Uma Análise Quantitativa do Esforço de Desenvolvimento em Plataformas de Simulações baseadas em Agentes

Fernando Santos[1,2], **Ingrid Nunes**[1,3]

[1]Instituto de Informática, UFRGS, Brazil

[2]Departamento de Engenharia de Software, UDESC, Brazil.

[3]TU Dortmund, Germany

fernando.santos@udesc.br, ingridnunes@inf.ufrgs.br

Resumo. *Simulações baseadas em agentes têm sido usadas para entender comportamentos emergentes de sistemas complexos. Tais simulações são normalmente desenvolvidas em plataformas de simulação, que oferecem recursos inerentes à área de agentes e portanto simplificam o desenvolvimento. O esforço requerido para desenvolver simulações pode influenciar a escolha de uma plataforma. Estudos que avaliam as plataformas existentes não consideram este aspecto. Através de um estudo de caso com a simulação* Sugarscape*, este artigo apresenta uma avaliação quantitativa do esforço de desenvolvimento em duas plataformas: NetLogo, que oferece linguagem de desenvolvimento textual; e SeSAm, com linguagem gráfica. Resultados indicam que o SeSAm demanda 35.6% mais esforço que o NetLogo, de acordo com o estudo de caso realizado.*

Abstract. *Agent-based simulations have been used to understand the emergent behavior of complex systems. These simulations are often developed using simulation platforms, which provide features that are intrinsic to agent technology, thus simplifying the development. The effort to develop simulations can influence platform selection. Other studies that evaluated existing platforms did not take this aspect into account. By means of a case study using the* Sugarscape *simulation, this paper presents a quantitative evaluation of the effort to develop simulations using two platforms: NetLogo, which provides a textual development language; and SeSAm, which provides a graphical language. Results provide evidence that SeSAm demands 35.6% more effort than NetLogo, according the performed case study.*

1. Introdução

Simulações baseadas em agentes têm sido usadas para entender comportamentos emergentes de sistemas complexos. Estes sistemas são constituídos por várias entidades, ou agentes, que podem interagir entre si. Estes agentes estão situados em um ambiente, e podem percebê-lo e modificá-lo por meio de ações. Não é trivial formular modelos analíticos para simular o comportamento destes sistemas complexos. Construí-los é uma tarefa desafiadora, que tem sido investigada no contexto de modelagem e simulação baseada em agentes ou, em inglês, *agent-based modeling and simulation* (ABMS). Simulações que fazem uso do conceito de agentes têm sido usadas para modelar sistemas em diversos domínios, tais como desastres naturais, economia e ciências sociais.

Para desenvolver simulações com agentes, linguagens de propósito geral, tais como Java ou C, podem ser utilizadas. Entretanto, existem plataformas de simulação que oferecem recursos inerentes a agentes e que, portanto, simplificam o desenvolvimento. Estas plataformas frequentemente adotam uma linguagem textual ou gráfica para desenvolver as simulações. Entre as plataformas de simulação mais proeminentes [Klügl e Bazzan 2012, Kravari e Bassiliades 2015], estão Swarm, Repast, MASON, NetLogo e SeSAm. Destas, apenas a última adota linguagem gráfica para o desenvolvimento; as demais adotam programação textual.

Trabalhos existentes avaliaram estas plataformas de simulação considerando características elementares, tais como a linguagem e os recursos disponibilizados para desenvolvimento. Entretanto, um aspecto que pode afetar a escolha da plataforma adotada é o esforço requerido para o desenvolvimento de simulações, isto é, o quão custoso é desenvolver uma simulação na plataforma escolhida, o que impacta no tempo e quantidade de desenvolvedores necessários para o desenvolvimento. Esforço é usualmente avaliado em termos de horas de trabalho que, em Engenharia de Software, está diretamente relacionado (i.e. é em função de) ao tamanho do código fonte, ou modelo. Além de fundamentar uma eventual escolha de plataforma, a métrica de esforço de desenvolvimento pode indicar se, de fato, uma linguagem alternativa, como uma linguagem gráfica, oferece benefícios em relação a uma linguagem textual.

Neste trabalho apresentamos uma avaliação do esforço requerido para desenvolver simulações em duas plataformas: NetLogo e SeSAm. Esta avaliação é inovadora, haja visto que avaliações existentes apenas consideraram características elementares das plataformas de simulação. A avaliação foi conduzida a partir de um estudo de caso, que considerou a simulação *Sugarscape* [Epstein e Axtell 1996], largamente utilizada no contexto de ABMS. Esta única simulação foi avaliada em ambas as plataformas, para proporcionar uma comparação justa. Métricas foram coletadas para avaliar o esforço de desenvolver em cada plataforma. O esforço é medido através do volume de código produzido (em termos de linhas de código) e do volume de artefatos de modelagem produzidos (em termos de elementos atômicos do modelo). A medição feita baseia-se em trabalhos que indicam como a contagem deve ser realizada. Os resultados indicam que o esforço para desenvolver com o SeSAm é 35.6% maior que com o NetLogo, considerando o estudo de caso. Uma análise dos artefatos criados no desenvolvimento apontou que isto se deve a maior abstração fornecida pelo NetLogo, que dispõe de uma biblioteca com comandos para comportamentos recorrentes, tais como percepção e movimentação.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica em ABMS e trabalhos relacionados. A Seção 3 apresenta o estudo de caso, sendo descritos a simulação *Sugarscape*, as plataformas NetLogo e SeSAm, as métricas de esforço adotadas, e os resultados obtidos. Por fim, as conclusões e trabalhos futuros são apresentados na Seção 4.

