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

JOSUÉ FILIPE KEGLEVICH DE BUZIN

# *Concerns* como Componentes Reutilizáveis em Simulações baseadas em Agentes de Software

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

Orientador: Profª. Drª. Ingrid Nunes
Co-orientador: Dr. Fernando Santos

Porto Alegre
2019

## RESUMO

A modelagem e simulação baseada em agentes, ou *Agent-based Modeling and Simulation* (ABMS), é um paradigma de simulação que usa agentes simulados para reproduzir e estudar eventos específicos. A partir deste modelo, é possível observar o comportamento de um agente inserido em um conjunto de indivíduos autônomos. Entre as áreas de aplicação pode-se citar desastres, gerenciamento ambiental, mobilidade e transporte. Apesar do seu potencial, muitas ferramentas disponíveis atualmente que suportam ABMS não possuem recursos para facilitar o reuso dos elementos de uma simulação. Há uma abordagem de desenvolvimento dirigida a modelos focada em ABMS com potencial para o reuso. Esta abordagem inclui uma linguagem específica de domínio chamada ABStractLang e a ferramenta que implementa a linguagem, denominada ABStractme. Esta monografia propõe a extensão desta abordagem, a partir do refinamento de um conceito previamente definido - *concern* - para a modularização de simulações. Este refinamento visa o reuso de *concerns* desenvolvidos em diversas simulações, a partir da definição de sua interface. Por fim, a ferramenta ABStractme, que dá suporte à abordagem dirigida a modelos, foi evoluída para incorporar a extensão proposta.



**Palavras-chave:** Plataforma de Agentes. Modelagem e Simulação baseada em Agentes. Linguagem Específica de Domínio. UFRGS.

## ABSTRACT

Agent-based modeling and simulation (ABMS) is a simulation paradigm where simulated agents are used to reproduce specific events so that they can be studied. With this model it is possible to observe the behavior of an agent inserted in a set of autonomous individuals. Areas of application include disasters, environmental management, mobility and transportation. Despite its potential, most tools currently available that support ABMS do not have the capabilities to facilitate the reuse of the elements of a simulation. There is a model-driven approach to development focused on ABMS with potential for reuse. This approach includes a domain-specific language called ABStractLang and the tool that implements the language, called ABStractme. This monograph proposes the extension of this approach with the refinement of a previously defined concept - emph concern - for the modularization of simulations. This refinement aims at the reuse of emph concerns developed in several simulations, from the definition of its interface. Also, the tool ABStractme that supports the model-driven approach has evolved to incorporate the extension proposed.



**Palavras-chave:** Agent Platform. Agent-Based Modeling and Simulation. Domain Specific Language. UFRGS.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ABMS | Agent-Based Modeling and Simulation |
| BDI | Belief-Desire-Intention |
| DSL | Domain-Specific Language |
| DTD | Document Type Definition |
| MAS | Multi-Agent System |
| MDD | Model-Driven Development |
| UML | Unified Modeling Language |
| XML | eXtensible Markup Language |

**SUMÁRIO**

# 1 INTRODUÇÃO

Com o advento da tecnologia da informação, surgiu uma oportunidade para explorar e entender fenômenos complexos através de abordagens computacionais. Uma das abordagens é o conceito de simulação, que é a execução de um modelo de fenômeno feito por software que visa reproduzir um evento real. Uma de suas vantagens é a diminuição de riscos, tais como a necessidade de observar os eventos *in loco* e fazer predições a partir de uma quantidade insuficiente de amostras de dados. Outra vantagem é a possibilidade de fazer descobertas sobre o comportamento destes eventos sem a necessidade de arcar com um custo significativo de tempo, em comparação ao tempo de observação necessário no caso de experimentos feitos *in loco*.

A modelagem e simulação baseada em agentes, ou *Agent-based Modeling and Simulation* (ABMS) (KLÜGL; BAZZAN, 2012), é um paradigma para modelagem de simulações largamente utilizado no estudo de cenários ou eventos, que são de interesse para pesquisas. Um agente é uma entidade individual simulada que possui um ou mais objetivos, atuando de forma autônoma e flexível. No contexto de ABMS, esses agentes estão inseridos em certo ambiente e reagem ao interagir com outros elementos, que podem ser outros agentes, entidades e o próprio ambiente, por exemplo. Estes outros elementos influenciam na tomada de decisão do agente. A partir da avaliação do comportamento destes agentes, durante a execução da simulação, é possível estudar um evento possivelmente análogo a um fenômeno que ocorre no mundo real.

Em uma simulação com agentes, o fenômeno a ser observado emerge a partir das interações entre os diversos elementos do ambiente. Pode-se exemplificar ABMS usando uma simulação de presa e predador. Cada agente predador tem a capacidade de se reproduzir e tem como objetivo consumir os agentes que representam as presas. As presas se reproduzem e consomem o pasto, que é um atributo do ambiente. O agente é influenciado pelo ambiente, pois possui um sensor que detecta qual o ponto próximo no mapa onde se encontra o pasto. Após detectar este ponto, o agente se move para o mesmo. A partir da execução desta simulação, é possível observar a velocidade com que os predadores consomem as presas em relação à taxa de reprodução das mesmas. Neste caso, seria possível prever quando ocorreria um fenômeno de extinção em massa de indivíduos por superpopulação e falta de alimento para o consumo.

Um dos maiores empecilhos para o uso de ABMS é a necessidade de desenvolver simulações em que a modelagem de agentes seja feita de forma simples e rápida. Galán

et al. (2009) avaliaram o processo de modelagem e concluíram que exige do projetista da simulação um conhecimento prévio de programação para modelar a mesma, o que leva a um custo de tempo no aprendizado, considerando que muitos especialistas de domínio não possuem experiência na área de programação. Por exemplo, no ambiente de simulação NetLogo, largamente utilizado, é preciso se familiarizar com a sua linguagem específica de programação antes de modelar as simulações. Para facilitar o acesso à modelagem de simulações por qualquer pessoa, quer tenha conhecimento em programação ou não, foi desenvolvida uma abordagem *Model-Driven Development* (MDD) chamada MDD4ABMS, que inclui a linguagem específica de domínio ABStractLang (SANTOS, 2018; SANTOS, 2019) e sua implementação na ferramenta ABStractme (MOREIRA et al., 2017). O objetivo da MDD4ABMS é permitir que o público em geral possa modelar as simulações em um ambiente gráfico, através da movimentação de componentes, sem a necessidade de aprender a programar. Se o projetista desejar executar a simulação em uma plataforma como o NetLogo, é possível gerar o código automaticamente para a execução imediata na plataforma (SANTOS; NUNES; BAZZAN, 2017a; SANTOS; NUNES; BAZZAN, 2017b; SANTOS; NUNES; BAZZAN, 2017c; SANTOS; NUNES; BAZZAN, 2017d).

Na abordagem MDD4ABMS, o modelo de simulação foi separado em elementos denominados *concerns*, para facilitar a modularização e a escalabilidade. Os *concerns* são uma abstração de elementos da simulação que decompõem a mesma em camadas conceituais. Sua função é agrupar a especificação de parâmetros, entidades, agentes com seus atributos e sua habilidade para tomar decisões conforme um interesse específico. O agrupamento ocorre a critério do projetista da simulação. Cada *concern* tem um diagrama próprio, que representa uma visão parcial do modelo, para facilitar a escalabilidade.

Simulações diferentes utilizam *concerns* que podem ser similares, o que leva o projetista das mesmas a remodelar elementos comuns a diversas simulações. Neste caso, ocorre um custo de tempo desnecessário na modelagem, pois é preciso modelar novamente estes elementos. Ao permitir o reuso de *concerns*, é possível reduzir o esforço na modelagem de simulações. O reuso também aumenta a confiabilidade, visto que os elementos precisam passar por diversos testes e correções até que possam ser reusáveis. Apesar da importância do reuso de *concerns* para reduzir o esforço no desenvolvimento de simulações, a abordagem MDD4ABMS atualmente não oferece essa possibilidade.

Com base na lacuna apontada com relação ao reuso limitado de *concerns* em MDD4ABMS, este trabalho propõe uma extensão do conceito de *concerns* para viabi-

lizar o seu reuso em diferentes simulações. As contribuições deste trabalho são: (i) um estudo de trabalhos voltados à modularidade e reuso de elementos frequentemente utilizados em simulações com agentes; (ii) refinamento da definição do *concern* a partir dos estudos realizados; (iii) extensão da ABStractLang e a implementação dos *concerns* reusáveis na ferramenta ABStractme, com a apresentação de uma proposta de alterações na geração automática de código; e (iv) estudo de caso com duas simulações similares, para verificar se houve benefícios no reuso de *concerns*.

A monografia está estruturada da seguinte forma. No Capítulo 2 é apresentada uma visão geral do paradigma ABMS juntamente com a abordagem escolhida para ser estendida neste trabalho. No Capítulo 3 é feita uma exploração de trabalhos relacionados. No Capítulo 4 são apresentadas a definição e a interface dos *concerns* reutilizáveis. No Capítulo 5 são apresentadas as modificações feitas na ferramenta ABStractme para alcançar o objetivo. No Capítulo 6 é mostrado o estudo de caso feito para validar as modificações propostas na ferramenta ABStractme. Por fim, no Capítulo 7 são apresentadas as conclusões do trabalho.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, apresenta-se uma visão geral da modelagem e simulação baseada em agentes (ABMS) e plataformas disponíveis no mercado que ajudam na modelagem de simulações. Também, é feita uma introdução à linguagem específica de domínio ABStractLang e é feito um resumo da ferramenta ABStractme, que são utilizados como base para este trabalho.

## 2.1 ABMS

A modelagem e simulação baseada em agentes (ABMS) (KLÜGL; BAZZAN, 2012), é um paradigma para modelagem de simulações que usa agentes autônomos e sistemas multiagente para reproduzir, analisar, predizer e explorar fenômenos. É usada em diversas áreas de aplicação. Entre elas pode-se citar desastres, gerenciamento ambiental, mobilidade e transporte, epidemiologia e economia (MACAL; NORTH, 2014). Conforme Macal and North (2014), seu uso deve-se ao fato de que pode incorporar explicitamente a complexidade de comportamentos individuais e interações que existem em cenários do mundo real, tornando as simulações mais realistas. Além disso, agentes têm a capacidade para aprender e evoluir conforme mudanças no seu ambiente.

O principal objetivo da ABMS é elucidar o comportamento de sistemas complexos. Conforme Klügl and Bazzan (2012), estes sistemas são compostos de três elementos: *ambiente*, *agentes* e *interações* entre os agentes e o ambiente. Neste contexto, os agentes estão situados em um ambiente, que podem perceber e modificar com suas ações. Os agentes são elementos individuais simulados e autônomos, que possuem sensores e atuadores para facilitar a interação com o ambiente. Suas características principais são a capacidade de ter um ou mais objetivos e poder aprender e evoluir com o tempo. Existem entidades que, apesar de similares aos agentes, não possuem autonomia e se comportam como objetos do ambiente, com os quais os agentes podem interagir. Por exemplo, em uma simulação de desastre ambiental, possíveis entidades são centros de socorro e os agentes são as pessoas vítimas do desastre que se refugiam nestes centros.

A modelagem da simulação envolve especificar um modelo que representa o sistema complexo que se deseja estudar. Durante a concepção da simulação, é necessária a contribuição de especialistas do domínio, com conhecimento mais detalhado da área em que o sistema está inserido, para produzir uma simulação com maiores chances de ser fiel

à realidade. A simulação em si ocorre ao se executar por certo período o modelo projetado anteriormente. Durante a execução, ocorrem repetidas interações entre os agentes e as entidades no ambiente. É a partir dos resultados obtidos durante a execução da simulação que se pode analisar o comportamento dos agentes.

## 2.2 Plataformas para a Modelagem de Simulações

Existem diversas plataformas que permitem a modelagem de simulações com agentes. Estas plataformas são brevemente detalhadas a seguir.

Swarm[1] é uma linguagem e conjunto de ferramentas para uso em domínios científicos. É a primeira plataforma criada para modelagem de simulações (KRAVARI; BASSILIADES, 2015), possuindo ferramentas que podem ser úteis na modelagem de qualquer simulação com agentes, mas que não focam em domínios específicos de simulação (RAILSBACK; LYTINEN; JACKSON, 2006). A Swarm permite a construção de simulações e a coleta de dados das mesmas para análise. Além disso, é preciso saber programação para usá-la e não há uma representação padrão do ambiente (KLÜGL; BAZZAN, 2012).

O *Recursive Porous Agent Simulation Toolkit*, ou Repast[2], é focado no domínio de ciências sociais e inclui ferramentas específicas para tal. Tem o objetivo adicional de facilitar a modelagem por usuários inexperientes, para isso possui um modelo com interfaces que permitem o uso de menus e código Python para construir modelos (RAILSBACK; LYTINEN; JACKSON, 2006). Baseada em Java, esta plataforma disponibiliza uma biblioteca de classes para as tarefas mais comuns associadas a uma implementação de simulação (KLÜGL; BAZZAN, 2012).

