no desenvolvimento da ferramenta, assim como sua arquitetura. Também apresentamos a implementação da ABM Tool, descrevemos suas principais características, as opções de interação do usuário, e regras de uso. No próximo capítulo, descreveremos a avaliação feita com usuário e os resultados obtidos.

4 AVALIAÇÃO

Este capítulo apresenta a avaliação realizada por alunos de graduação e pós-graduação do Instituto de Informática da UFRGS, com o objetivo de avaliar usabilidade da ferramenta ABM Tool, a partir da perspectiva de seus utilizadores. Três questões de pesquisa foram derivadas deste objetivo: (i) a ferramenta é *útil* para modelar ABMS, que torna seus utilizadores mais eficazes e produtivos? (ii) a ferramenta é *intuitiva*, de tal forma que seus utilizadores consigam especificar modelos sem a necessidade de orientações sobre as funcionalidades da ferramenta? e (iii) a ferramenta é *simples*, de tal forma que os seus utilizadores rapidamente consigam explorar o sistema e realizar suas tarefas?

A Seção 4.1 descreve a metodologia adotada, enquanto que a Seção 4.2 analisa os resultados obtidos através da avaliação.

4.1 Metodologia

A avaliação ocorreu em um ambiente controlado, em um laboratório do Instituto de Informática da UFRGS. Para cada participante, foi fornecido um modelo impresso da simulação do terremoto no Haiti (CROOKS; WISE, 2013), que replica as condições do terreno após o terremoto de 2010 no Haiti, e inclui a modelagem de civis, que podem se mover e propagar rumores, e os centros de comunitários podem distribuir alimentos. Além do modelo, foi fornecido um cartão referência da *DSL4ABMS* e um computador com a ferramenta já inicializada. O cartão referência da *DSL4ABMS* serve para dar uma visão geral dos elementos da linguagem e sua sintaxe concreta. Nenhuma informação adicional foi dada aos participantes sobre as funcionalidades da ferramenta. Também não foi permitido acesso a internet ou a fontes de informação durante a realização da avaliação.

Antes do inicio da avaliação, cada participante respondeu um questionário pré-avaliação, com o intuito de traçar seu perfil, quantificando os seus conhecimentos em ferramentas de modelagem para engenharia de software e em ABMS através de uma escala Likert (1932) de 1 a 7. Também foi solicitado que o voluntário indicasse se havia participado de um experimento anterior realizado para avaliar a *DSL4ABMS*, que avaliou a sua facilidade de compreensão. Estes participantes que participaram do experimento poderiam ter mais facilidade na utilização da ferramenta, pois já conhecem a linguagem e a sintaxe. A Tabela 4.1 mostra as questões contidas no questionário de pré-avaliação.

Tabela 4.1: Perguntas do questionário pré-avaliação

#	***Questão***
Q1	Nome completo do avaliador
Q2	Qual seu nível de experiência em ferramentas de modelagem visual, por exemplo para modelagem UML, BPM, Banco de Dados, etc?
Q3	Qual seu nível de conhecimento em ABMS (Agent-based Modeling and Simulation)?
Q4	Você participou do experimento realizado por Fernando Santos, que avaliou a facilidade de compreensão de sua Domain-specific Language para ABMS em relação ao NetLogo?

Após responder o questionário de pré-avaliação, foi dado a cada participante instruções para realização da tarefa. Foi explicado, que a tarefa constituía em replicar o modelo Haiti impresso, utilizando a ferramenta ABM Tool, tal que a representação gráfica do modelo obtida através da ferramenta seja equivalente a o modelo impresso. Foi fixado um tempo limite para conclusão da tarefa pelo participante de 40 minutos, que é o dobro do tempo que um especialista na ferramenta demorou para realizar a tarefa. Durante a avaliação, os participantes foram assistidos pelo aplicador, especialista na ferramenta, que, quando solicitado, deu diretrizes sobre funcionalidades da ferramenta.

Concluída a tarefa, cada participante respondeu um questionário de pós-avaliação. O objetivo do questionário é coletar as impressões e opiniões do participante sobre da usabilidade da ferramenta ABM Tool, desconsiderando questões da linguagem e da sintaxe. A Tabela 4.2 contém as sentenças do questionário pós-avaliação. As questões do questionário foram baseadas questões USE propostas por Lund (2001) as quais focam na avaliação de importantes domínios de usabilidade, tais como utilidade, facilidade de uso, facilidade de aprendizagem e satisfação. Todas as perguntas foram quantificadas em uma escala *Likert* de 1 a 7, onde 1 significa que o participante descorda fortemente da sentença, e 7 significa que ele concorda plenamente com a sentença. No final do questionário, foi solicitado a cada participante que listasse os aspectos positivos e negativos quanto a usabilidade da ABM Tool. Na seção seguinte, apresentaremos e discutiremos os resultados obtidos pela análise dos dados coletados através dos questionários e por observações durante a avaliação.