2. Fundamentação Teórica e Trabalhos Relacionados

A modelagem e simulação baseada em agentes (ABMS) é um paradigma de simulação que utiliza agentes para analisar, reproduzir ou predizer fenômenos normalmente complexos e emergentes. Em ABMS, um modelo do sistema em estudo é construído em termos de agentes, que interagem com seus pares e também com o ambiente. A par-

tir da simulação do modelo, através de repetida execução por determinado intervalo de tempo, pode-se obter informações significativas a respeito das propriedades do sistema ou fenômeno analisado, bem como sobre sua própria evolução [Garro e Russo 2010].

ABMS tem sido usada para modelar e simular sistemas em diversas áreas, tais como agrigultura, tráfego aéreo, ecologia, economia, epidemiologia, assistência à saúde e redes sociais [Macal e North 2014]. Além disso, segundo [Macal e North 2014], ABMS tem sido usada nestas áreas em função da possibilidade de incorporar comportamentos individuais e interações complexas que existem no mundo real.

Simulações baseadas em agentes são frequentemente desenvolvidas em plataformas de agentes. Estas plataformas disponibilizam recursos inerentes a agentes visando simplificar o desenvolvimento. Há uma entrada na Wikipedia[1] onde a comunidade de agentes mantém uma relação de plataformas disponíveis. No momento em que este artigo foi elaborado, haviam 89 plataformas relacionadas. Enquanto muitas destas plataformas oferecem recursos apenas para desenvolver agentes, outras oferecem recursos focados no aspecto de simulação (por exemplo, *setup* da execução, visualização e coleta de dados). As plataformas Swarm [Minar et al. 1996], Repast [North et al. 2006], MASON [Luke et al. 2005], NetLogo [Wilensky 1999] e SeSAm [Klügl et al. 2006] são apontadas como as mais proeminentes para simulação [Klügl e Bazzan 2012, Kravari e Bassiliades 2015]. Destas, SeSAm é a única que adota uma linguagem gráfica para o desenvolvimento de simulações. Todas as demais adotam uma linguagem de programação textual. Repast e MASON adotam a linguagem de programação Java; Swarm adota Objective-C; e NetLogo adota uma variação da linguagem Logo.

Avaliações destas plataformas têm sido realizadas sob diferentes pontos de vista. [Railsback et al. 2006] analisaram quatro plataformas (NetLogo, Mason, Repast e Swarm). A análise considerou o suporte a determinados elementos que podem estar presentes em simulações, tais como o ambiente e sua estrutura, a dinâmica dos agentes, rotinas de leitura/escrita em arquivos e recursos estatísticos. Um *template* de simulação, chamado de *StupidModel*, foi usado para verificar o suporte a estes elementos. Os autores concluíram que, apesar de adotarem terminologias diferentes, estas plataformas suportam os elementos considerados. Aspectos geoespaciais foram avaliados no trabalho de [Castle e Crooks 2006], sendo consideradas as plataformas Repast, Swarm e Mason.

Outros trabalhos dedicaram-se a analisar as características elementares das várias plataformas disponíveis. Linguagem de programação, sistema operacional requerido, tipo de licença, domínio de aplicação e suporte ao usuário foram as características analisadas por [Nikolai e Madey 2009]. Já [Kravari e Bassiliades 2015] incluiram outras características na análise, tais como usabilidade, performance, maturidade e segurança. Estes trabalhos limitam-se a descrever estas características para cada plataforma analisada.

Como pode-se constatar, as avaliações existentes não consideram um aspecto que pode afetar a escolha da plataforma de simulação: o esforço de desenvolvimento. Este esforço está relacionado com o custo de desenvolvimento de sistemas, que há tempos é um objeto de estudo da Engenharia de Software [Albrecht 1979]. Métodos de estimativa de custos, tais como *Function Points* [Albrecht 1979] e *COCOMO* [Boehm et al. 1995], consideram métricas do esforço aplicado no desenvolvimento de sistemas já concluídos

[1] `https://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software`, Fev. de 2017

para estimar o esforço em projetos futuros.

3. Estudo de Caso

A avaliação do esforço de desenvolvimento foi conduzida a partir de um estudo de caso, que considera a simulação *Sugarscape*. Em seguida, detalhamos a simulação *Sugarscape* (Seção 3.1) e as plataformas selecionadas para análise (Seção 3.2), Posteriormente, apresentamos as métricas adotadas (Seção 3.3) e os resultados obtidos (Seção 3.4).