MASON[3] é uma alternativa mais rápida ao Repast, que foca em modelos com alta demanda computacional, onde é necessária a execução de simulações com uma quantidade significativa de agentes e muitas interações. Possui ferramentas que podem ser úteis na modelagem de diversas simulações com agentes. Esta plataforma pode ser insuficiente dependendo do domínio específico de simulação de interesse do projetista (RAILSBACK; LYTINEN; JACKSON, 2006). É uma biblioteca baseada em Java, que requer conhecimento em programação.

[1] <http://www.swarm.org/wiki/Main_Page>
[2] <https://repast.github.io/index.html>
[3] <https://cs.gmu.edu/~eclab/projects/mason/>

NetLogo[4] é um ambiente de desenvolvimento que facilita o processo de modelagem e execução de simulações (RAILSBACK; LYTINEN; JACKSON, 2006) e é focado na facilidade de aprendizado e uso. Por esta razão, apesar da modelagem ser feita por programação, a linguagem de alto nível do NetLogo possui uma sintaxe própria com alta abstração de conceitos, o que simplifica a modelagem. O projetista da simulação pode executar a mesma para observar o comportamento dos elementos após a compilação do código. A plataforma é versátil pois possui a capacidade para a modelagem de simulações de eventos relativos a diversas áreas de aplicação. Além disso, possui uma extensa documentação e uma biblioteca com modelos pré-existentes.

Das plataformas apresentadas, Railsback, Lytinen and Jackson (2006) concluíram que a plataforma NetLogo possui as melhores vantagens no uso em relação às outras. Entre suas vantagens está o fato de exigir menos conhecimento técnico em programação do que as outras plataformas mencionadas. Outra vantagem está na possibilidade do projetista visualizar a simulação em execução, o que pode auxiliar usuários, sem prévio conhecimento sobre o assunto, a entender o que é e como funciona ABMS. Por estes motivos, foi escolhida para auxiliar na visualização e execução das simulações a que este trabalho se refere. A ferramenta ABStractme, descrita a seguir, suporta a geração de código automática para a plataforma NetLogo, reduzindo assim a necessidade de digitar código durante a modelagem da simulação.

## 2.3 ABStractLang e ABStractme

Um dos empecilhos para o uso de ABMS está na complexidade de sua modelagem, pois envolve vários elementos que interagem entre si. Para o sucesso da modelagem e simulação é preciso que haja a participação de especialistas do domínio, que entendem do sistema a ser simulado, e de especialistas técnicos, que efetivamente implementam a proposta. É necessário que haja uma comunicação constante entre os diferentes especialistas. Como resultado, há uma alta complexidade e dificuldades no processo de implementação.

Para diminuir o problema, Santos, Nunes and Bazzan (2017a) propuseram aumentar o nível da abstração da ABMS. Desta forma, quem não possui conhecimento em programação pode modelar simulações sem necessidade do envolvimento constante de especialistas técnicos. A partir desta proposta foi concebida a linguagem específica de domínio ABStractLang (SANTOS; NUNES; BAZZAN, 2017c; SANTOS, 2018; SAN-

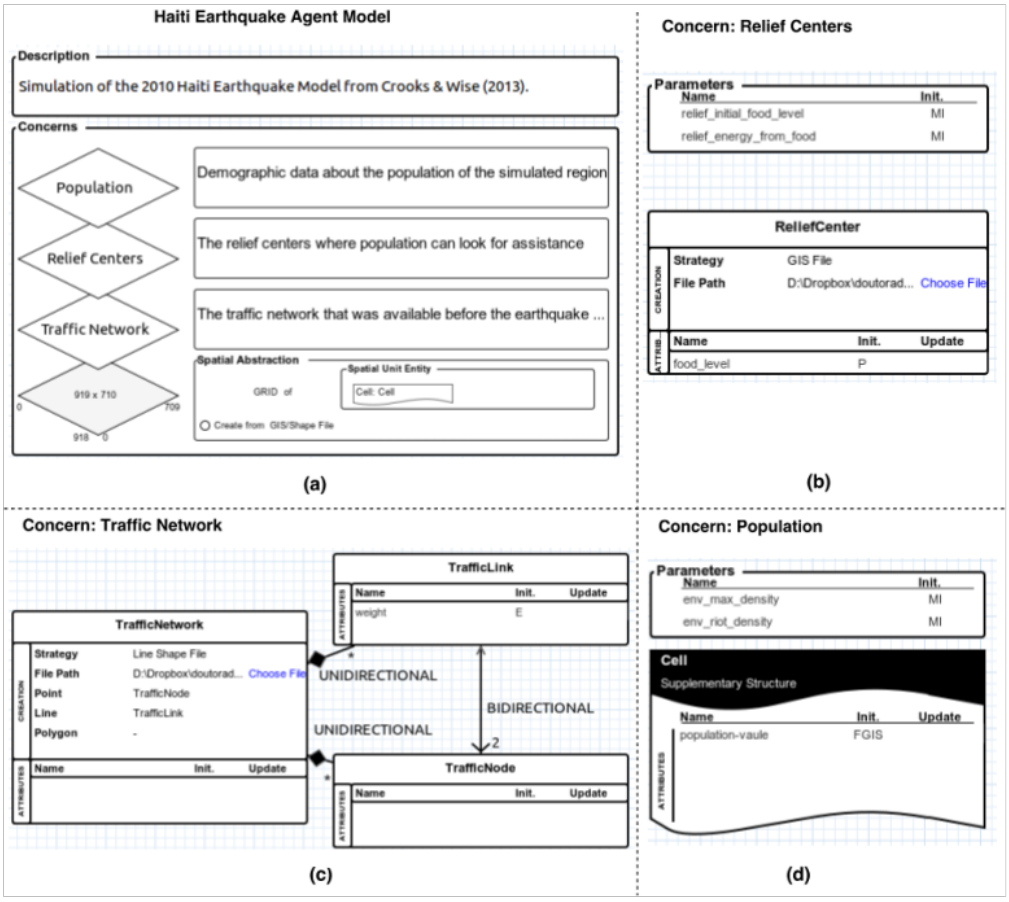
[4] <http://ccl.northwestern.edu/netlogo/>

TOS, 2019).

Uma linguagem específica de domínio, ou DSL (MERNIK; HEERING; SLOANE, 2005), é concebida para facilitar a abstração em um domínio específico. No caso da ABStractLang, o domínio está nas simulações com agentes, e a função da linguagem é abstrair os elementos da simulação para que possam ser identificados como módulos claramente definidos, simplificando a modelagem para que qualquer um possa modelar e executar o sistema complexo sem ajuda.

O consenso na definição dos modelos da linguagem e do nível de abstração dos mesmos foi feito a partir do estudo de diversas simulações. Os modelos foram definidos com base nos padrões identificados nas diferentes simulações estudadas. A linguagem possui uma sintaxe abstrata e uma concreta (SANTOS, 2018). A sintaxe abstrata é a estrutura interna das abstrações de domínio, representada como um metamodelo. A sintaxe concreta é a representação visual dos modelos. A Figura 2.1 apresenta uma exemplificação da sintaxe concreta contendo os principais elementos da ABStractLang.

Figura 2.1: Sintaxe concreta da linguagem



Na ABStractLang, é usado o conceito de *concerns* para facilitar a modularidade em ABMS. *Concern*, de forma similar ao que é definido na orientação a aspectos, é uma unidade conceitual única. O seu objetivo é modularizar conceitos, separando componentes do todo e agrupando conforme interesses específicos. A Figura 2.1(a) mostra uma visão parcial do diagrama que representa a visão global da simulação, cujo objetivo é mostrar o propósito da simulação sendo modelada. No diagrama é possível observar o título, a descrição da simulação e a pilha de *concerns*. Cada *concern* tem o seu diagrama, como mostrado na Figura 2.1(b)-(d), onde são especificadas as entidades com seus atributos e os parâmetros. A representação visual dos mesmos é análoga aos diagramas de classes da Linguagem de Modelagem Unificada, ou UML. A Figura 2.1(d) mostra um exemplo de estrutura suplementar, que pode conter seus próprios atributos.

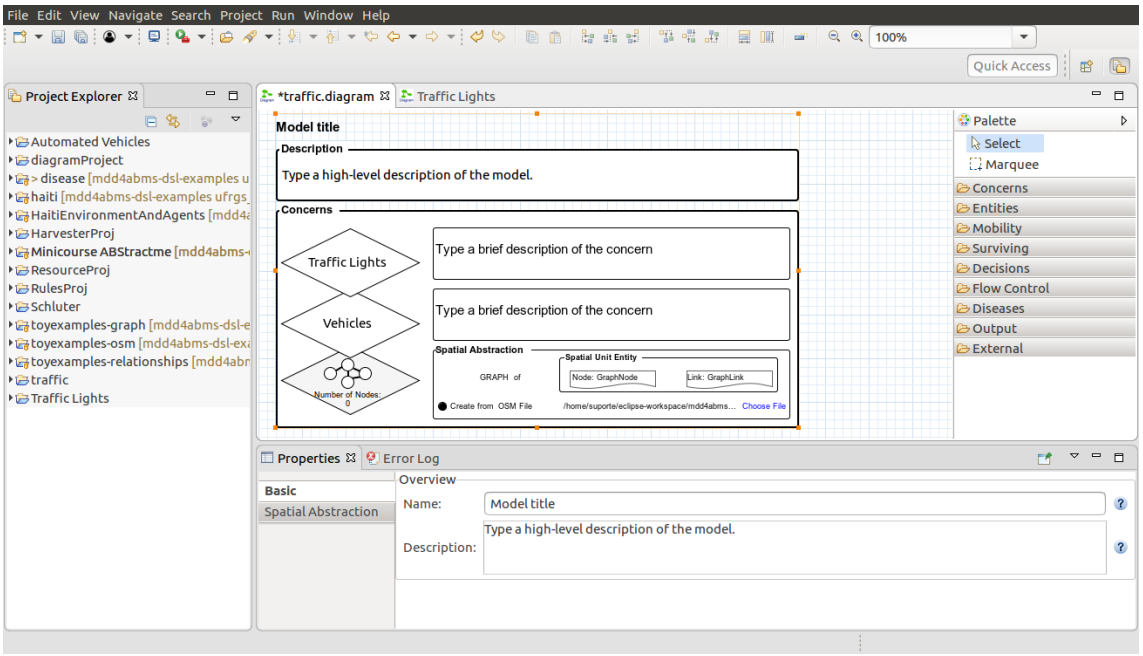
A ABStractLang faz parte da MDD4ABMS, uma abordagem MDD para ABMS (SANTOS; NUNES; BAZZAN, 2017b; SANTOS, 2019). A abordagem foca na abstração dos detalhes de modelos para simplificá-los e reduzir o tempo de desenvolvimento. O objetivo é aumentar a produtividade, diminuindo a quantidade do código que precisa ser desenvolvido manualmente para implementar a simulação. Assim, a MDD4ABMS disponibiliza geradores de código para gerar automaticamente o código fonte dos elementos do modelo. Para o desenvolvimento da abordagem foi feita uma análise de diferentes domínios de simulação. A partir desta análise foi concebido o metamodelo da sintaxe abstrata juntamente com a linguagem, além das transformações de modelo para código.

Para a MDD4ABMS alcançar seu objetivo, a ferramenta ABStractme foi implementada como suporte para a linguagem ABStractLang (MOREIRA et al., 2017). A visão geral da ferramenta pode ser vista na Figura 2.2. A ABStractme é um plugin para a IDE Eclipse[5] e possui uma interface gráfica, além de um gerador de código para a plataforma NetLogo. A ferramenta usa os frameworks Graphiti, um conjunto de bibliotecas voltadas à criação e manutenção da interface gráfica, e Xpand, que permite a geração de código a partir dos elementos construídos na interface gráfica.

Na Figura 2.2 é possível visualizar, da esquerda para a direita: (i) *aba de exploração do projeto*, padrão da IDE Eclipse, onde podem ser observados os arquivos com os diagramas; (ii) *aba do diagrama com a visão global da simulação*, com os elementos definidos na sintaxe concreta da linguagem; (iii) *paleta*, onde podem ser escolhidos os elementos para inserir nos diagramas da simulação; e (iv) *aba de propriedades padrão da IDE Eclipse*, com os campos para editar o nome e a descrição da simulação.