Tabela 4.2: Perguntas do questionário pós-avaliação

#	***Questão:* Com relação a ferramenta ABM Tool ...**	***Critério***
Q1	Me ajuda a ser mais eficaz.	Utilidade
Q2	Me ajuda a ser mais produtivo.	Utilidade
Q3	É útil.	Utilidade
Q4	Torna as coisas que eu quero realizar mais fácil de fazer.	Utilidade
Q5	Poupa-me tempo quando o uso.	Utilidade
Q6	Faz tudo o que eu esperava que fizesse.	Utilidade
Q7	É fácil de usar.	Facilidade
Q8	É simples de usar.	Facilidade
Q9	É amigável.	Facilidade
Q10	Requer o menor número possível de etapas para realizar o que eu quero fazer com ele.	Facilidade
Q11	É flexível.	Facilidade
Q12	Eu o uso sem esforço.	Facilidade
Q13	Eu posso usá-lo sem instruções escritas.	Facilidade
Q14	Eu não detectei inconsistências ao usá-lo.	Facilidade
Q15	Usuários ocasionais e regulares gostariam.	Facilidade
Q16	Posso me recuperar dos erros rapidamente e facilmente.	Facilidade
Q17	Eu posso usá-lo com sucesso todas as vezes.	Facilidade
Q18	Eu aprendi a usá-lo rapidamente.	Aprendizagem
Q19	Eu me lembro facilmente como usá-lo.	Aprendizagem
Q20	É fácil de aprender a usá-lo.	Aprendizagem
Q21	Eu rapidamente me tornei hábil com ele.	Aprendizagem
Q22	Estou satisfeito com ele.	Satisfação
Q23	Eu recomendaria a um amigo.	Satisfação
Q24	É divertido de usar.	Satisfação
Q25	Ele funciona da maneira que eu quero que ele funcione.	Satisfação
Q26	É agradável de usar.	Satisfação
Q27	Liste os aspectos negativos da ferramenta.	Feedback
Q28	Liste os aspectos positivos da ferramenta.	Feedback

4.2 Resultados e Discussão

Durante o período em que a ferramenta esteve disponível para avaliação, 10 participantes se voluntariaram, todos estudantes de Ciência da Computação, sendo que desses, 9 são alunos da pós-graduação e 1 da graduação. O gráfico da Figura 4.1 apresenta a experiência dos participantes em ferramentas de modelagem e ABMS. Na figura podemos observar que 60% dos participantes possuem pouca ou nenhum conhecimento em ABMS, 30% possuem um conhecimento intermediário e 50% possuem um conhecimento acima do intermediário. Quanto as experiências dos participantes com ferramentas de modelagem, 60% relataram ter pouca ou nenhuma experiência no uso dessas ferramentas, 20%

Figura 4.1: Conhecimento dos participantes em ferramentas de modelagem e ABMS

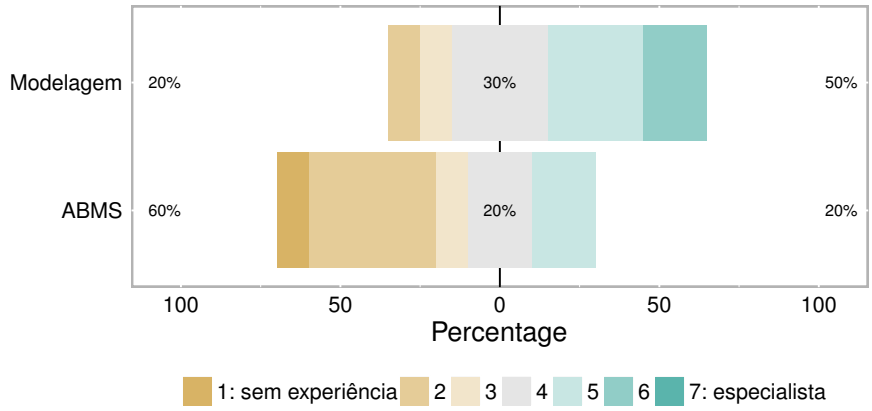
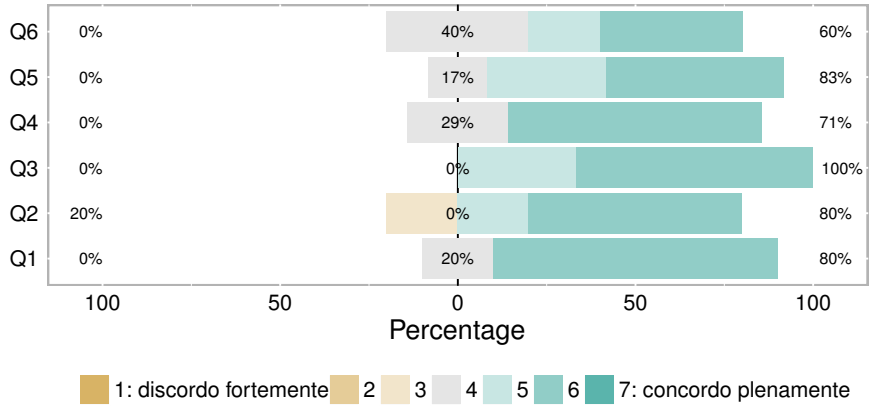