3.1. A Simulação Sugarscape

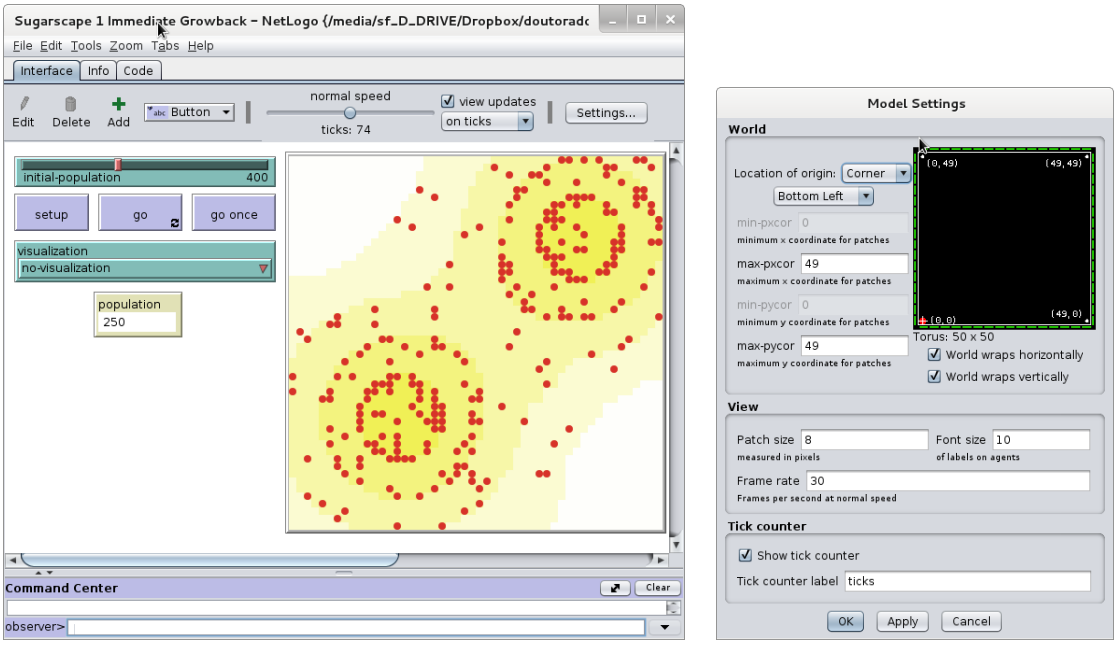
Sugarscape é um modelo de sociedade artificial proposto por [Epstein e Axtell 1996]. É amplamente utilizado na comunidade de simulações baseadas em agentes como um exemplo de simulação que apresenta fenômenos emergentes. Em essência, o modelo *Sugarscape* simula uma população que depende de recursos limitados existentes no ambiente (no caso, açúcar). O ambiente é representado por diferentes regiões, algumas ricas, outras pobres em açúcar. Agentes são capazes de perceber regiões próximas e possuem uma taxa metabólica. O alcance da visão e a taxa metabólica variam entre os agentes, formando uma população heterogênea. Ao longo da simulação, cada agente move-se no ambiente para coletar açúcar, haja visto que seu metabolismo consome açúcar para mantê-lo vivo, vindo a falecer ao consumir todo seu estoque individual.

No modelo de [Epstein e Axtell 1996], o ambiente é um conjunto de 2500 células, que representam as regiões, organizadas em uma grade de dimensão 50 x 50. Cada célula possui um determinado montante de açúcar e uma capacidade máxima de armazenamento. O montante inicial de açúcar de cada célula é definido de modo a formar dois picos de açúcar, no nordeste e sudeste da grade. Um arquivo externo fornece o montante inicial de açúcar de cada célula. A medida que a simulação evolui, o açúcar consumido pelos agentes é reposto. Na regra de reposição mais simples, adotada neste estudo de caso, o montante inicial de açúcar é reposto a cada passo da simulação. Esta regra é conhecida como crescimento imediato.

Na inicialização da simulação, referenciada como *setup*, agentes são criados com um estoque inicial de açúcar, um alcance de visão e uma taxa metabólica—estes dois últimos se mantém inalterados ao longo da simulação. Para movimentar-se no ambiente em busca de açúcar, cada agente executa as seguintes ações [Epstein e Axtell 1996]:

1. observa as células ao alcance da visão;
2. identifica a célula com maior montante de açúcar que não está ocupada;
3. se existirem várias células com o mesmo montante, seleciona a mais próxima;
4. move para a célula selecionada; e,
5. coleta todo o montante de açúcar nesta nova célula.

Apesar de simples, a simulação *Sugarscape* evidencia, por exemplo, o fenômeno ecológico emergente da capacidade de carga de uma espécie—segundo o qual um dado ambiente pode suportar apenas uma população finita [Epstein e Axtell 1996]. Esta simulação foi escolhida para o estudo de caso em função da sua simplicidade e também da existência de implementação para as plataformas selecionadas. Ao usar implementações existentes, diminui-se a probabilidade de introdução de viés no desenvolvimento destas simulações para o propósito deste estudo. Um exemplo é o uso desnecessário de instruções para implementar uma funcionalidade quando já há uma instrução disponível que a implementa.