[5] <http://www.eclipse.org/>

Figura 2.2: Visão geral da ferramenta ABStractme

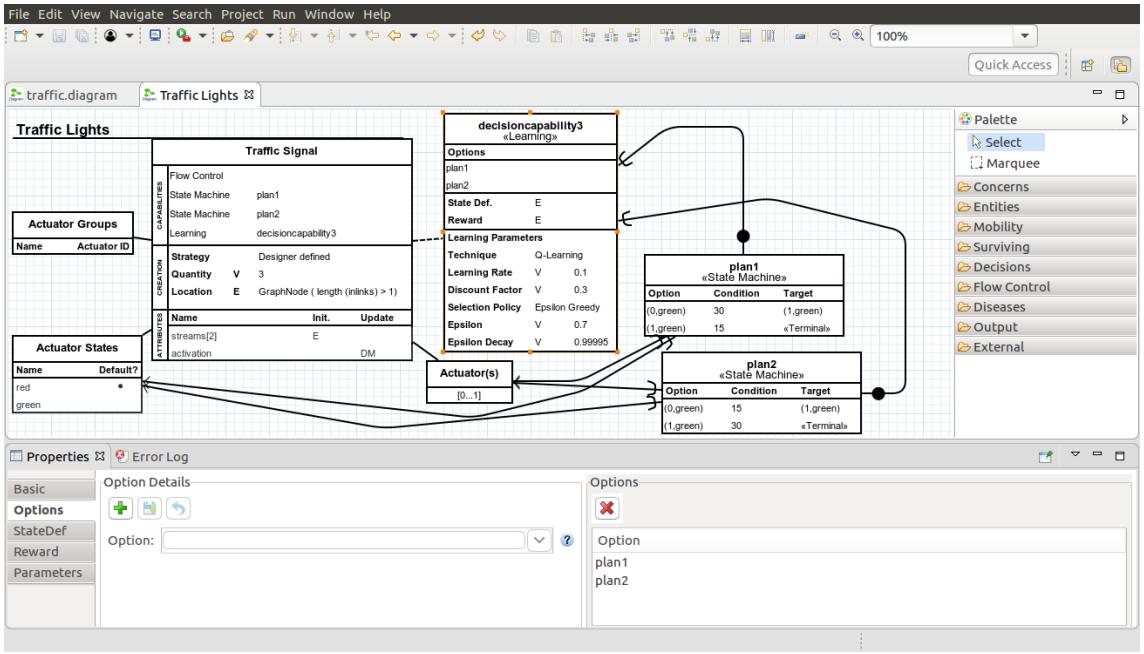



A ferramenta tem suporte para a modelagem dos seguintes elementos de simulação definidos na ABStractLang: (i) abstração espacial do ambiente onde habitam os agentes, que é o elemento base da simulação; (ii) *concerns*; (iii) parâmetros dos *concerns*, que são atributos relacionados ao elemento base da simulação; (iv) estruturas suplementares dos *concerns*, herdadas da abstração espacial, que podem conter seus próprios atributos; (v) entidades com seus atributos e relacionamentos; e (vi) agentes com suas diversas habilidades, que incluem definição de objetivos, movimentar-se, contrair doenças, tomar decisões e aprender comportamentos. A diferença entre entidades e agentes está no fato de que entidades são objetos estáticos da simulação, sem autonomia. O agente é uma entidade que possui a capacidade de ter autonomia para ter objetivos e tomar decisões.

A modelagem da simulação é feita em um diagrama que possui um conjunto de *concerns*, como mostrado na Figura 2.2 e cada *concern* tem seu subdiagrama onde podem ser inseridos e modificados os agentes, entidades, etc, como mostrado na Figura 2.3. Ao ser modelados neste diagrama, tanto agentes quanto entidades podem estabelecer conexões entre si no mesmo *concern* ou entre *concerns*, e os atributos podem ter uma relação de dependência com elementos de outros *concerns*. Como o projetista pode gerar relações de dependência entre *concerns*, o reuso dos mesmos é dificultado por causa deste acoplamento que surge a partir destas relações.

A principal limitação dos *concerns* está no fato de que sua definição está limitada ao agrupamento de elementos contidos no subdiagrama. Estes componentes agrupados no *concern* estão unicamente identificados como parte da simulação em geral, não há uma forma de referenciá-los como parte do *concern*. Além disso, elementos de um *concern* podem referenciar elementos de outros *concerns* sem qualquer restrição. Por causa da existência dessas lacunas, não é possível reutilizar os elementos definidos no *concern* em outra simulação sem ter de criar e configurar os mesmos novamente. Um exemplo onde o reuso pode ser vantajoso aparece na Figura 2.3, que apresenta um *concern* que representa o uso de sinais de trânsito como agentes para o controle do fluxo de veículos numa simulação de tráfego de automóveis. Este mesmo *concern* pode ser útil em outro contexto, como uma simulação de terremoto, onde é necessário monitorar o fluxo de veículos durante a evacuação dos sobreviventes. Para que o reuso deste *concern* seja viável, é necessário definir uma interface para os *concerns* que corrija as lacunas identificadas e diminua o acoplamento dos mesmos.

Figura 2.3: Diagrama de um *concern*



## 2.4 Considerações Finais

A proposta deste trabalho é estender a definição do *concern*, que está situado no contexto de ABMS. As plataformas que o projetista tem ao seu alcance para a modelagem de sistemas multiagente evidenciam a necessidade de conhecimento técnico no processo de modelagem e execução das simulações. A proposta da abordagem MDD4ABMS é facilitar este processo através da linguagem ABStractLang e sua implementação na ferramenta ABStractme. Utilizando a linguagem e a ferramenta como base, é possível refinar a definição do *concern* para diminuir o seu acoplamento. Como resultado, o *concern* se torna reusável e facilita ainda mais o processo de modelagem e execução de simulações.

# 3 TRABALHOS RELACIONADOS

O foco deste trabalho é dar suporte ao desenvolvimento de simulações baseadas em agentes através de técnicas que promovem o reuso de software. Desta forma, neste capítulo, são apresentadas, discutidas e comparadas abordagens nesta direção, tanto no contexto de ABMS quanto no de sistemas multiagente em geral. Em relação aos estudos são discutidos os pontos: (i) como é organizado o comportamento dos agentes; (ii) a possibilidade de usar a solução proposta para construir simulações ou sistemas em diversos domínios específicos e, a partir deste ponto, verificar a possibilidade de reusar os agentes entre estes domínios; (iii) explicitar o nível de conhecimento técnico necessário para implementar a solução e a facilidade de uso da mesma; e (iv) como foi avaliada a proposta.

## 3.1 Modularidade e Reuso em ABMS

Scerri et al. (2010) realizaram um estudo com foco na modelagem de uma simulação que explora a adaptação de uma população a mudanças climáticas. Esta simulação possui uma alta complexidade pois precisa do envolvimento de especialistas em diferentes domínios. O desenvolvimento da simulação envolve o uso de módulos individuais reutilizáveis, que podem ser pré-existentes ou implementados em paradigmas diferentes. Neste caso, cada especialista de domínio implementa os seus módulos separadamente do resto da simulação.

Um módulo de doença e um módulo de imigração afetam a mesma população, ou seja, há um compartilhamento de dados. Neste caso, um módulo, além de ser independente, não deve ter informações sobre a população diretamente. Para que o compartilhamento seja possível e os módulos continuem independentes, há um nível global onde estes dados podem ser acessados por diferentes módulos.

A interface para execução das simulações é uma versão modificada do *game* Civilization IV[1], neste caso a facilidade de uso ocorre somente ao executar a simulação. A modelagem da simulação necessita de conhecimento técnico e a proposta mostra alguns algoritmos entretanto não há um foco nos detalhes de como foi feita a implementação. Apesar do trabalho promover a modularidade em ABMS, reuso de agentes é apenas uma das consequências da adoção da abordagem proposta e ela não traz detalhes sobre como

[1] <https://sourceforge.net/projects/civ4mods/>

seria este reuso. O objetivo principal da abordagem é facilitar o compartilhamento de variáveis globais em simulações complexas e escaláveis, sem que ocorram conflitos.

Para analisar a eficácia da proposta foi feita uma avaliação de desempenho que consiste na coleta de dados relativos ao tempo de execução do modelo. Foram comparados os tempos de execução ao usar todos os agentes num único modelo e ao usar os módulos individuais apresentados na proposta. Nos resultados o tempo de execução da proposta foi menor, logo para os autores a mesma é considerada escalável em termos de tempo de execução.

Com o objetivo similar de facilitar a escalabilidade de simulações a partir da separação entre o agente e o resto do sistema, Al-Zinati et al. (2013) propuseram o framework *Dynamic Information Visualization of Agent systems*, ou DIVAs, que permite implementar uma variedade de simulações em diversos domínios de simulação. A arquitetura conta com uma coleção de classes abstratas e componentes de software reusáveis em Java que podem ser usados como bibliotecas de código. No caso, o ambiente da simulação é uma entidade de primeira classe que está desacoplada dos agentes, o que leva a uma separação clara entre os elementos da simulação que facilita a modularização e a escalabilidade. Um agente é uma implementação da sua classe abstrata que faz parte do DIVAs.

É possível construir o ambiente usando uma interface gráfica, similar a um editor de mapas de um *video game*. A interface gráfica também permite acessar e modificar os agentes e propriedades do ambiente dinamicamente, sem interromper totalmente a execução da simulação. Como exemplo, um agente pode parar a sua ação atual e decidir mudar seu objetivo a partir de uma mudança na opção de editor disponível ao observador. O projetista da simulação também tem o poder para remover objetos que estão no ambiente ao pausar a execução da mesma. A facilidade de uso, neste caso, é alta e está relacionada com a interface gráfica.

É possível usufruir das vantagens no uso das bibliotecas fornecidas se as mesmas forem o suficiente para modelar a simulação desejada, ou seja, se estas bibliotecas podem prover o tipo de comportamento necessário aos agentes que serão modelados. Neste caso, a implementação é simplificada e ocorre o reuso a partir da utilização das bibliotecas. Caso contrário, será necessário um estudo das bibliotecas e implementação de diversas mudanças. Neste caso, a complexidade na modelagem da simulação aumenta de forma significativa e é necessário possuir conhecimento técnico para preencher as lacunas. Desta forma, o reuso dos agentes está sujeito ao domínio específico da simulação que será modelada.

Os autores da proposta da DIVAs fizeram uma avaliação com foco no desempenho, com o objetivo de averiguar a escalabilidade. Foi analisado o tempo de execução de uma simulação com uma quantidade significativa de agentes e, a partir do resultado, o autores concluíram que não houveram problemas significativos de desempenho.

Em ABMS, agentes normalmente estão ligados ao ambiente e não podem ser usados em outros cenários. Apeldoorn (2015) introduz um framework focado no reuso de agentes em diferentes cenários sem precisar mudar a interface. A proposta ajuda na observação de como o agente se adapta a problemas diferentes. Ela é baseada na linguagem de modelagem *AbstractSwarm*[2], que foca em problemas de logística, e, de forma similar às propostas anteriores, faz separação entre o ambiente e a modelagem do agente.

O framework possui dois componentes básicos: agentes e estações. As estações são apresentadas como locais onde os agentes podem se encontrar. O cenário é um grafo onde estações são nodos e as relações entre eles são vértices, sendo que os agentes podem percorrer por este grafo. A visualização da simulação está no estudo do gráfico criado a partir de uma interface gráfica, que deixa explícito o comportamento dos agentes e como se relacionam com as estações e entre si.

A proposta apresenta o caso de uso de um calendário escolar, em que existem pontos de vista diferentes, definidos como perspectivas na proposta, que no caso são o pontos de vista diferentes dos professores, classes e salas. Um segundo caso de uso é o de máquinas que analisam produtos sendo produzidos por trabalhadores numa fábrica, em que as perspectivas são o ponto de vista dos trabalhadores e dos produtos. Ambos casos envolvem agendamento, que é feito por agentes que são similares e podem ser reusados, apesar das simulações estarem em diversos domínios de simulação. Em ambos os casos de uso foi feita uma avaliação de desempenho que consiste em comparar o tempo de espera dos agentes até selecionar o alvo com uma implementação em que esta seleção é aleatória. Na implementação proposta o tempo de espera é menor, logo a política adaptativa dos agentes proposta é mais efetiva do que uma seleção aleatória.

Entre as vantagens da abordagem estão criar e rodar simulações considerando apenas o cenário e implementação genérica dos agentes para reuso em qualquer cenário modelado. É importante ressaltar que, apesar da linguagem *AbstractSwarm* possuir uma implementação, não foi encontrada documentação sobre o seu uso em inglês e também não é possível averiguar se as simulações são ou não limitadas a problemas de logística. Sem documentação, o nível de conhecimento técnico necessário para modular e executar

[2]<https://github.com/dapel/AbstractSwarm>

as simulações é alto e a facilidade de uso é baixa, sendo necessário um conhecimento significativo de gráficos e de algoritmos.