Figura 4.2: Utilidade da ferramenta ABM Tool para o participantes



relataram ter um conhecimento intermediário, e 20% relataram que tem um conhecimento avançado ou são especialistas em ferramentas de modelagem. Também foi questionado quanto a participação dos usuários no experimento realizado por Santos, Nunes and Bazzan (2016), que avaliou a facilidade de compreensão de sua *DSL4ABMS* para ABMS em relação ao NetLogo. Dentre os participantes 70% responderam que participaram, e 30% responderam que não.

A Figura 4.2 mostra o gráfico das respostas dos participantes quanto a utilidade da ferramenta ABM Tool. Nele pode-se constatar que 100% dos participantes concordam que a ferramenta é eficaz, produtiva e útil, e mais de 67% dos participantes julgaram que a ferramenta poupa tempo e torna a vida mais fácil.

A Figura 4.3 mostra o gráfico das respostas dos participantes quanto a facilidade de utilização da ferramenta ABM Tool. Neste grupo de questões, os participantes foram questionados o quanto a ferramenta é fácil, simples, amigável, se apresenta inconsistências, requer o menor empenhado de esforço para sua utilização, permite recuperação de erros rapidamente e facilmente. Dos participantes, 100% concordaram em algum nível,

Figura 4.3: Facilidade de utilização da ferramenta ABM Tool pelos participantes

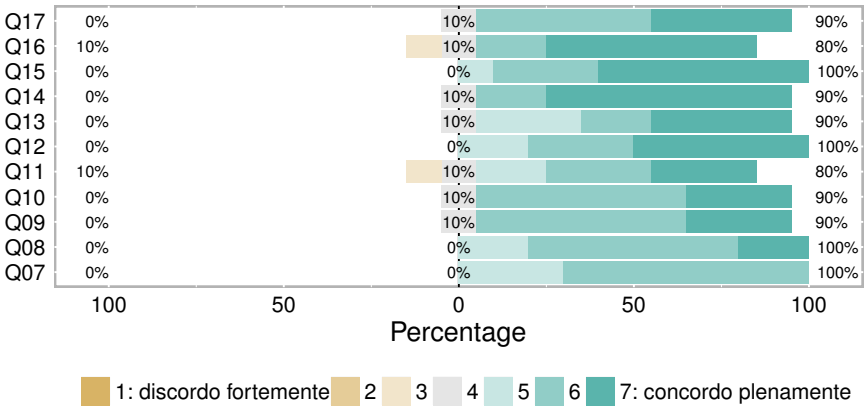
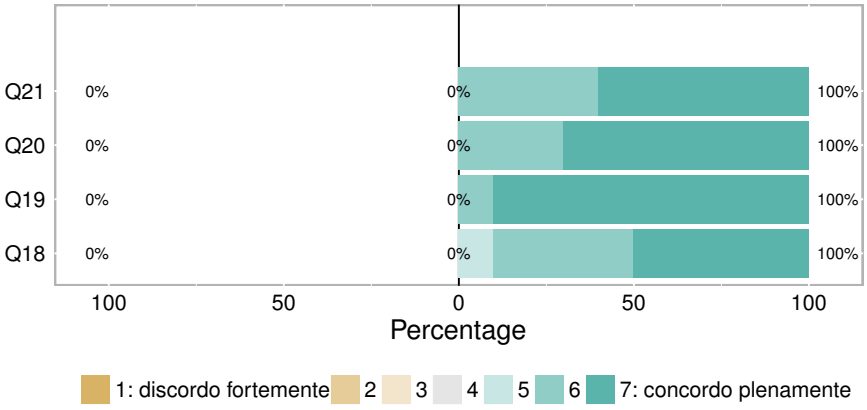


Figura 4.4: Facilidade de aprendizado dos participantes no uso da ferramenta ABM Tool