(a) Simulação *Sugarscape* (b) Especificação do ambiente.

Figura 1. Interface do NetLogo.

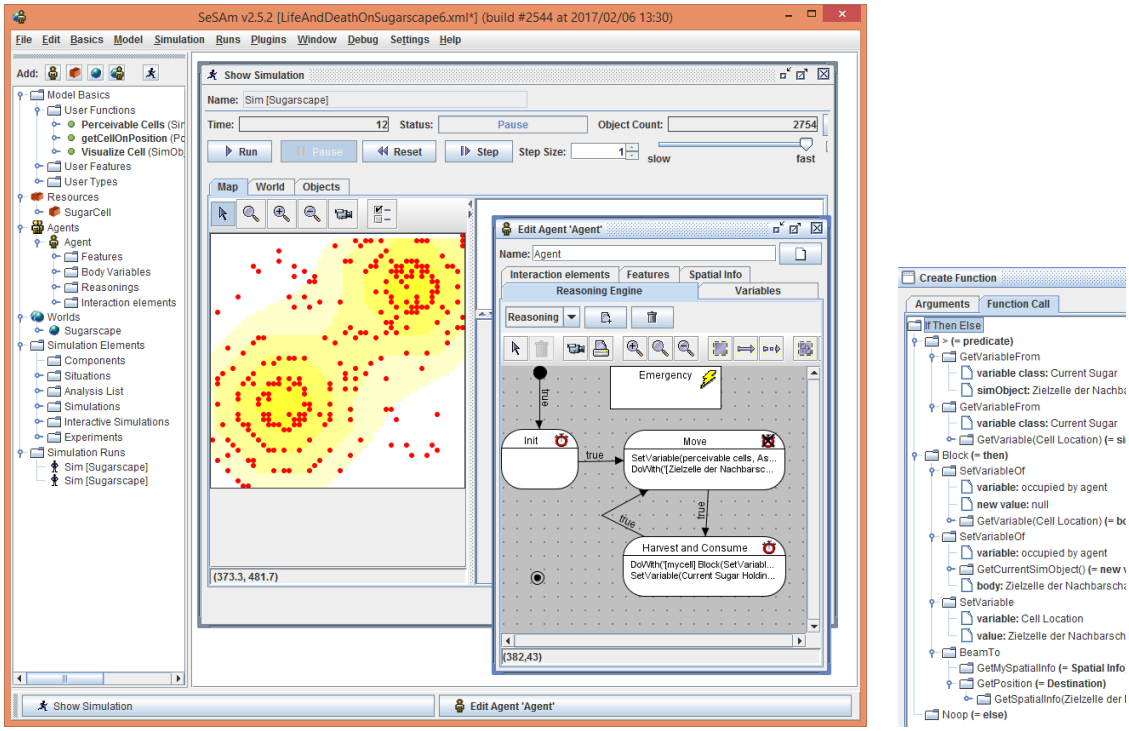
3.2. Plataformas Selecionadas

O objetivo deste trabalho é comparar o esforço para desenvolvimento de simulações em plataformas que adotam linguagem gráfica em relação àquelas que adotam linguagem textual. As plataformas inicialmente consideradas foram aquelas enumeradas por [Klügl e Bazzan 2012] e consideradas por [Kravari e Bassiliades 2015] como as mais proeminentes. Destas, a plataforma SeSAm [Klügl et al. 2006] é a única que adota linguagem gráfica. Entre as que adotam linguagem textual, a plataforma NetLogo [Wilensky 1999] foi escolhida, tendo em vista sua popularidade na comunidade de simulação com agentes. Estatísticas mostram que, dentre todas as simulações disponíveis no repositório *CoMSES* [OpenABM Consortium 2017] em 2014, 79% eram desenvolvidas em NetLogo [Rollins et al. 2014].

3.2.1. NetLogo

A plataforma NetLogo [Wilensky 1999] constitui-se de uma linguagem de programação e uma interface gráfica para especificar as características do ambiente simulado e os componentes para entrada de dados. Por natureza, o ambiente simulado é sempre uma grade composta de células. A Figura 1(a) apresenta a interface do NetLogo com a simulação *Sugarscape* carregada. A implementação do *Sugarscape* é distribuída com o NetLogo, estando contida em sua biblioteca de modelos. As características do ambiente são especificadas através de grupos de componentes, mostrados na Figura 1(b). O grupo *world* especifica as dimensões do ambiente. Nos grupos *view* e *tick counter* ficam os detalhes de visualização.

Através da linguagem de programação do NetLogo, projetistas podem definir tipos de agentes e suas características. A dinâmica do ambiente e dos agentes é definida apartir



(a) Simulação *Sugarscape* e diagrama de atividades do agente. (b) Funções.