Santos, Nunes and Bazzan (2017c) propõem a linguagem específica de domínio ABStractLang, introduzida no capítulo anterior, cujo objetivo é facilitar a modelagem de um ambiente simulado. A linguagem foi concebida a partir de abstrações que existem em comum entre diversas simulações e possui um conjunto completo de componentes para modelar o ambiente e a simulação. Estas abstrações foram explicadas em detalhes na seção anterior. A proposta promove a modularidade a partir do uso de modelos que representam aspectos da simulação. Exemplos de aspectos são a topologia do ambiente ou técnicas de aprendizado, sendo que os modelos foram concebidos com o objetivo de ser reusáveis. Na linguagem é usado o conceito de *concerns* para facilitar a modularidade em ABMS. Os *concerns* reduzem a complexidade da simulação que está sendo desenvolvida e permitem uma separação entre agente e ambiente, como ocorre na proposta de Al-Zinati et al. (2013).

A avaliação da ABStractLang foi feita a partir de um experimento com usuários que averiguou o nível de dificuldade para compreender simulações baseadas em agentes em comparação com a modelagem feita na plataforma NetLogo. A partir do experimento, os autores concluíram que o uso da linguagem trouxe vantagens em relação ao NetLogo quanto ao entendimento de ABMS. Um dos pontos que trouxe melhorias no entendimento foi a facilidade de uso dos modelos. Foi feita outra avaliação após a implementação da ferramenta ABStractme (SANTOS, 2019), em que os participantes implementaram simulações na plataforma NetLogo e na ferramenta. Esta avaliação evidenciou os benefícios tanto da linguagem quanto da abordagem MDD como um todo.

## 3.2 Modularidade e Reuso em Sistemas Multiagente

A seguir estão relacionados os trabalhos inseridos no contexto de sistemas multiagente que não lidam com simulações. Todas estas propostas utilizam a arquitetura *Belief-Desire-Intention* (BDI) (RAO; GEORGEFF, 1995), cujo foco está na implementação de três conceitos que definem a interação do agente com o ambiente: (i) crenças (*beliefs*), que representam o que o agente acredita conhecer sobre o ambiente; (ii) desejos (*desires*), que representam a motivação do agente ou os seus objetivos; e (iii) intenções (*intentions*), que representam os objetivos que o agente escolheu executar.

Warwas and Klusch (2011) propuseram o reuso de agentes BDI em uma camada

independente de plataforma em que é usada uma linguagem específica de domínio para apresentar uma representação dos diversos artefatos. A linguagem, chamada DSML4MAS, cobre os aspectos centrais de sistemas multiagente, tais como agentes, protocolos de interação, objetivos e comportamentos. Como exemplo, é feita uma especificação do mapeamento entre conceitos da plataforma Jadex[3] e a DSML4MAS. A proposta usa engenharia reversa dirigida a modelos para associar a linguagem à plataforma Jadex.

Na DSML4MAS é necessário que seja implementada uma transformação automática da linguagem para a plataforma de interesse do projetista, que no exemplo apresentado é a plataforma Jadex. Depois desta transformação é necessário refinar os modelos manualmente pois alguma informação pode estar ausente, logo é necessário um alto nível de conhecimento técnico. O modelo dos agentes pode ser reusado em outras plataformas multiagente com as devidas transformações. Os autores não exploram as possibilidades de modelagem de diversos sistemas e o reuso dos agentes entre estes sistemas na linguagem DSML4MAS. O foco do trabalho é em demonstrar como integrar a DSML4MAS a diferentes plataformas multiagente, permitindo o reuso dos elementos da linguagem nestas plataformas.

Para separar a modelagem do agente das outras partes do sistema de forma similar às propostas apresentadas na seção anterior, Busetta et al. (2000) introduzem o conceito de *capabilities* de agentes, que são componentes dos mesmos que permitem a separação dos agentes em módulos. As *capabilities* representam partes funcionais que se repetem em diferentes agentes. Separar o agente em módulos facilita o desenvolvimento modular, o reuso de componentes e diminui o retrabalho na modelagem de simulações. Uma *capability* pode ser usada da mesma forma que uma classe é usada em uma linguagem orientada a objetos, pois a mesma pode ter múltiplas instâncias no mesmo agente. A partir do exemplo, foi feita uma implementação em JACK[4], um kit para desenvolvimento comercial de agentes, em que as *capabilities* são transformadas em classes Java para ser instanciadas pelos agentes. Estas *capabilities* podem ser reusadas em agentes que fazem parte de outros sistemas de domínios específicos, já a facilidade de uso depende de como o desenvolvedor irá implementar as *capabilities* no seu sistema.

A partir do conceito de *capability* (BUSETTA et al., 2000), Nunes (2014) propôs a possibilidade de relacionamentos entre as *capabilities* de um agente. A proposta foca na produção de componentes de software modulares, com baixo acoplamento e alta coesão, no contexto de agentes BDI, para que, da mesma forma que nas propostas descritas

[3] <https://www.activecomponents.org/index.html#/project/news>
[4] <http://www.agent-software.com.au/products/jack/>

anteriormente, possa ser viável desenvolver sistemas multiagente de alta complexidade. Estes relacionamentos são análogos aos relacionamentos usados no desenvolvimento de software orientado a objetos, sendo do tipo associação, composição e generalização.

Com relação à software orientado a objetos, as similaridades entre *capabilities* e objetos estão nos atributos e nos métodos, que no caso de *capabilities* são semelhantes, respectivamente, às crenças e aos planos de ação. A *capability* também possui objetivos, que representam os serviços que a *capability* oferece, e o conjunto de objetivos é a interface da *capability*. Uma *capability* pode fornecer serviços a outras, neste caso ocorre uma relação de associação entre elas para que possam colaborar entre si e alcançar seus objetivos. O relacionamento de composição ocorre quando duas *capabilities* precisam compartilhar conhecimento, neste caso uma *capability* precisa ter acesso às crenças de outra *capability* para executar seus planos, ou seja, elas não são independentes uma da outra. O relacionamento de herança possui a função de facilitar o reuso, pois generaliza o comportamento de uma *capability*. As *capabilities* podem ser reusadas em agentes de outros domínios específicos.

Por fim, Ortiz-Hernández et al. (2016) propuseram um modelo de agentes BDI com foco na modularidade, em que as funcionalidades dos agentes se tornam unidades de código. De acordo com a proposta, para alcançar a modularidade é necessário: (i) um mecanismo para evitar colisão de nomes; (ii) satisfazer o princípio de ocultação de informação, que envolve esconder detalhes de implementação de um elemento de outros elementos que estão fora do escopo deste elemento; e (iii) no mesmo caminho das propostas descritas anteriormente, prover interfaces modulares. Para satisfazer estes requisitos, a proposta define dois elementos. O primeiro elemento, denominado *módulo*, é um conjunto de crenças, objetivos e planos que satisfaz o terceiro requisito. O segundo elemento, denominado *namespace*, é um nome usado para identificar um agrupamento de módulos, que satisfaz o primeiro e o segundo requisito. A colisão de nomes é evitada pois todos os identificadores dos elementos que fazem parte do conjunto do módulo são prefixados com um *namespace* para torná-los exclusivos ao escopo do *namespace*. Como o *namespace* pode ser global ou local, é possível satisfazer o princípio de ocultação de informação usando *namespaces* locais, que não são acessíveis a outros módulos.

Uma das vantagens no uso de *namespaces* é que é independente de qualquer linguagem e pode ser implementado em qualquer sistema. Como exemplo, foi feita uma implementação em Jason para ilustrar o funcionamento da proposta em uma linguagem específica. No exemplo é feito o desenvolvimento de um sistema multiagente baseado em

Contract Net Protocol (CNP) (SMITH, 1980), que possui os módulos iniciador e participante. Os agentes iniciam uma instância de CNP toda vez que forem executar uma tarefa. Toda instância de CNP usa um *namespace* diferente para isolar as crenças e eventos de cada negociação. Foi feita uma avaliação da proposta através de uma comparação entre duas versões do sistema CNP implementadas em Jason, com e sem o uso de *namespaces*. A quantidade de atualizações ao código é comparada depois da inserção das mesmas alterações ao código de ambas versões do sistema. A versão sem *namespaces* precisa de 197 atualizações em relação às 77 atualizações da versão com *namespaces*. A partir destes dados os autores concluem que a versão com *namespaces* é mais adequada para lidar com mudanças. A proposta foca no agrupamento de elementos do agente, cujo reuso em outros domínios específicos é facilitado graças à modularidade introduzida.

## 3.3 Comparação dos Trabalhos Relacionados

Com o objetivo de comparar os trabalhos relacionados apresentados anteriormente, uma série de critérios foram adotados, como explicado a seguir. São oito itens utilizados para a avaliação, conforme sintetizado na Tabela 3.1. Entre os critérios, está um resumo do que é proposto para se alcançar a modularidade e reuso. Também é avaliado o escopo dos artigos, a forma de avaliação da solução utilizada, a arquitetura proposta e qual é o elemento reusável de cada proposta. Além disso, é verificado se a proposta inclui uma interface gráfica para facilitar o uso, que pode tornar a solução mais acessível a quem não tem conhecimento técnico. Destacam-se ainda quais propostas utilizam uma linguagem específica de domínio para permitir a modelagem dos sistemas sem a necessidade de ter de lidar constantemente com uma linguagem de programação de propósito geral. Também são incluídos critérios que mostram quais propostas trouxeram facilidade de uso juntamente com a implementação da solução, para pessoas sem conhecimento técnico. Por último é possível avaliar como foi o suporte ferramental para cada proposta. Se a implementação é parcial, há a possibilidade de reusar elementos modelados em outros cenários mas pode ser que a solução não seja totalmente compatível com o cenário desejado, neste caso é necessário introduzir mudanças na implementação.

Tabela 3.1: Abordagens que promovem reuso em ABMS e MAS

| **Artigo** | **Modularidade e Reuso** | **Escopo** | **Avaliação** | **Arquitetura** | **Elemento Reusável** | **Interface Gráfica** | **DSL** | **Suporte Ferramental** |
|---|---|---|---|---|---|---|---|---|
| (SCERRI et al., 2010) | Uso de variáveis locais e compartilhadas | ABMS | Analisar tempo de execução | Arquitetura reativa | Módulos individuais | X | - | Implementação parcial |
| (AL-ZINATI et al., 2013) | Separação entre agente e ambiente | ABMS | Analisar escalabilidade | Arquitetura reativa | Comportamento do agente | X | - | Implementação parcial |
| (APELDOORN, 2015) | Agentes e estações | ABMS | Analisar tempo de espera | Arquitetura reativa | Comportamento do agente | X | X | Implementação parcial |
| (SANTOS; NUNES; BAZZAN, 2017c) | Desenvolvimento dirigido por modelos | ABMS | Analisar facilidade de entendimento | Arquitetura reativa | Aspectos da simulação | - | X | Implementação parcial |
| (WARWAS; KLUSCH, 2011) | Independência de plataforma | MAS | - | BDI | Componentes do sistema | X | X | Implementação parcial |
| (BUSETTA et al., 2000) | *Capabilities* | MAS | - | BDI | *Capabilities* | - | - | - |
| (NUNES, 2014) | Relacionamentos entre *capabilities* | MAS | - | BDI | *Capabilities* | - | - | - |
| (ORTIZ-HERNÁNDEZ et al., 2016) | *Namespaces* | MAS | Comparação entre versões | BDI | Trechos de código | - | - | - |

A partir da comparação entre as propostas estudadas pode-se verificar que a modularidade está bem difundida na área de sistemas multiagente e o reuso é viável em todos os casos graças à modularidade apresentada, mas quando há implementação, nem sempre a possibilidade de reuso está prevista ou há pouca informação sobre esta característica da implementação. Em todos os casos não é previsto o reuso de um agrupamento de componentes como está sendo proposto neste trabalho, somente de cada elemento separadamente.

Com base na avaliação do escopo dos artigos é possível observar que a metade não lida com simulações. Isto ocorre pois há poucos estudos em ABMS com foco na modularidade e no reuso uma vez que a própria ABMS é uma área de estudo pouco explorada. Ao observar como foi a avaliação da solução, fica explícito que boa parte das propostas tem entre seus objetivos facilitar a escalabilidade de sistemas multiagente. A maioria das avaliações foca em comparações entre versões destes sistemas nas quais o importante é o tempo de execução ser menor. A exceção está nos casos de Ortiz-Hernández et al. (2016), em que o objetivo da avaliação é averiguar a facilidade na manutenção e otimização do software ao usar *Namespaces*, e de Santos, Nunes and Bazzan (2017c), cujo objetivo foi facilitar o entendimento de simulações baseadas em agentes.