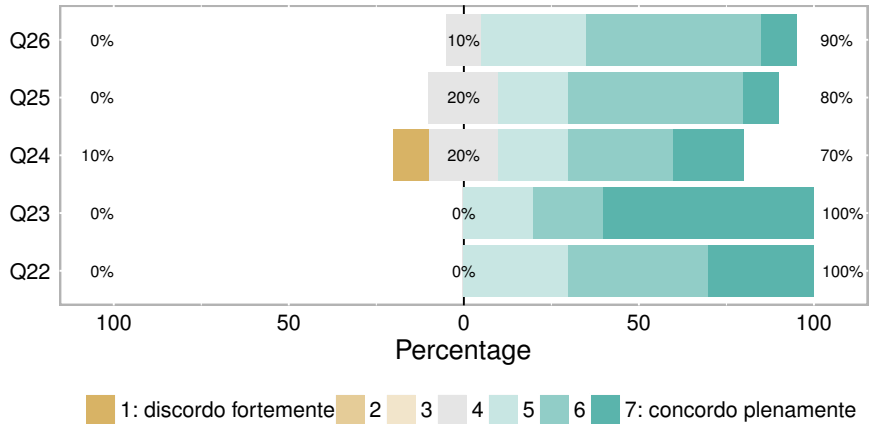


que a ferramenta ABM Tool é fácil, simples, que a utilizaram sem esforço, que não detectaram inconsistência ao utilizá-la e 90% relataram que conseguiriam utilizar a ferramenta sem instruções escritas, como por exemplo manuais e tutoriais, e que podem recuperar facilmente e rapidamente de eventuais erros que tenham cometido durante a modelagem.

No gráfico da Figura 4.4 é apresentado os resultados da seção que coletou a facilidade de aprendizado dos participantes ao utilizarem a ferramenta ABM Tool. Para 100% dos participantes concordaram que aprenderam a usar ABM Tool rapidamente e com facilidade, e também que a ferramenta as tornou mais hábeis.

Já no gráfico da Figura 4.5 é apresentado os resultados da seção que coletou a satisfação dos participantes no uso da ferramenta ABM Tool. Para 100% participantes ficaram satisfeitos com a ferramenta e a recomendariam a um amigo, 70% concordaram que a ferramenta é divertida e agradável de usar. No entanto, 10% dos participantes relataram que a ABM Tool não funcionou de maneira que queriam. Isso porque alguns participantes tiveram dificuldades na edição dos campos dos elementos, e também porque funcionalidades padrões em campos de texto (por exemplo a tecla TAB para alternar entre

Figura 4.5: Satisfação dos participantes no uso da ferramenta ABM Tool



os campos) na edição de texto não estão disponíveis para utilização na ferramenta.

Todos os participantes conseguiram completar a tarefa em um tempo inferior a o limite estabelecido de 40 minutos. O gráfico da Figura 4.6 compara o tempo que foi gasto pelo participante para conclusão da tarefa com a experiencia na participação do experimento que avaliou a *DSL4ABMS*. No gráfico notamos que os que não participaram, demoraram em média, 5 minutos a mais que os demais que participaram. A Tabela 4.3 relaciona os tempos para conclusão da tarefa pelos participantes com a sua experiência com ferramentas de modelagem e ABMS. A experiência em ABMS e a utilização de ferramentas para modelagem não foram relevantes na eficiência na manipulação da ferramenta, quando comparado o tempo de conclusão da tarefa.

Como aspectos positivos da ABM Tool, os participantes apontaram que a ferramenta possibilita o aprendizado facilmente, torna mais visível o que se pretende modelar e a curva de aprendizagem para seu uso é bem pequena. Comentaram também, que a ferramenta possui uma interface simples, intuitiva e agradável, e que ela representa perfeitamente a linguagem proposta por Santos, Nunes and Bazzan (2016). A funcionalidade de detalhamento de *Concern* foi muito elogiada, pois permite aprofundar a granularidade dos *Concerns*.

Apesar dos resultados positivos da avaliação, os participantes reportaram alguns fatores negativos de usabilidade da ferramenta, como por exemplo, a impossibilidade de redimensionar os elementos no diagrama, a dificuldade para editar campo de texto e criar relacionamentos com entidades em *Concerns* distintos. Tais limitações já eram conhecidas, conforme descritas na Seção 3.4, e evidenciam a necessidade de um aprimoramento de algumas funcionalidade, como por exemplo a de edição de atributos, e também a implementação de novas funcionalidades, como por exemplo a que permite o redimensiona-