Figura 2. Interface do SeSAm.

da implementação de *procedures* e *reporters*. Uma extensa biblioteca de instruções está disponível[2], incluindo instruções para percepção e movimentação dos agentes, além de instruções condicionais e de repetição.

3.2.2. SeSAm

A plataforma SeSAm [Klügl et al. 2006] permite o desenvolvimento de simulações de modo totalmente gráfico. A sua interface oferece componentes gráficos para especificar todos os elementos da simulação, que são classificados em *resources*, *agents*, *worlds*, *user functions*, *user types*, *variables*, *activities*, *transitions*, e *situations*. Diagramas de atividade são usados para especificar aspectos dinâmicos do ambiente e agentes. Cada atividade é composta por ações, que por sua vez são elaboradas a partir de funções pré-definidas, ou definidas pelo projetista. A Figura 2(a) apresenta a interface do SeSAm com a simulação *Sugarscape* carregada, onde também pode ser visto o diagrama de atividades que define o comportamento dos agentes.

As ações que compõem as atividades são especificadas graficamente, através da elaboração de uma árvore de funções, como pode ser visto na Figura 2(b). Não há linguagem de programação textual para definir as ações. A implementação da simulação *Sugarscape* também é distribuída com a plataforma SeSAm, em sua biblioteca de modelos. A implementação faz uso da biblioteca de funções primitivas do SeSAm[3], comparativa-

[2] https://ccl.northwestern.edu/netlogo/docs/dictionary.html
[3] http://130.243.124.21/mediawiki/index.php/Primitives

mente menor do que a biblioteca de instruções do NetLogo.

3.3. Métricas para Avaliação de Esforço de Desenvolvimento

A área de Engenharia de Software tem usado métricas de código, tal como a quantidade de linhas de código, ou *lines of code* (LoCs), como forma de medir o tamanho de um sistema. Esta métrica leva a medidas mais tradicionais de esforço como horas de trabalho necessárias para desenvolver sistemas. Na área de agentes, [Challenger et al. 2015] propõem o uso de LoCs como métrica objetiva para avaliar ambientes de desenvolvimento de agentes. Em ambientes que oferecem uma linguagem de desenvolvimento textual, caso do NetLogo, a métrica LoCs pode ser diretamente aplicada.

Em ambientes que oferecem uma linguagem gráfica, como o SeSAm, a métrica de LoCs não pode ser aplicada. Para linguagens gráficas, [Bettin 2002] propõe a métrica de elementos atômicos do modelo, ou *atomic model elements* (AMEs). Uma AME é um elemento de modelagem visual que é equivalente a uma LoC. Para classificar elementos como AMEs é necessário estabelecer regras de classificação. Por exemplo, [Bettin 2002] propõe uma regra generosa para classificar elementos de diagramas de classes: cada elemento que representa uma classe é considerado AME. Idem para os elementos que representam atributos de classe. Para realizar uma comparação justa, foram adotadas convenções para contar LoCs e AMEs em cada plataforma.

A contagem de LoCs no NetLogo considera a indentação original do código fonte, que segue as práticas observadas em todos os modelos disponíveis nesta plataforma. Cada instrução, seja de leitura/escrita ou controle, contabiliza uma LoC. Declarações de *procedures* ou *reporters* contabilizam duas LoCs adicionais (cabeçalho e rodapé). Comentários não são contabilizados, bem como linhas em branco e os delimitadores de bloco '[' e ']' quando são o único conteúdo de uma linha. Já para AMEs no NetLogo, as regras de contagem foram elaboradas com base nas regras de [Bettin 2002] para diagramas de classe. Cada elemento visual que especifica um item da simulação é contabilizado como AME. Estes elementos gráficos não correspondem, necessariamente, a componentes gráficos (ex: campo texto, caixa de seleção), mas sim a um conjunto de componentes que definem um item da simulação. Por exemplo, o NetLogo possui vários campos de texto para definir as dimensões do ambiente—mostrados na Figura 1(b). O grupo destes componentes é considerado uma AME, haja visto que uma única linha de código seria suficiente para representá-los (ex: uma chamada de método com os parâmetros que definem as dimensões). Portanto, as seguintes regras foram estabelecidas para contar AMEs no NetLogo:

- uma AME para cada grupo de componentes *world*, *view* e *tick counter*; e
- uma AME para cada componente visual incluído no modelo com a finalidade de entrada de dados: *sliders*, *switches*, *choosers*, e *inputs*.

No caso do SeSAm, as convenções de contagem se restringem às AMEs, haja visto que não há linhas de código nesta plataforma. A contagem também tomou como base as regras de [Bettin 2002] para diagramas de classe: cada elemento visual que corresponderia a uma LoC é contabilizado como uma AME. Similarmente ao NetLogo, estes elementos gráficos não correspondem necessariamente a um componente do SeSAm (ex: elementos na árvore de funções), mas sim a um conjunto de componentes que definem um elemento do modelo que poderia ser expresso em uma linha de código. Funções comentadas não são consideradas. Desta forma, as seguintes regras foram estabelecidas para contar as AMEs no modelo *Sugarscape* do SeSAm.