Metade das soluções tem por base a arquitetura BDI, que tem sido largamente utilizada. As propostas de Scerri et al. (2010), Al-Zinati et al. (2013), Apeldoorn (2015) e Santos, Nunes and Bazzan (2017c) seguem uma arquitetura reativa padrão para o comportamento dos agentes, em que os mesmos percebem o ambiente, decidem e agem a partir desta percepção. Em relação ao elemento reusável no caso de cada proposta, todos elementos envolvem todo ou parte do comportamento do agente, sendo que os trabalhos de Warwas and Klusch (2011), Santos, Nunes and Bazzan (2017c) e Ortiz-Hernández et al. (2016) são mais genéricos quanto ao reuso mas incluem a possibilidade de reusar o comportamento dos agentes.

As soluções de Apeldoorn (2015) e Al-Zinati et al. (2013) permitem o reuso em cenários diferentes se os mesmos estiverem previstos nas classes ou na linguagem implementadas, logo são as que trazem as maiores vantagens neste ponto. Contudo, a dificuldade em encontrar documentação para a proposta de Apeldoorn (2015) torna difícil averiguar se o reuso está limitado a problemas logísticos, que envolvem a otimização das atividades de uma organização. Desta forma, pode-se concluir que uma ferramenta ideal para reuso em sistemas multiagente deve incluir as características da solução de Al-Zinati et al. (2013): modularidade a partir da separação entre agente e ambiente, facilidade de

uso e possibilidade de reuso em cenários diferentes. Quando a implementação da solução é inexistente, o objetivo da proposta é demonstrar a ideia, a possibilidade de reuso está subentendida a partir da modularidade apresentada. Apesar de estar viabilizado, o reuso precisa ser uma feature da implementação para que o projetista possa utilizá-lo durante a modelagem dos sistemas.

A solução de Santos, Nunes and Bazzan (2017c), que é o foco deste trabalho, utiliza a noção de *concerns*, que são similares às *capabilities* (BUSETTA et al., 2000) de agentes no ponto em que permitem a modularização de elementos usados na modelagem da simulação, mas ao contrário de *capabilities*, não permitem o reuso. Como o *concern* separa a simulação em módulos, se for possível reusar o mesmo em outra simulação que tenha módulos similares, haverá uma simplificação na modelagem da simulação e um reuso do agrupamento de elementos da simulação que não está previsto nas propostas estudadas.

## 3.4 Considerações Finais

Uma constante nas propostas comparadas é a necessidade de modularizar os elementos da simulação, para facilitar a modelagem de sistemas multiagente. O reuso não é o foco de todos os trabalhos, o mesmo ocorre a partir da alta coesão destes elementos, que são reusados individualmente e podem ou não estar acoplados uns aos outros. O conceito de *concerns* foi introduzido com o objetivo de permitir a modularização de um agrupamento de componentes e pode facilitar o reuso de conjuntos de elementos, mas na definição do mesmo há a possibilidade de ocorrer um alto acoplamento. O foco deste trabalho é expandir o encapsulamento de *concerns* para permitir um maior reuso de software na linguagem proposta por Santos, Nunes and Bazzan (2017c).

# 4 *CONCERN*: UMA ABSTRAÇÃO PARA REUSO EM ABMS

Neste capítulo, é apresentada a proposta deste trabalho, que consiste em um refinamento da abstração dos elementos da simulação, com o objetivo de dar suporte ao desenvolvimento de simulações baseadas em agentes através do reuso de software. Este refinamento da definição dos *concerns* introduz uma interface para o *concern* que o torna uma unidade conceitual reusável, separando os seus componentes dos elementos que pertencem a outras partes da simulação. A partir desta interface é feito o compartilhamento de dados entre *concerns*. Também é feita a definição da injeção de dependência entre *concerns*.

## 4.1 Definição

O *concern*, como estabelecido na abordagem MDD4ABMS apresentada no Capítulo 2, é uma abstração de partes da simulação que visa agrupar elementos relacionados, conforme o interesse do projetista, para auxiliar na modularidade e escalabilidade em ABMS. Mais especificamente, a lista dos elementos agrupáveis em um *concern* pode ser vista na Tabela 4.1. A principal função desta abstração é permitir uma alta coesão do sistema, possibilitando o agrupamento de elementos que possuem objetivos em comum de forma a proporcionar uma separação conceitual destes elementos em relação a outros que possuem funções diversas.

Cada *concern*, apesar de representar uma visão parcial do modelo, está altamente acoplado aos demais elementos de outros *concerns* do sistema. Assim, os elementos agrupados no *concern* podem fornecer ou depender de dados relacionados a outros *concerns*. No caso de uma simulação em que sinais de trânsito fazem o controle do tráfego de veículos, modelada com base no trabalho de Gershenson (2005), o agente que representa o

Tabela 4.1: Partes do *concern*

| **Parte** | **Detalhes** |
|---|---|
| Título | Nome do *Concern*, precisa ser único. É necessário para identificar quais são os elementos contidos em qual *concern* |
| Descrição | Descrição do *concern* |
| Entidades | Entidades e agentes com seus atributos |
| Parâmetros | Representam os atributos do *concern* |

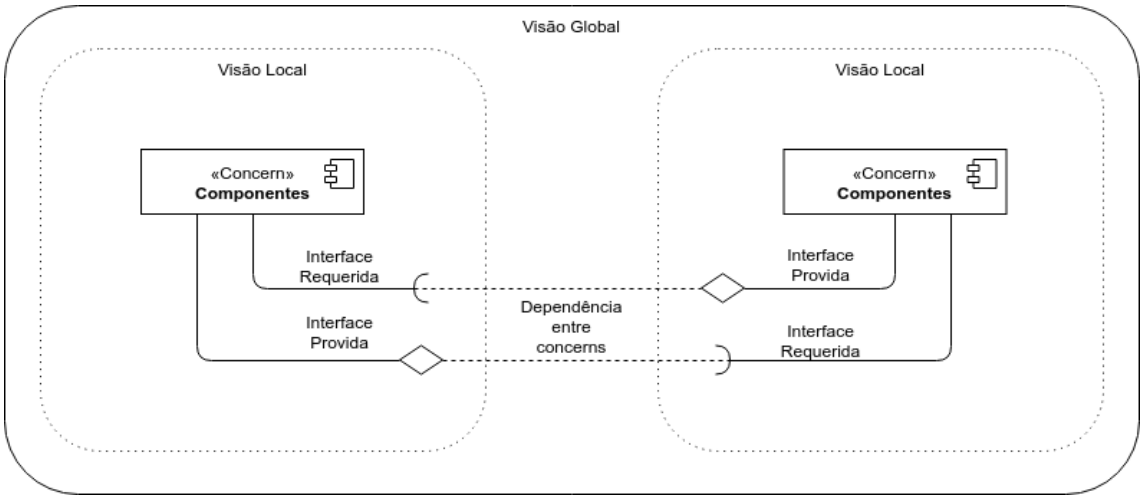
controle de um sinal de trânsito está contido no *concern* que contém os sinais de trânsito. Para que este agente esteja localizado na rede de tráfego, o mesmo precisa saber o ponto de cruzamento da rede de vias onde deve estar situado. Essa informação encontra-se no *concern* que representa a rede de autoestradas, mais especificamente a entidade contida no *concern* que define a localização dos cruzamentos. Há uma relação de dependência entre o agente que representa o sinal de trânsito e a entidade que contém a localização dos cruzamentos. A separação do *concern* em relação ao resto da simulação é somente uma forma de aumentar a coesão e permitir a escalabilidade do sistema, não há qualquer interface ou conjunto de regras que limite o relacionamento entre *concerns*.

O alto acoplamento é um obstáculo para o reuso destes agrupamentos de elementos pois não há uma separação conceitual destas partes do resto do sistema para permitir o seu reuso em outras simulações. Logo, é necessário modificar a definição do *concern* de forma a evidenciar uma separação da especificação dos seus parâmetros, entidades e agentes do resto da simulação.

Assim, na proposta deste trabalho, o *concern* é definido de forma a separá-lo em duas partes que representam duas visões diferentes do mesmo: (i) definição dos componentes do *concern*, que inclui todos os elementos da sintaxe da linguagem ABStractLang especificados na Tabela 4.1. Nesta definição, estes componentes estão identificados como contidos somente no *concern* e não podem se relacionar diretamente com elementos de outros *concerns*. Dessa forma, o *concern* se torna uma unidade conceitual capaz de encapsular os seus elementos, separando-os do resto da simulação e ocultando informações que não são necessárias para o funcionamento de partes de outros *concerns*; e (ii) definição da interface do *concern*, onde estão evidenciados apenas os elementos contidos no *concern* que dependem de ou fornecem algum dado para o ambiente externo ao *concern*. Nesta interface, são referenciados os componentes do *concern* de forma a torná-los acessíveis ao resto do sistema.

De forma complementar à definição do *concern*, há uma visão local e global da simulação. Na visão local se encontram os componentes e a interface do *concern*. A visão global é onde estão localizados os relacionamentos definidos entre *concerns*. Como exemplo, no caso da simulação do controle de sinais de trânsito sobre o tráfego de veículos, a interface do *concern* que representa os sinais de trânsito inclui o atributo, contido no agente de controle de um sinal de trânsito, que define a criação do sinal de trânsito no cruzamento de estradas. Este atributo está aberto para receber uma dado que define essa localização no mapa. O *concern* que representa a rede de autoestradas, por sua vez, possui

Figura 4.1: Refinamento da definição do *concern*




o atributo que define a localização dos cruzamentos na rede de vias, que está na sua interface e oferece o dado contido neste atributo para o meio externo ao *concern*. Em ambos os casos, há uma abertura no *concern* em que o mesmo está preparado para receber ou enviar algum dado, mas não está definida qualquer ligação entre estes *concerns*. O relacionamento efetivo entre *concerns* ocorre pela dependência necessária ao funcionamento, neste caso pela localização. Assim, no exemplo, o agente que representa o controle do sinal de trânsito recebe a localização de cruzamento de vias do atributo que contém tal informação, mas que está no *concern* que representa a rede de autoestradas. Esta dependência especifica qual é o *concern* que está provendo o dado e qual é o *concern* que está recebendo este dado. Uma visão geral do refinamento da definição do *concern* juntamente com a definição da dependência entre *concerns* podem ser vistas na Figura 4.1.

## 4.2 Interface do *Concern*

Este estudo propõe um refinamento da definição do *concern*, que passa a ser composto por seu conjunto de componentes e por sua interface. A definição da interface do *concern*, então, é refinada para especificar se a mesma é uma interface requerida ou provida, para elucidar os pontos em que os elementos contidos no *concern* dependem de ou fornecem alguma informação. A interface requerida representa funcionalidades de partes do *concern* que precisam ser completadas com dados que estão fora do *concern*. Estas partes do *concern* só poderão exercer sua função através de uma injeção de depen-

dência. A interface provida, por sua vez, reflete os elementos contidos no *concern* que podem fornecer dados para fora do *concern*. A interface possui um ou mais conectores onde pode ser injetada a dependência. Nestes pontos são definidas as referências para os componentes do *concern* que precisam ser compartilhados na visão global do sistema.

Assim, foi definida uma linguagem que descreve esta interface e permite que sejam feitas conexões entre *concerns* para facilitar o reuso destas unidades conceituais. Nesta linguagem, é estabelecida a interface requerida e provida do *concern*, de modo que o projetista pode controlar o que pode ser compartilhado na visão global do sistema. Os elementos do *concern* que não estão especificados dentro desta definição não podem ser referenciados por elementos que estão fora do *concern*, pois estes são componentes internos que não fazem parte da interface. Para facilitar a sua identificação ao ocorrer o seu reuso em contextos diferentes, as abstrações que podem ser compartilhadas entre *concerns* estão descritas claramente como parte do *concern* e não somente como parte de todo o sistema. A sintaxe da linguagem está no formato de Definição de Tipo de Documento, ou *Document Type Definition* (DTD), e pode ser vista na Figura 4.2. Os detalhes da linguagem podem ser vistos na Tabela 4.2.