Figura 4.6: Experiência com a *DSL4ABMS versus* tempo de avaliação

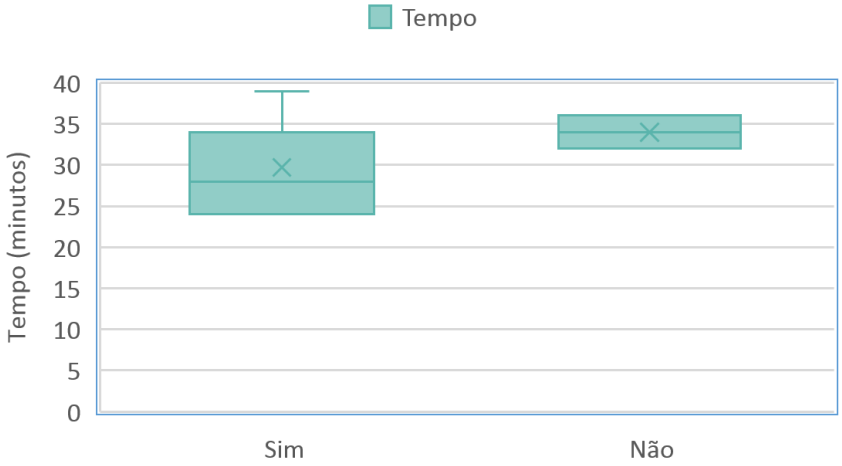


Tabela 4.3: Experiência *versus* tempo para conclusão da avaliação

Participante	***Experiência de Modelagem***	***Experiência de ABMS***	***Duração da Avaliação***
P1	3	4	39
P2	5	2	31
P3	2	2	34
P4	4	5	24
P5	6	5	24
P6	5	2	28
P7	6	2	28
P8	4	1	36
P9	4	4	32
P10	5	3	34

mento dos elementos, visando sanar as dificuldades edição e visualização apontadas.

Os participantes também apontaram problemas na visualização dos elementos que compõem o diagrama, como por exemplo, tamanho pequeno dos textos no elemento de visão geral que acaba dificultando a leitura, o tamanho das entidades no diagrama, a falta de cores nos elementos, que podem adicionar informação sobre o tipo de dado que o objeto contém ou representa.

Ainda, os participantes fizeram diversas sugestões de melhoria em funcionalidades e na interface gráfica pelos participantes, tais como: permitir filtrar as caixas de sugestão (lista com todos os valores possíveis para o campo) gerando mais produtividade ao usuário; a habilitação da tecla TAB, para navegação entre os campos dos objetos; e adição de teclas de atalhos para manipulação dos elementos no diagrama.

Durante todo o período da avaliação, os participantes foram observados, sem interferência, com o objetivo de coletar informações de como os usuários utilizam a ferramenta e quais são suas maiores dificuldades. Observou-se que a maior dificuldade dos usuários

concentra-se na edição dos campos de texto e na inclusão de relacionamento com entidades em *Concerns* distintos. Também, a notação da forma de inicialização dos atributos e parâmetros está diferente da notação original, gerando confusão e dúvida. Em alguns casos, ocorreram dificuldades na edição da cardinalidade de uma relação, pois a fonte é muito pequena, dificultando a leitura e a edição dos campos de texto.

4.3 Considerações Finais

Nesta seção descrevemos a metodologia e os resultados da avaliação da ferramenta ABM Tool. Também discutimos as dificuldades encontradas pelos participantes da avaliação e possíveis melhorias que podem ser implementadas para sanar os problemas apontados. Na próxima seção, descrevemos algumas ferramentas para ABMS, como o ambiente é definido, os elementos que compõem o ambiente e como o ambiente e os elementos são criados e inicializados.

5 TRABALHOS RELACIONADOS

Neste capítulo são apresentados trabalhos relacionados ao desenvolvimento de ferramentas voltadas para a modelagem baseada em agentes. Estes trabalhos podem ser classificados em dois grupos, de acordo com a sintaxe concreta disponibilizada: modelagem textual ou modelagem visual. Dentre as ferramentas de modelagem textual, pode-se citar *Swarm*, *NetLogo*, *RePast* e o *TerraME*. Estas ferramentas disponibilizam essencialmente uma linguagem de programação para desenvolver simulação com agentes. Já dentre as ferramentas visuais, há o *SeSAm*.

O *Swarm* é uma plataforma para modelagem baseada em agentes que inclui um *framework* conceitual para projetar, descrever e conduzir experimentos em modelagem baseada em agentes. No *Swarm* é possível representar um modelo de simulação completo, composto de agentes e a representação do tempo. O *Swarm* suporta modelagem hierárquica, onde agentes podem ser compostos por outros modelos do *Swarm* em estruturas aninhadas. Ele também oferece uma biblioteca orientada a objetos com componentes reutilizáveis para construção, análise, exibição e controle de modelos. Agentes, entidades situadas, unidades espaciais e interações são modeladas na semântica da orientação a objetos, através de classes e atributos. Porém o *Swarm* possui um curva de aprendizagem acentuada, sendo necessário ter conhecimento em linguagens de programação como Java ou Objective-C, e familiarizado com a metodologia de orientação a objetos. Além disso, o *Swarm* suporta apenas abstrações espaciais do tipo grade (MINAR et al., 1996; SWARM, 2016).