- Uma AME é contabilizada para cada *resource*, *agent*, *world*, *user function*, *user type*, *variable*, *activity* (exceto a *emergency activity*), *transition*, *situation*.
- Funções utilizadas para criar comportamentos e compor outras funções seguem regras que consideram o tipo da função.
  - Funções simples, que não são compostas por outras (ex: funções de leitura/escrita), constituem uma AME cada.
  - Funções condicionais, tais como *If-Then-Else* e *SwitchOnEnum*, geram uma AME para a condição, e várias outras AMEs para as funções definidas nos blocos condicionais (ex: blocos *then* e *else*).
  - Funções de repetição, tais como *DoWith*, *ForElement* e *ForTimes*, geram uma AME para a condição de repetição, e várias outras AMEs para os comandos do bloco de repetição.
  - Funções que simplesmente agrupam outras, tais como *Block* e *Progn*, não constituem AMEs.

Considerando que há correspondência entre AMEs e LoCs, haja visto que uma AME corresponde a aproximadamente uma LoC [Bettin 2002], é possível considerar a soma destas métricas como o esforço de desenvolvimento em termos de artefatos produzidos. A seguir, apresentamos os resultados destas métricas para a simulação *Sugarscape*.

3.4. Resultados e Discussão

As simulações *Sugarscape* do NetLogo e SeSAm foram submetidas às regras de contagem para extração das métricas. Os resultados obtidos são apresentados na Tabela 1. As colunas indicam a quantidade de LoCs, de AMEs, e o esforço de desenvolvimento, que corresponde à soma de LoCs + AMEs. A coluna *diferença* mostra a diferença percentual entre o esforço SeSAm em relação ao esforço NetLogo.

Podemos verificar que com o SeSAm, o esforço de desenvolvimento é 35.6% maior do que com o NetLogo. Este resultado não é intuitivo, haja visto que o propósito do SeSAm é justamente facilitar a construção de simulações a partir de modelagem visual. De fato, a especificação com o SeSAm não requer LoCs, apenas AMEs.

Para elucidar as razões desta diferença, o gráfico da Figura 3 apresenta uma visão do esforço para desenvolver diferentes aspectos da simulação. Aspectos estruturais dizem respeito à estrutura do ambiente e dos agentes (ex: tipos e atributos). Aspectos dinâmicos referem-se aos processos que definem o comportamentos do ambiente e dos agentes. O aspecto de *setup* da simulação envolve a criação e inicialização do ambiente e dos agentes. A visualização refere-se aos comandos e funções destinados a atualizar a visualização da simulação. Por fim, o aspecto *outros* refere-se a funções e comandos utilitários, sem relação com os aspectos anteriores, mas que são usados na simulação (ex: geradores de valores aleatórios). Cada barra do gráfico da Figura 3 mostra a quantidade de LoCs e AMEs por aspecto, sendo que a soma destas é mostrada entre parêntesis. Legendas à direita das barras mostram qual plataforma requer maior esforço naquele aspecto, e qual a diferença percentual deste esforço em relação à outra plataforma.

Como pode ser visto, o SeSAm requer maior esforço para desenvolver aspectos estruturais do ambiente e dos agentes, da dinâmica dos agentes e de visualização. Ao compararmos os artefatos usados para implementar a simulação, notamos que a plataforma NetLogo fornece maior quantidade de artefatos prontos para uso. No caso da estrutura do ambiente, o NetLogo conta por padrão com um ambiente de grade com células,

Tabela 1. Comparação do Esforço de Desenvolvimento.

NetLogo			SeSAm			Diferença
LoCs	AMEs	Esforço*	LoCs	AMEs	Esforço*	SeSAm/NetLogo
69	4	73	0	99	99	+35.6%