Figura 4.2: Definição da linguagem para os conectores

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE concern [
    <!ELEMENT concern (id, description, connectors*, elementTypes*)>
        <!ELEMENT id (#PCDATA)>
        <!ELEMENT description (#PCDATA)>
        <!ELEMENT connectors (connector+)>
            <!ELEMENT connector (internalElementType, internalElementId)>
            <!ATTLIST connector id CDATA #REQUIRED direction (required|provided) #REQUIRED type (data|function) #REQUIRED>
                <!ELEMENT internalElementType EMPTY>
                <!ATTLIST internalElementType type (attribute|entity|agent) #REQUIRED>
                <!ELEMENT internalElementId EMPTY>
                <!ATTLIST internalElementId id CDATA #REQUIRED>
        <!ELEMENT elementTypes (attribute*, entity*, agent*)>
            <!ELEMENT attribute (initSource, updateSource, capability, entity)>
            <!ATTLIST attribute name CDATA #REQUIRED type CDATA #REQUIRED cardinality CDATA #REQUIRED>
                <!ELEMENT initSource EMPTY>
                <!ATTLIST initSource type CDATA #REQUIRED>
                <!ELEMENT updateSource EMPTY>
                <!ATTLIST updateSource type CDATA #REQUIRED>
                <!ELEMENT capability EMPTY>
                <!ATTLIST capability id CDATA #REQUIRED>
                <!ELEMENT entity EMPTY>
                <!ATTLIST entity type CDATA #REQUIRED name CDATA #REQUIRED>
            <!ELEMENT entity (attributes?)>
            <!ATTLIST entity name CDATA #REQUIRED>
                <!ELEMENT attributes (attribute+)>
            <!ELEMENT agent (attributes?)>
            <!ATTLIST agent name CDATA #REQUIRED>
                <!ELEMENT attributes (attribute+)>
]>
```

Tabela 4.2: Definição da interface requerida e provida

| Parte | Detalhes |
|---|---|
| *Id* | Identificação, corresponde ao título do *concern* definido durante a sua criação |
| *Description* | Descrição do *concern* |
| *Connectors* | Conectores da interface requerida e provida. Cada conector, identificado como *connector*, possui uma identificação para diferenciá-lo do resto, definida como atributo *id*. A interface pode ser de dois tipos, definidos no atributo *direction*: (i) *required*, em que o *concern* requer dados externos para completar alguma funcionalidade de um elemento interno e (ii) *provided*, em que o *concern* oferece algum dado de um elemento interno para ser acessado por outros *concerns*. O conector da interface pode ser dos tipos *data* ou *function* |
| *InternalElementType* | O tipo do elemento interno precisa ser fornecido na definição do conector da interface, para facilitar na sua identificação. Pode ser dos tipos *attribute*, *entity* ou *agent* |
| *InternalElementId* | Todo elemento interno que provê ou requer um dado precisa ter o seu identificador explicitado na definição do conector da interface |
| *ElementTypes* | Detalhes complementares dos elementos internos identificados nos conectores da interface que precisam ficar visíveis para outros *concerns*. Inclui a lista de elementos internos referenciados nos conectores da interface com o seu identificador, que podem ser do tipo *attribute*, *entity* ou *agent*. Todo atributo possui *name*, *type*, *cardinality*, *initSource*, *updateSource*, *capability* e *entity*, que devem ser preenchidos conforme a definição do mesmo na modelagem do *concern*. No caso em que houver referências a um ou mais atributos contidos em uma entidade ou agente, é necessário fornecer o nome desta entidade ou agente |

## 4.3 Injeção de Dependência

A interface apresentada somente deixa claro o que pode ser conectado entre os *concerns*, a injeção de dependência em si representa a forma como o vínculo entre *concerns* é criado e descreve a dependência de uma interface requerida em relação a uma interface provida. Esta conexão está no nível global do sistema e possui sua própria linguagem em que é descrita a injeção do componente provido no elemento requerido. A sintaxe desta linguagem também está no formato DTD e pode ser vista na Figura 4.3. Os detalhes da linguagem podem ser vistos na Tabela 4.3. O mesmo *concern* pode estar conectado a outros através de múltiplas conexões mas cada *concern* possui o seu conjunto de

Figura 4.3: Definição da linguagem para a injeção de dependência

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE connection [
    <!ELEMENT connection (title, type, requiringConcern, providingConcern)>
        <!ELEMENT title (#PCDATA)>
        <!ELEMENT type (#PCDATA)>
        <!ELEMENT requiringConcern (connector)>
        <!ATTLIST requiringConcern title CDATA #REQUIRED>
            <!ELEMENT connector EMPTY>
            <!ATTLIST connector id CDATA #REQUIRED internalElementId CDATA #REQUIRED>
        <!ELEMENT providingConcern (connector)>
        <!ATTLIST providingConcern title CDATA #REQUIRED>
            <!ELEMENT connector EMPTY>
            <!ATTLIST connector id CDATA #REQUIRED>
]>
```

Tabela 4.3: Definição da injeção de dependência entre *concerns*

| **Parte** | **Detalhes** |
|---|---|
| *Title* | Título para identificar a injeção de dependência entre *concerns* |
| *Type* | Tipo de dependência. Pode ser de dados ou funções |
| *RequringConcern* | Identificação do *concern* que requer os dados, cujo nome está no atributo *title*, juntamente com a identificação do conector da interface requerida e do elemento interno do *concern* referido no conector da interface, ambos definidos como os atributos *id* e *internalElementId* do *connector* |
| *ProvidingConcern* | Identificação do *concern* que está provendo o dado, cujo nome está no atributo *title*, para o elemento do *concern* que requer estes dados juntamente com a identificação do conector da interface provida, definida como o atributo *id* do *connector* |

interfaces requeridas e providas. A partir destas definições, é possível reusar os *concerns*, juntamente com todas as entidades e parâmetros que estão encapsulados nos mesmos. O reuso se aplica para os *concerns* e para suas interfaces, as dependências entre *concerns* são específicas para cada simulação.

## 4.4 Considerações Finais

A partir do refinamento da definição do *concern*, que inclui a sua separação em componentes e interface, é possível definir um encapsulamento para esta abstração de elementos. No detalhamento da interface, são apresentadas as interfaces do tipo requerida e

provida, que esclarecem quais são os elementos do *concern* que dependem de ou fornecem dados externos ao *concern*. O *concern* se torna um elemento reusável a partir deste refinamento de sua definição, juntamente com a descrição das injeções de dependência que definem o fluxo de dados entre *concerns* na visão global da simulação.

# 5 EXTENSÃO DA ABSTRACTME PARA REUSO DE *CONCERNS*

Neste capítulo, são descritas as extensões implementadas na ferramenta de modelagem ABStractme para suportar o refinamento da definição do *concern* descrito anteriormente. A extensão introduz conectores para interfaces requerida e provida, injeção de dependência entre *concerns* e importação de *concerns* reusáveis.

## 5.1 *Concern* e suas Interfaces

Conforme o refinamento da definição do *concern* como uma unidade conceitual reusável, a ferramenta ABStractme foi estendida para permitir a especificação dos conectores da interface do *concern*. Esta especificação foi feita a partir da definição de um arquivo no formato XML, em que a linguagem definida anteriormente foi usada para descrever estes conectores. Cada *concern* possui o seu descritor de conectores que especifica onde pode ser feita uma injeção de dependência entre *concerns*. Um exemplo de arquivo descritor de conectores pode ser visto na Figura 5.1.

Na figura é possível visualizar: (i) título e descrição do *concern*; (ii) conjunto de conectores do *concern*, cada um com seu identificador, direção para definir se a interface é do tipo requerida ou provida e tipo de dado compartilhado; (iii) tipo e identificador do elemento interno do *concern* que está sendo compartilhado; e (iv) dados detalhados do

Figura 5.1: Descritor dos conectores da interface do *concern*

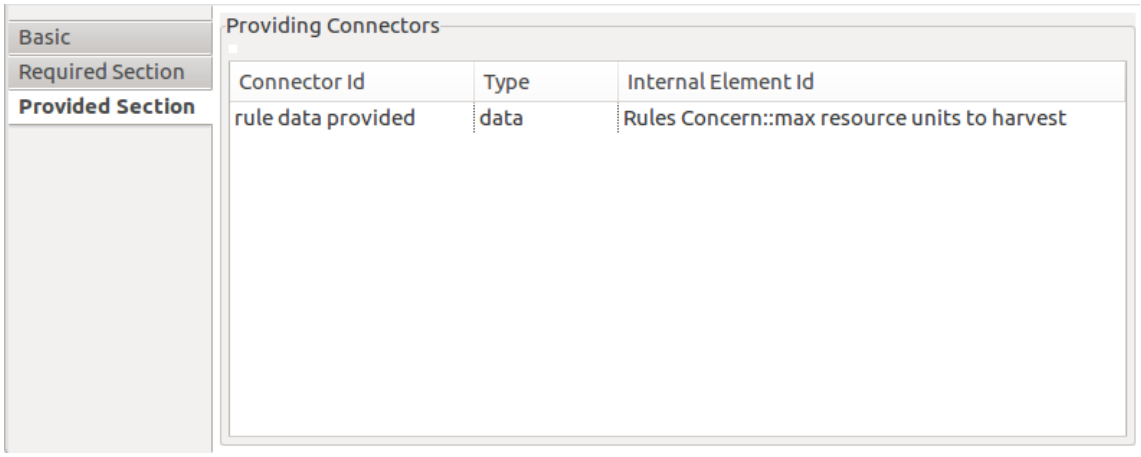
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<concern>
    <id>Rules Concern</id>
    <description>Concern that holds the rules for harvesting the resource</description>

    <connectors>

      <connector id="rule data provided" direction="provided" type="data">
        <internalElementType type="attribute" />
        <internalElementId id="Rules Concern::max resource units to harvest" />
      </connector>

    </connectors>
    <elementTypes>
      <attribute name="max resource units to harvest" type="INTEGER" cardinality="1">
        <initSource type="static value" value="5"/>
        <updateSource type="empty"/>
        <capability id="empty"/>
        <entity type="empty"/>
      </attribute>
    </elementTypes>
</concern>
```

Figura 5.2: Conectores da interface provida do *concern*



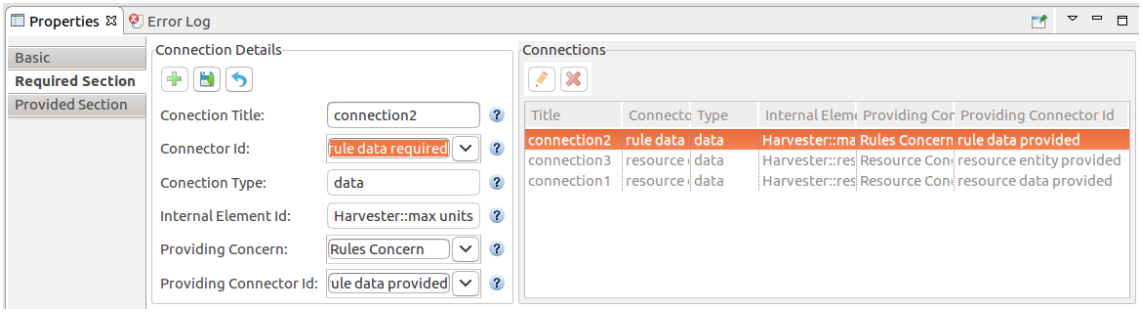

elemento interno do *concern* que requer ou fornece os dados, conforme a necessidade.

A ferramenta ABStractme foi estendida para permitir a observação dos conectores da interface provida descritos no arquivo, como no exemplo que aparece na Figura 5.2. Neste exemplo, pode-se ver as propriedades da IDE Eclipse onde é possível editar os dados do *concern*. Ao selecionar a aba para visualizar os conectores da interface provida, chamada *Provided Section*, é apresentada ao projetista uma tabela que descreve todos os conectores do *concern* em que ocorre o fornecimento de dados para o resto do sistema, conforme o arquivo descritor dos conectores da interface do *concern*. A função desta tabela é facilitar a visualização dos conectores, os mesmos são editados somente pelo arquivo descritor. Em cada coluna é possível observar: (i) a identificação dos conectores do *concern* que fornecem dados; (ii) tipo de dado fornecido; e (iii) identificador do elemento interno cujos dados estão sendo fornecidos.

## 5.2 Injeção de Dependência

Durante a modelagem das simulações, para que o projetista possa reusar *concerns* que estão separados dos resto da simulação, é necessário que o mesmo possa gerar conexões que representam a dependência e fluxo de dados entre estas unidades conceituais na visão global da simulação. A ferramenta ABStractme foi estendida para suportar a especificação de conexões entre *concerns*. Quando houver um *concern* que requer dados a partir dos conectores da interface requerida definidos no arquivo descritor, as conexões podem ser criadas na ferramenta ABStractme. Ao abrir a ferramenta, no diagrama com a

Figura 5.3: Campos para a geração de uma injeção de dependência entre *concerns*




visão global da simulação apresentado no Capítulo 2, ao selecionar o losango do *concern* e escolher a aba *Required Section* das propriedades padrão da IDE Eclipse mostrada na Figura 5.3, o projetista pode criar uma injeção de dependência.

Nesta figura é possível visualizar, da esquerda para a direita: (i) abas de propriedades dos *concern*, na figura aparece selecionada a aba de criação de conexões entre *concerns* a partir dos conectores da interface requerida; (ii) campos para criar ou editar e salvar uma conexão entre *concerns*. Cada injeção de dependência possui um identificador específico. Além disso, é preciso que o projetista selecione algum conector que requer algum dado a partir do que é especificado no arquivo descritor de conectores. Se houver um *concern* que faça parte do mesmo projeto e esteja fornecendo um dado do mesmo tipo que o dado requerido, o nome do *concern* e o identificador deste conector de interface provida poderão ser selecionados nos últimos dois campos de criação de conexão. No exemplo da figura, o *concern* e o conector provido são fornecidos pelo arquivo descritor de conectores definido na Figura 5.1. A aba de conectores providos do *concern* que está provendo o dado aparece na Figura 5.2; e (iii) os dados desta injeção de dependência que, ao ser salvos em um arquivo XML que descreve a mesma, ficam visíveis na tabela à direita.

Qualquer conexão criada pode ser editada ou excluída ao selecionar a linha da mesma na tabela, as modificações são salvas no arquivo descritor de conexões, cujo formato está conforme a sintaxe da linguagem que descreve a dependência entre *concerns* apresentada no Capítulo 4. O conteúdo do arquivo XML descritor da injeção de dependência que estava sendo editada na Figura 5.3 aparece na Figura 5.4, onde é possível observar: (i) título e tipo de conexão entre *concerns*; (ii) identificação do *concern* que requer os dados juntamente com a identificação do conector e do elemento interno do *concern* referido no conector; e (iii) identificação do *concern* que está provendo os dados

Figura 5.4: Descritor de uma injeção de dependência entre *concerns*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<connection>
    <title>connection2</title>
    <type>data</type>
    <requiringConcern title="Harvester Concern">
        <connector id="rule data required" internalElementId="Harvester::max units to harvest"/>
    </requiringConcern>
    <providingConcern title="Rules Concern">
        <connector id="rule data provided"/>
    </providingConcern>
</connection>
```