O *NetLogo* é um ambiente multiplataforma para modelagem de ambientes multiagentes, particularmente adaptado para modelar sistemas complexos que se desenvolvem ao longo do tempo. O ambiente suporta milhares de agentes, todos operando de forma independente. No *NetLogo* são suportadas as abstrações espaciais do tipo grade e GIS. Ele oferece uma linguagem simples de programação, visualização 2D e 3D do modelo, controle de velocidade da simulação e monitores que permitem inspecionar e controlar os agentes. O *NetLogo* prove uma linguagem própria de programação para desenvolver os comportamentos das entidades e dos agentes, definindo por fim o comportamento associado a cada elemento da interface. A plataforma não oferece recursos e facilidades específicas para a construção do modelo, tendo o modelador que especificar diretamente através de uma linguagem de programação, ainda que de alto nível, as estruturas e regras que definem o comportamento do modelo (WILENSKY, 1999).

O *RePast* é um *framework* para modelagem baseada em agentes. O *Repast* é inteiramente orientado a objetos, permite a o modelador modificar dinamicamente as propriedades dos agentes, equações de comportamento, e propriedade do modelo em tempo de execução. Além disso, no *Repast* os modelos podem ser desenvolvidos em diversas linguagens de programação como Java, C#, Visual Basic e Python. Os agentes, as unidades espaciais, e as interações entre entidades e com o ambiente são modelados utilizando classes e atributos. O RePast também implementa abstrações espaciais, que podem ser do tipo grade, GIS, espaço continuo e rede. Os agentes se relacionam através de redes (NORTH; COLLIER; VOS, 2006; REPAST, 2016).

O *TerraME* ou (*TerraLib Modelling Environment*) é uma ferramenta para a implementação e simulação de modelos ambientais que envolvam a representação explícita do espaço. O TerraME provê mecanismos que permitem a fácil representação e a eficiente simulação de modelos espaciais dinâmicos integrados a um sistema de informações geográficas. Os componentes de sua arquitetura de software, oferecem serviços específicos a usuários com diferentes níveis de conhecimento em algoritmos e técnicas de programação. Usuários experientes podem implementar modelos utilizando diretamente o framework de modelagem *TerraME* através da linguagem de programação C++, enquanto aqueles que possuem apenas os conhecimentos básico sobre algoritmos e modelagem computacional podem utilizar a linguagem de programação de alto nível *TerraME Modeling Language*, que permite a fácil escrita, leitura e alteração dos modelos (CARNEIRO et al., 2013).

O *SeSAm* ou (*Shell for Simulated Agent Systems*) é uma ferramenta de modelagem e simulação baseada em sistemas multiagentes. Ele prove um ambiente genérico para construção de modelos complexos, que podem incluir interdependências dinâmicas ou comportamentos emergentes. Diferente das abordagens como o *NetLogo* e *Repast*, o *SeSAm* propõem a construção de modelos a partir da programação visual, através da definição de estados e comportamento em nível de especificação, utilizando diagramas de atividades para especificar os comportamentos dos agentes e os estados possíveis dentro do sistema. Além disso, o *SeSAm* disponibiliza os usuários, uma grande quantidade de componentes predefinidos, possibilitando ao usuário projetar toda a simulação graficamente, sem a necessidade de estar familiarizado com uma linguagem de programação especifica (KLÜGL; HERRLER; FEHLER, 2006).

A ABM Tool engloba a maioria dos recursos de modelagem presentes nas ferramentas apresentadas, tais como a definição de entidades, parâmetros, estratégias de

criação e abstrações espaciais variados, como grade, plano cartesiano e grafo. A ABM Tool tem a vantagem de ser uma ferramenta visual, que possibilita ao usuário uma visão de tudo que está sendo modelado, enquanto que as ferramentas apresentadas, em sua grande maioria, oferecem um ambiente de modelagem textual, que muitas vezes depende de conhecimento prévio em programação. Além disso, ABM Tool possui uma interface simplificada e amigável, focando apenas na modelagem do ambiente. No entanto, a ferramenta não possui um complemento para simulação e nem um procedimento que converta o modelo desenvolvido para plataformas de simulação, como por exemplo o NetLogo.

5.1 Considerações Finais

Este capítulo apresentou ferramentas e abordagens adotadas para ABMS. Destacamos vantagens e desvantagens de cada ferramenta, como por exemplo, o fato das ferramentas que utilizam uma abordagem textual exigirem mais conhecimento de programação do projetista. Algumas ferramentas como *SeSAm* utilizam diagramas de estado para especificar o comportamento dos agentes, outras como o *NetLogo*, que utilizam uma linguagem textual própria. Também há frameworks para ABMS, como por exemplo *RePast*, que é totalmente orientado a objetos, podendo ser utilizado por linguagens de programação como Java, C#, C++, entre outras. No final, comparamos a ABM Tool com as ferramentas apresentada, elencando vantagens e desvantagens da ABM Tool, quando comparada com as outras ferramentas.