*Esforço = LoCs + AMEs

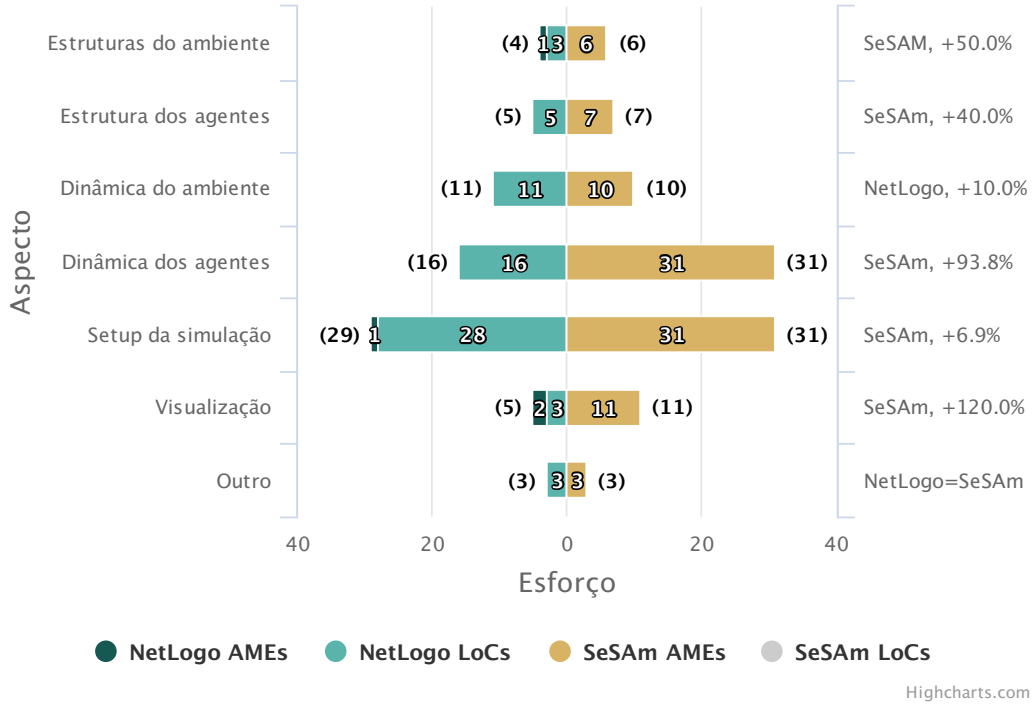


Figura 3. Esforço de Desenvolvimento por Aspecto da Simulação.

enquanto que no SeSAm é necessário definir as células como *resources*, demandando esforço de desenvolvimento 50.0% maior que o NetLogo. Quanto aos aspectos estruturais dos agentes, o SeSAm requer variáveis adicionais para armazenar a localização do agente e sua percepção, o que resulta em um esforço de desenvolvimento 40.0% maior. Já no NetLogo, estas variáveis são fornecidas de forma integrada, prontas para uso. No caso do aspecto dinâmica dos agentes, observou-se algo similar: o NetLogo fornece diversos comandos de forma integrada, para percepção e movimentação dos agentes; enquanto que no SeSAm é preciso definir manualmente as funções que desempenham estas ações, requerendo um esforço 93.8% maior que o NetlLogo. No aspecto de visualização é onde o SeSAm requer o maior grau de esforço em relação ao NetLogo: 120.0%. Isto também ocorre pelo fato de o NetLogo fornecer comandos de visualização prontos para uso.

Há aspectos onde a diferença entre o esforço com SeSAm e com NetLogo é pequena, ou inexistente. No aspecto *setup* da simulação, o esforço com o SeSAm é 6.9% maior pela necessidade de instanciar as células (*resources*), enquanto que o NetLogo faz isso automaticamente. No aspecto *outros*, não houve diferença. Por fim, no aspecto dinâmica do ambiente, o esforço com o NetLogo foi 10.0% maior que o SeSAm. Isto se deve ao fato de que no código NetLogo é preciso chamar explicitamente a função que avança o relógio da simulação, enquanto que no SeSAm isto acontece automaticamente.

Os resultados obtidos demonstram que, no caso da simulação *Sugarscape*, a plataforma NetLogo requer menor esforço para o desenvolvimento por conta das abstrações oferecidas, tanto para definir um ambiente composto por células, quanto para utilizar comandos de percepção, localização, movimentação e visualização. Apesar de o SeSAm proporcionar modelagem visual, não requerendo necessariamente um conhecimento em programação, ele não oferece estas abstrações, e o projetista precisa especificá-las por completo, resultando em maior esforço. Isto evidencia que o fator chave para permitir o desenvolvimento de simulações com o mínimo de esforço transcende o formato da linguagem (gráfica ou textual) e concentra-se nas abstrações disponíveis para uso. Esta constatação vai de encontro com os princípios de *model-driven development* (MDD), onde abstrações de domínio são disponibilizadas para uso na modelagem ou desenvolvimento. Trabalhos anteriores constataram que MDD de fato aumenta a produtividade, pois o esforço de desenvolvimento concentra-se no uso das abstrações disponíveis, em vez de nos comandos de baixo nível de linguagens de programação [Sprinkle et al. 2009].

Por fim, quanto aos trabalhos relacionados, a análise conduzida neste artigo é inovadora por abordar o aspecto esforço de desenvolvimento, não considerado previamente, mas que pode afetar a escolha da plataforma adotada.

4. Conclusões

Simulações baseadas em agentes normalmente são desenvolvidas sobre alguma plataforma de simulação, tendo em vista os recursos inerentes à área de agentes que tais plataformas oferecem. Além de características elementares, como a linguagem adotada, os recursos de desenvolvimento oferecidos e o sistema operacional exigido, o esforço requerido para desenvolver uma simulação pode ser um aspecto que afeta a escolha da plataforma adotada.