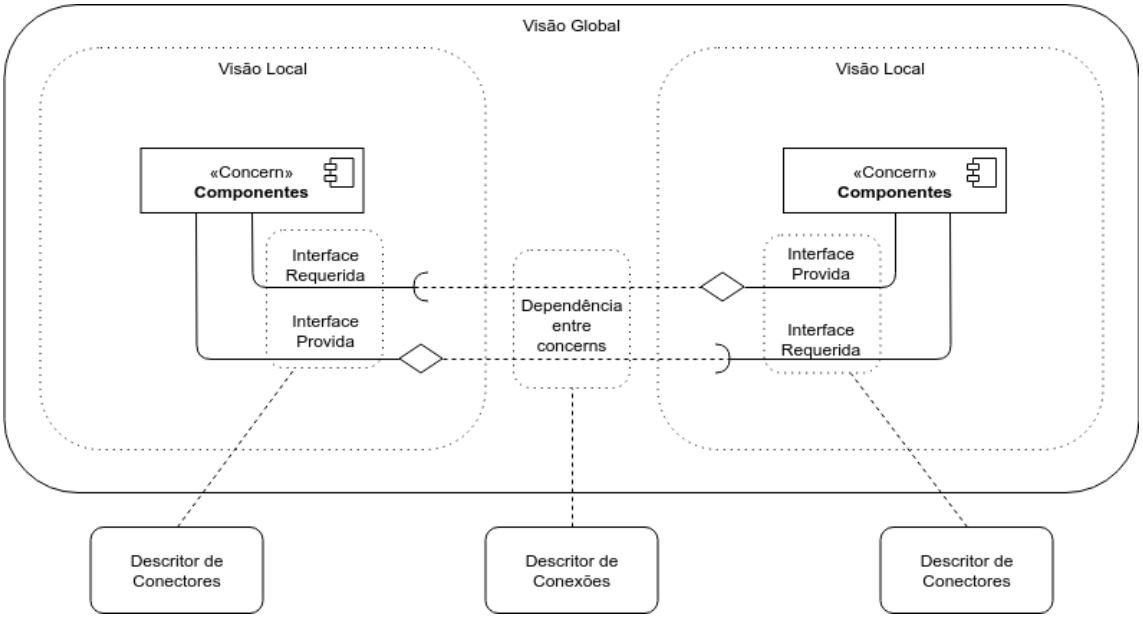
juntamente com a identificação do conector.

## 5.3 Importação de *Concerns* Reusáveis

Ao estender a ferramenta para permitir a criação dos conectores e da injeção de dependência entre *concerns*, não há mais obstáculos para o reuso dos *concerns*. A visão geral do refinamento da definição do *concern* atualizada com a definição dos arquivos descritores dos conectores e das conexões entre *concerns* pode ser vista na Figura 5.5. Para possibilitar o reuso de software, primeiramente foi removida da ferramenta a possibilidade do projetista gerar alguma dependência entre elementos de *concerns* diferentes pelo diagrama de edição do *concern*, para impedir que seja feita uma conexão entre componentes contidos em *concerns* diferentes que não estão referenciados nas interfaces dos mesmos. Após, foi estendida a ferramenta para suportar a importação de *concerns* de outras simulações. Neste caso, *concerns* de outras simulações podem ser reusados juntamente com os seus arquivos descritores dos conectores da interface do *concern*.

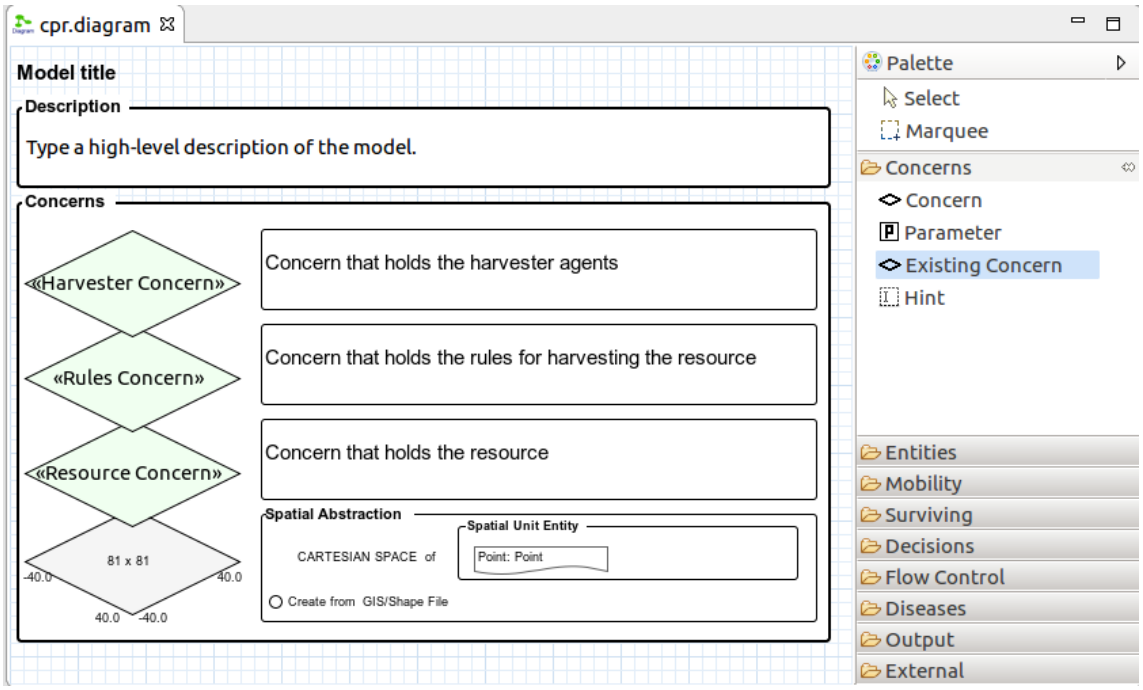
Para a extensão da ferramenta, foi adicionado o item *Existing Concern* à paleta da ferramenta, que pode ser utilizado para acrescentar um *concern* à pilha de *concerns*. Ao adicionar este tipo de *concern*, o projetista precisa selecionar um projeto em que já exista um *concern* para importar o *concern* juntamente com o seu arquivo descritor de conectores. Este *concern* é criado com o mesmo título e descrição do *concern* importado e funciona de forma similar a um *concern* criado pelo projetista, com a diferença de que o seu losango aparece com outra cor na visão geral da simulação e seu identificador aparece entre aspas angulares, como pode ser observado na Figura 5.6. Além disso, este *concern* importado pode ter unicamente o seu título e descrição alterados pelo projetista, e não o seu diagrama, que pode ser visto e editado somente no projeto de origem do mesmo. A partir do momento em que o *concern* reusável é adicionado à simulação, o projetista tem

Figura 5.5: Definição dos arquivos descritores dos conectores e das conexões entre *concerns*




a liberdade para gerar injeções de dependência entre o mesmo e *concerns* que já existam na simulação.

Figura 5.6: Visão geral da ferramenta com a importação de *concerns* externos



## 5.4 Considerações Finais

A partir do refinamento da definição do *concern*, é possível expandir a ferramenta ABStractme para permitir o reuso dos mesmos. Os passos de implementação incluem: (i) a criação de arquivos descritores dos conectores da interface; (ii) a visualização dos conectores da interface provida nas propriedades da IDE Eclipse; (iii) a geração das injeções de dependência a partir dos conectores da interface requerida e a sua visualização nas propriedades da IDE Eclipse; e (iv) a opção para importar *concerns* reusáveis na ferramenta. Esta extensão da ferramenta pode ser usada para ilustrar o reuso de *concerns* a partir de duas simulações diferentes que contém *concerns* similares.

# 6 ESTUDO DE CASO

Neste capítulo, é apresentado um estudo de caso para demonstrar as vantagens do reuso de software através da utilização de abstrações reusáveis como *concerns*. O estudo de caso envolve o compartilhamento de dados a partir de agentes que extraem um recurso em comum. São identificados exemplos de conectores de interfaces provida e requerida dos *concerns* que existem em comum nos trabalhos apresentados. Os *concerns* são então modelados através da ferramenta ABStractme. Para demonstrar o reuso de *concerns* em simulações diferentes que possuem pontos em comum, dois casos em que há o compartilhamento de recursos são modelados.

## 6.1 Simulações Alvo

Foram estudadas duas simulações para exemplificar o reuso e compartilhamento de *concerns* a partir do refinamento da definição dos mesmos apresentado anteriormente. Em ambas simulações, ocorre o compartilhamento de um recurso em comum, que possui um conjunto de características que o classifica como um *common-pool resource* (OSTROM, 1990). A principal característica deste *common-pool resource* é estar disponível para o consumo de diversos agentes, sendo que nenhum agente possui o direito exclusivo de posse sobre o recurso. Neste caso, há o risco de o recurso, que geralmente é finito e renovável, ser consumido sem controle até se extinguir. Para que a fonte de sustento de todos os indivíduos da comunidade não deixe de existir, os agentes precisam se submeter a um conjunto de regras sobre o consumo deste recurso para garantir a sobrevivência do grupo.

A primeira simulação estudada é definida com base no trabalho de Schlüter, Tavoni and Levin (2016), em que é explorada a cooperação entre agentes que precisam compartilhar um recurso qualquer, sendo que os agentes regulam o consumo dos vizinhos. A segunda tem como base o trabalho de Castilla-Rho et al. (2015), em que é definida a simulação da interação entre agentes que representam fazendeiros que precisam compartilhar recursos hídricos para a irrigação de suas plantações com agentes que representam controladores de poços que abastecem a população de uma cidade com água potável, sendo que há um agente que regula o consumo dos recursos. Uma apresentação dos trabalhos estudados pode ser vista na Tabela 6.1. Os trabalhos estão resumidos quanto à descrição dos agentes indicados em cada um juntamente com uma explicação breve da tomada de

Tabela 6.1: Resumo das simulações alvo

| | **(SCHLÜTER; TAVONI; LEVIN, 2016)** | **(CASTILLA-RHO et al., 2015)** |
|---|---|---|
| Agentes | Agentes como coletores de recurso que regulam uns aos outros | Agentes que representam fazendeiros individuais e controladores de poços da cidade, que tomam decisões ligadas à operação de poços para coleta de água. Agentes reguladores da coleta de água, que limitam as decisões dos outros agentes baseados em regras definidas |
| Tomada de Decisão | O agente coletor tem um conjunto de estratégias. Toda decisão é influenciada pela estratégia selecionada. Cada decisão traz uma vantagem e um custo | Agentes dependem das informações disponíveis no local para tomar suas decisões. Existem mecanismos de adaptação que o agente pode usar para modificar sua estratégia de coleta de recurso |
| Recurso | Um recurso qualquer compartilhado entre os agentes que pode ser explorado. O recurso é composto de unidades que representam o que pode ser extraído do recurso | Lençóis freáticos - o recurso representa um aquífero |
| Uso do Recurso | O recurso é coletado para satisfazer as necessidades do agente | O recurso é utilizado para irrigação de plantações e para abastecimento da população da cidade |
| Regulamento para uso do recurso | As regras sobre a coleta do recurso são impostas pela sociedade, desobedecer as regras significa sanções sociais: isolamento, perda de apoio dos vizinhos, etc | O regulador limita o comportamento dos fazendeiros e sabe os pontos do mapa em que o aquífero está sendo explorado além do que pode repor |

decisão dos mesmos em cada caso. Quanto ao recurso, é especificado o que é e como é utilizado. Também há um resumo da regulamentação definida para a coleta do recurso em cada caso.