6 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de uma ferramenta para construção de modelos baseados em agentes, partindo de uma especificação de DSL de Santos, Nunes and Bazzan (2016). A ferramenta foi implementada utilizando o *Framework Graphiti* em conjunto com a IDE Eclipse. A solução proposta serve como ponto de partida para o desenvolvimento de ferramentas gráficas que auxiliem usuários comuns no desenvolvimento de modelos de simulação baseado em agentes.

Foi realizada uma avaliação da ferramenta com usuários, visando testar a usabilidade da ferramenta. Constatamos que a ferramenta pode ser usada com facilidade por usuários com e sem experiência em ferramentas de modelagem. A avaliação possibilitou também coletar as opiniões dos usuários, e observar dificuldades no uso da ferramenta, no intuito de detectar melhorias a serem implementadas que melhorem a usabilidade da ferramenta.

Acreditamos que com a evolução da ferramenta seja possível, a partir de um modelo criado pela nossa ferramenta, gerar código fonte para ferramentas de modelagem e simulação consolidadas no mercado, como por exemplo o *NetLogo*. Isto caracterizaria uma abordagem *model-driven* para ABMS, que já tem se mostrado vantajosa no contexto de engenharia de software, reduzindo o esforço de desenvolvimento.

Como trabalhos futuros, a curto prazo, são necessárias melhorias em alguns elementos do diagrama, para que seja possível o redimensionamento de pictogramas. Também é necessário criar um instalador da ferramenta para a IDE Eclipse, que facilite a localização da ferramenta pelo usuário. A longo prazo, funcionalidades já implementadas devem ser revistas e melhoradas, a fim de se obter melhorias de desempenho no processamento gráfico. Outro objetivo a longo prazo é integrar a ABM Tool com geradores de código, cujo desenvolvimento já está em curso pelos autores da DSL.

REFERÊNCIAS

BARZDINS, J. et al. Domain specific languages for business process management: a case study. In: **Proceedings of DSM**. [S.l.: s.n.], 2009. v. 9, p. 34–40.

BIZAGI. **Bizagi Official Digital Business Platform and BPMS**. 2016. Acesso em: 16 de Novembro de 2016. Disponível em: <http://www.bizagi.com/pt/>.

BONABEAU, E. Agent-based modeling: Methods and techniques for simulating human systems. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 99, n. suppl 3, p. 7280–7287, 2002.

CARNEIRO, T. G. de S. et al. An extensible toolbox for modeling nature–society interactions. **Environmental Modelling & Software**, v. 46, p. 104 – 117, 2013. ISSN 1364-8152. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1364815213000534>.

CASTLE, C. J. et al. Agent-based modelling and simulation using repast: A gallery of gis applications. In: **Proceedings of the GIS Research UK 14th Annual Conference GISRUK 2005**. [S.l.: s.n.], 2005.

COLLIER, N. **Repast: An extensible framework for agent simulation**. 2003.

CROOKS, A. T.; WISE, S. {GIS} and agent-based models for humanitarian assistance. **Computers, Environment and Urban Systems**, v. 41, p. 100 – 111, 2013. ISSN 0198-9715. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0198971513000550>.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. **Sigplan Notices**, v. 35, n. 6, p. 26–36, 2000.

DEURSEN, A. van; KLINT, P. Little languages: Little maintenance. **Journal of Software Maintenance**, John Wiley & Sons, Inc., New York, NY, USA, v. 10, n. 2, p. 75–92, mar. 1998. ISSN 1040-550X. Disponível em: <http://dx.doi.org/10.1002/(SICI)1096-908X(199803/04)10:2<75::AID-SMR168>3.0.CO;2-5>.

ECLIPSE. **Graphiti Developer Guide**. 2016. Acesso em: 08 de Novembro de 2016. Disponível em: <http://help.eclipse.org/kepler/>.

FOWLER, M. **Domain-specific languages**. [S.l.]: Pearson Education, 2010.

GALÁN, J. M. et al. Errors and artefacts in agent-based modelling. **Journal of Artificial Societies and Social Simulation**, v. 12, n. 1, p. 1, 2009. ISSN 1460-7425. Disponível em: <http://jasss.soc.surrey.ac.uk/12/1/1.html>.

GOSLING, J. **The Java language specification**. [S.l.]: Addison-Wesley Professional, 2000.