Neste artigo, avaliamos o esforço para desenvolvimento de simulações nas plataformas NetLogo e SeSAm em um estudo de caso com a simulação *Sugarscape*. As métricas utilizadas consideraram as quantidades de artefatos produzidos para implementar a simulação. Os resultados indicam que o esforço para desenvolver com o SeSAm é 35.6% maior que com o NetLogo, no contexto do estudo de caso realizado. Inicialmente, este resultado é contra-intuitivo, pois o SeSAm é uma plataforma que permite desenvolvimento de forma totalmente gráfica, não requerendo conhecimento de programação. No entanto, uma análise apontou que isto ocorre devido à maior abstração fornecido pelo NetLogo, que dispõe de uma ampla biblioteca de comandos que abstraem ações de percepção, movimentação e visualização. Isso indica que, mais importante do que prover novas alternativas de codificação e modelagem, é necessário prover entidades de primeira ordem que abstraiam comportamentos recorrentes, facilitando o desenvolvimento.

Avaliar o esforço para desenvolvimento de outras simulações é um passo importante para verificar como os resultados obtidos generalizam para além do estudo de caso. Esta análise poderia identificar qual plataforma requer menor esforço em cada classe de simulação. Outras dimensões de esforço podem ser futuramente consideradas, como por exemplo o esforço cognitivo (i.e., curva de aprendizagem) para utilizar estas plataformas. Também considera-se avaliar o esforço para desenvolvimento em plataformas que oferecem uma abordagem *model-driven development*, como por exemplo a INGENIAS Development Toolkit (IDK) [Pavón et al. 2006]. Considerando que um dos princípios

de *model-driven development* é disponibilizar várias abstrações de domínio para uso na modelagem, o esforço de desenvolvimento poderá ser ainda menor nestas plataformas.

Referências

[Albrecht 1979] Albrecht, A. J. (1979). Measuring application development productivity. In *Proceedings of the joint SHARE/GUIDE/IBM application development symposium*, volume 10, pages 83–92.

[Bettin 2002] Bettin, J. (2002). Measuring the potential of domain-specific modeling techniques. In *Second Domain-Specific Modelling Languages Workshop (OOPSLA)*, pages 39–44, Seattle, Washington.

[Boehm et al. 1995] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., e Selby, R. (1995). Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1(1):57–94.

[Castle e Crooks 2006] Castle, C. J. E. e Crooks, A. T. (2006). Principles and concepts of agent-based modelling for developing geospatial simulations. Technical report, University College London.

[Challenger et al. 2015] Challenger, M., Kardas, G., e Tekinerdogan, B. (2015). A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Software Quality Journal*, pages 1–41.

[Epstein e Axtell 1996] Epstein, J. e Axtell, R. (1996). *Growing Artificial Societies Social Science From The Bottom Up*. MIT Press.

[Garro e Russo 2010] Garro, A. e Russo, W. (2010). easyABMS: A domain-expert oriented methodology for agent-based modeling and simulation. *Simulation Modelling Practice and Theory*, 18(10):1453–1467.

[Klügl e Bazzan 2012] Klügl, F. e Bazzan, A. L. C. (2012). Agent-based modeling and simulation. *AI Magazine*, 33(3):29–40.

[Klügl et al. 2006] Klügl, F., Herrler, R., e Fehler, M. (2006). SeSAm: Implementation of agent-based simulation using visual programming. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1439–1440, New York, NY, USA. ACM.

[Kravari e Bassiliades 2015] Kravari, K. e Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11.

[Luke et al. 2005] Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., e Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527.

[Macal e North 2014] Macal, C. e North, M. (2014). Introductory tutorial: Agent-based modeling and simulation. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 6–20, Piscataway, NJ, USA. IEEE Press.

[Minar et al. 1996] Minar, N., Burkhart, R., Langton, C., e Askenazi, M. (1996). The swarm simulation system: A toolkit for building multi-agent simulations. Technical report, Santa Fe Institute, Santa Fe.

[Nikolai e Madey 2009] Nikolai, C. e Madey, G. (2009). Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12(2):2.

[North et al. 2006] North, M. J., Collier, N. T., e Vos, J. R. (2006). Experiences creating three implementations of the Repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 16(1):1–25.

[OpenABM Consortium 2017] OpenABM Consortium (2017). OpenABM. `http://www.openabm.org`, Acesso em: Abr/2017.

[Pavón et al. 2006] Pavón, J., Gómez-Sanz, J., e Fuentes, R. (2006). Model driven development of multi-agent systems. In Rensink, A. e Warmer, J., editors, *Model Driven Architecture – Foundations and Applications*, volume 4066 of *Lecture Notes in Computer Science*, pages 284–298. Springer Berlin Heidelberg.

[Railsback et al. 2006] Railsback, S. F., Lytinen, S. L., e Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *SIMULATION*, 82(9):609–623.

[Rollins et al. 2014] Rollins, N. D., Barton, C. M., Bergin, S., Janssen, M. A., e Lee, A. (2014). A computational model library for publishing model documentation and code. *Environmental Modelling & Software*, 61(0):59 – 64.

[Sprinkle et al. 2009] Sprinkle, J., Mernik, M., Tolvanen, J. P., e Spinellis, D. (2009). Guest editors' introduction: What kinds of nails need a domain-specific hammer? *IEEE Software*, 26(4):15–18.

[Wilensky 1999] Wilensky, U. (1999). NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.