HIEBELER, D. et al. The swarm simulation system and individual-based modeling. **Proceedings of Decision Support 2001: Advanced technology for natural resource management**, p. 20–26, 1994.

HUDAK, P. Modular domain specific languages and tools. In: IEEE. **Software Reuse, 1998. Proceedings. Fifth International Conference on**. [S.l.], 1998. p. 134–142.

KIEBURTZ, R. B. et al. A software engineering experiment in software component generation. In: IEEE COMPUTER SOCIETY. **Proceedings of the 18th international conference on Software engineering**. [S.l.], 1996. p. 542–552.

KLÜGL, F.; BAZZAN, A. L. C. Agent-based modeling and simulation. **AI Magazine**, v. 33, n. 3, p. 29–40, 2012. Disponível em: <http://www.aaai.org/ojs/index.php/aimagazine/article/view/2425>.

KLÜGL, F.; HERRLER, R.; FEHLER, M. Sesam: Implementation of agent-based simulation using visual programming. In: **Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems**. New York, NY, USA: ACM, 2006. (AAMAS '06), p. 1439–1440. ISBN 1-59593-303-4. Disponível em: <http://doi.acm.org/10.1145/1160633.1160904>.

LADD, D. A.; RAMMING, J. C. Two application languages in software production. In: **USENIX Very High Level Languages Symposium Proceedings**. [S.l.: s.n.], 1994. p. 169–178.

LANGLOIS, B.; JITIA, C. E.; JOUENNE, E. DSL Classification. In: **Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling**. [s.n.], 2007. Disponível em: <http://www.dsmforum.org/events/DSM07/papers/langlois_jitia.pdf>.

LIKERT, R. A technique for the measurement of attitudes. **Archives of psychology**, 1932.

LONGLEY, P. **Geographic information systems and science**. [S.l.]: John Wiley & Sons, 2005.

LUND, A. M. Measuring usability with the use questionnaire12. 2001.

MACAL, C.; NORTH, M. Introductory tutorial: Agent-based modeling and simulation. In: **Proceedings of the 2014 Winter Simulation Conference**. Piscataway, NJ, USA: IEEE Press, 2014. (WSC '14), p. 6–20. Disponível em: <http://dl.acm.org/citation.cfm?id=2693848.2693856>.

MACAL, C. M.; NORTH, M. J. Tutorial on agent-based modeling and simulation part 2: How to model with agents. In: **Proceedings of the 38th Conference on Winter Simulation**. [S.l.]: Winter Simulation Conference, 2006. (WSC '06), p. 73–83. ISBN 1-4244-0501-7.

MACAL, C. M.; NORTH, M. J. Tutorial on agent-based modelling and simulation. **Journal of simulation**, Nature Publishing Group, v. 4, n. 3, p. 151–162, 2010.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. **ACM computing surveys (CSUR)**, ACM, v. 37, n. 4, p. 316–344, 2005.

MINAR, N. et al. The swarm simulation system: A toolkit for building multi-agent simulations. Technical report, Swarm Development Group, 1996.

NORTH, M. J.; COLLIER, N. T.; VOS, J. R. Experiences creating three implementations of the repast agent modeling toolkit. **ACM Trans. Model. Comput. Simul.**, ACM, New York, NY, USA, v. 16, n. 1, p. 1–25, jan. 2006. ISSN 1049-3301. Disponível em: <http://doi.acm.org/10.1145/1122012.1122013>.

NORTH, M. J.; MACAL, C. M. **Managing business complexity: discovering strategic solutions with agent-based modeling and simulation**. [S.l.]: Oxford University Press, 2007.

REPAST. **Homepage Repast**. 2016. Acesso em: 18 de Novembro de 2016. Disponível em: <http://repast.sourceforge.net/repast_3/index.html>.

RESNICK, M. Starlogo: An environment for decentralized modeling and decentralized thinking. In: **Conference Companion on Human Factors in Computing Systems**. New York, NY, USA: ACM, 1996. (CHI '96), p. 11–12. ISBN 0-89791-832-0. Disponível em: <http://doi.acm.org/10.1145/257089.257095>.

RUSSELL, S. J. et al. **Artificial Intelligence: A Modern Approach**. [S.l.]: Prentice hall Upper Saddle River, 2003.

SANTOS, F.; NUNES, I.; BAZZAN, A. L. **A Language for Environment Modeling in Agent-based Modeling and Simulations**. 2016. Submitted.

SWARM. **Swarm Main Page**. 2016. Acesso em: 18 de Novembro de 2016. Disponível em: <http://www.swarm.org/>.

WILENSKY, U. **NetLogo**. 1999. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. Disponível em: <http://ccl.northwestern.edu/netlogo/>.

WOOLDRIDGE, M. J. **Multi-agent systems: an introduction**. [S.l.]: Wiley Chichester, 2001.