## 6.2 *Concerns*: Design e Implementação

Para modelar as simulações alvo na ferramenta ABStractme de modo a fazer o reuso de *concerns*, é necessário determinar os *concerns* que podem ser utilizados em ambas as simulações alvo juntamente com as abstrações encapsuladas e os conectores

das interfaces de cada *concern*. Após a definição, estes elementos são modelados na ferramenta e descritos no arquivo descritor de conectores das interfaces de cada *concern*.

A partir do conceito de uma comunidade que precisa gerenciar o consumo de um *common-pool resource*, é possível definir um modelo comum a diversas simulações e um conjunto de variáveis mínimo que permita este gerenciamento (OSTROM, 1990). Neste modelo podem ser definidos três *concerns* para ilustrar as relações entre os componentes da simulação: (i) o *concern* que representa conjunto dos agentes coletores do recurso; (ii) o *concern* que representa o recurso que precisa ser compartilhado entre os agentes e é fundamental para o seu sustento; e (iii) o *concern* que representa as regras às quais os agentes devem se submeter para que haja um controle sobre o consumo do recurso. Este modelo é um ponto de partida para a definição dos *concerns* reusáveis. Com este objetivo, foi feita uma comparação entre os trabalhos que foca nos pontos em comum. A comparação pode ser vista na Tabela 6.2. Nesta tabela, os dados coletados na Tabela 6.1 são sintetizados para deixar em destaque o que há em comum entre as simulações alvo. Os trabalhos são comparados quanto à variáveis dos agentes indicados em cada um. Quanto ao recurso, são especificadas as variáveis definidas para o mesmo. Também há um resumo da regulamentação definida para a coleta do recurso em cada caso. Por fim, é definido um conjunto de *concerns* de cada caso e como os mesmos se relacionam.

Entre os pontos em comum das simulações alvo pode-se citar:

1. definição de agentes que precisam coletar um recurso para o seu sustento;
2. uso de estratégias de adaptação: cada agente possui uma lista de comportamentos possíveis em relação à coleta do recurso;
3. um recurso que seja finito e possa ter seu estado modificado por fatores externos, sendo que estas mudanças de estado são parcialmente percebidas pelos agentes;
4. parâmetros culturais do agente: crenças e valores influenciados por tradições e pelos vizinhos que são um fator na escolha da estratégia de adaptação;
5. definição de regras para a coleta do recurso: há um limite para a coleta do recurso, e o agente que passar deste limite pode ser punido economicamente e
6. definição clara dos custos e benefícios da coleta de cada unidade do recurso.

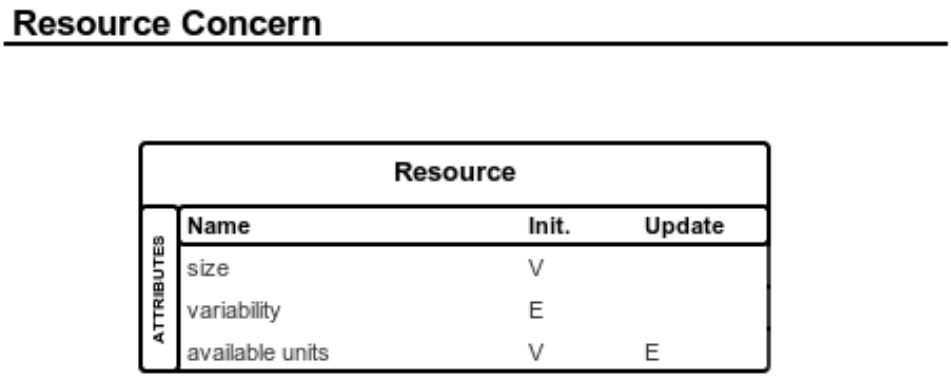
Usando a Tabela 6.2 como base, foram modelados três *concerns* reusáveis na ferramenta ABStractme para facilitar a modelagem das duas simulações apresentadas anteriormente. Os conectores definidos podem ser reusados juntamente com as abstrações

Tabela 6.2: Resumo dos pontos em comum nas simulações alvo

| | **(SCHLÜTER; TAVONI; LEVIN, 2016)** | **(CASTILLA-RHO et al., 2015)** |
|---|---|---|
| Variáveis dos Agentes | Benefícios e custos esperados por coletar uma unidade do recurso; normas internas que influenciam na escolha de estratégia de coleta; energia do agente e lista de estratégias de coleta de recurso | Variáveis dos fazendeiros: benefícios e custos esperados por bombear o recurso hídrico; lucro do agente; lista de estratégias de coleta de recurso; taxa de bombeamento; estresse hídrico acumulado e normas internas que influenciam na escolha de estratégia de coleta. Variáveis dos poços da cidade: localização espacial e taxa de bombeamento |
| Variáveis do Recurso | Tamanho/quantidade máxima de unidades, variabilidade no tamanho das unidades, unidades do recurso disponíveis para coleta | Rebaixamento, tamanho/quantidade máxima de unidades, variabilidade no tamanho das unidades, unidades do recurso disponíveis para coleta |
| Regulamento para uso do recurso | Conjunto de regras: número máximo de unidades do recurso que pode ser extraído por cada agente, multa por exceder o limite de unidades extraídas, benefícios e custos por seguir as regras, sanções a aplicar a quem desobedecer as regras e normas sociais compartilhadas entre vizinhos | Conjunto de regras: número de poços de bombeamento de água que podem ser construídos por ano, multa por exceder o limite de poços que podem ser construídos |
| *Concerns* | (i) *Harvester*: contém os agentes que representam os coletores, além de dados sobre o comportamento dos agentes; (ii) *Resource*: contém os dados do recurso; e (iii) *Rules*: contém a lista de regras para a coleta do recurso | (i) *Farmer*: contém agentes que representam os fazendeiros; (ii) *City*: contém dados da cidade e agentes que representam os controladores dos poços da cidade; (iii) *Aquifer*: contém dados do aquífero; (iv) *Rules*: contém a lista de regras para a construção de poços de água; e (v) *Rules Agent*: contém o agente regulador |
| Relações entre *Concerns* | *Harvester* precisa importar dados dos *concerns Resource* e *Rules* para a tomada de decisão de seus agentes | *Farmer* e *City* precisam importar dados dos *concerns Aquifer* e *Rules* para a tomada de decisão dos agentes. O agente contido no *concern Rules Agent* importa dados sobre o estado atual do aquífero contidos no *concern Aquifer* |

encapsuladas em cada *concern*. Estes conectores surgem quando a tomada de decisão do agente depende do conhecimento que o mesmo possui do estado do recurso e das regras

Figura 6.1: *Concern* reusável que representa o recurso compartilhado

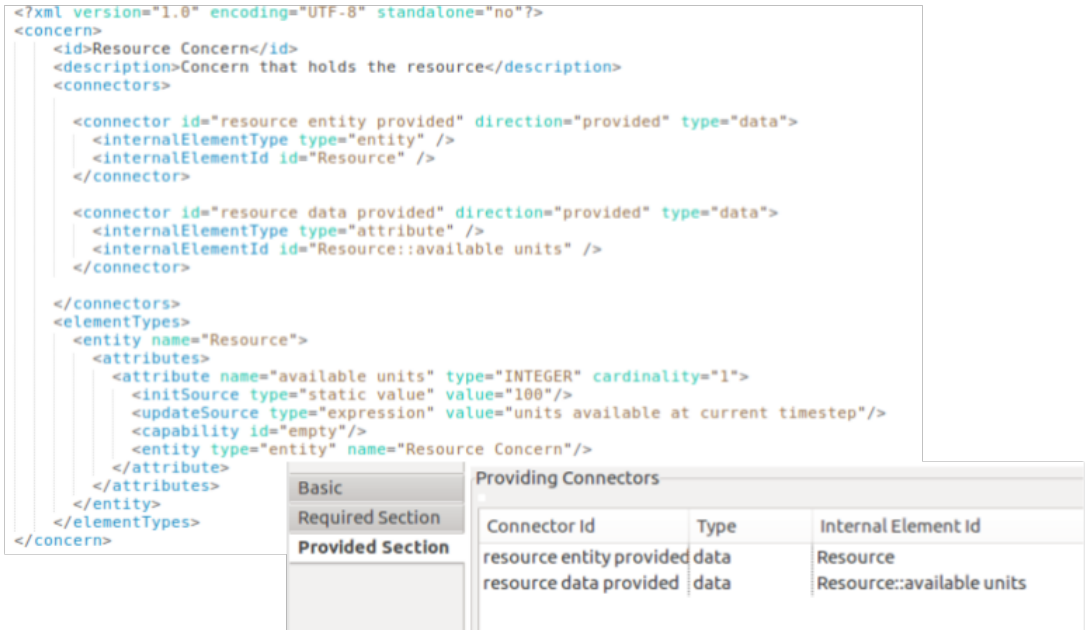



para a coleta do recurso, que são definidos em *concerns* diferentes por estarem inseridos em contextos diferentes. Para exemplificar o reuso, os três *concerns* foram implementados em projetos separados.

O primeiro *concern* reusável representa o recurso a ser compartilhado e está ilustrado na Figura 6.1 juntamente com a entidade e seu conjunto de atributos que representam o recurso. Fazem parte do conjunto de atributos do recurso as variáveis: (i) tamanho ou quantidade máxima do recurso, que é inicializada com um valor definido pelo projetista identificado como V ou *static value*; (ii) variabilidade das unidades do recurso, que é inicializada com uma expressão definida pelo projetista identificada como E ou *expression*; e (iii) unidades disponíveis ou estado atual do recurso, que é inicializada com um *static value*. Este valor é atualizado conforme uma expressão definida pelo projetista, identificada como E ou *expression*. O arquivo descritor do *concern* com os conectores da interface provida juntamente com a aba de propriedades padrão da IDE Eclipse que mostra a tabela de conectores da interface provida podem ser vistos na Figura 6.2. O *concern* que representa o recurso oferece conectores para acesso externo à entidade que descreve o recurso e às unidades disponíveis do recurso. Ambos os conectores podem ser vistos na aba de propriedades da IDE Eclipse que mostra os conectores providos. O projetista tem a liberdade de descrever conectores para acesso a outros atributos da entidade no arquivo, conforme a necessidade.

O segundo *concern* reusável representa as regras para o compartilhamento do recurso e está ilustrado na Figura 6.3 juntamente com os parâmetros do *concern* que podem ser reusados em diversas simulações a partir do reuso do mesmo. Nesta figura estão explícitas as regras que definem os parâmetros para arbitrar o recurso: (i) número máximo

Figura 6.2: Arquivo descritor do *concern* que representa o recurso compartilhado juntamente com a tabela de conectores providos




de unidades ou a quantidade de recurso que podem ser extraídos por um agente, que é inicializado com um *static value*; (ii) valor da multa conforme a severidade da infração, que é inicializado com um *static value*; (iii) benefícios oferecidos por obedecer às regras, inicializados com um *static value*; (iv) custos para obedecer as regras, inicializados com um *static value*; (v) possíveis sanções por violação de regras, podem ser a perda de benefícios oferecidos, inicializadas com uma *expression*; e (vi) normas compartilhadas entre os agentes, inicializadas com uma *expression*. O arquivo descritor do *concern* com o conector da interface provida e a aba de propriedades padrão da IDE Eclipse com a tabela de conectores da interface provida podem ser vistos na Figura 6.4. O *concern* que representa as regras para o compartilhamento do recurso oferece um conector para acesso externo ao número máximo de unidades que podem ser extraídas por um agente. Este conector pode ser observado na aba de propriedades da IDE Eclipse que mostra os conectores da interface provida. O projetista tem a liberdade de definir conectores para acesso externo a outros parâmetros no arquivo, conforme a necessidade.

O terceiro *concern* reusável mostra os agentes coletores do recurso e está ilustrado na Figura 6.5 juntamente com o agente e seu conjunto de atributos que representam os coletores do recurso. Os atributos dos agentes coletores definidos no *concern* reusável são: (i) benefícios esperados ao coletar o recurso, inicializados com um *static value*; (ii)

Figura 6.3: *Concern* reusável que representa as regras para coleta do recurso

**Rules Concern**

**Parameters**

| Name | Init. |
|---|---|
| max resource units to harvest | V |
| fines | V |
| benefits for following rule | V |
| costs for following rule | V |
| sanctions | E |
| shared norms | E |

Figura 6.4: Arquivo descritor do *concern* que representa as regras para coleta do recurso juntamente com a tabela de conectores providos

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<concern>
    <id>Rules Concern</id>
    <description>Concern that holds the rules for harvesting the resource</description>

    <connectors>

      <connector id="rule data provided" direction="provided" type="data">
        <internalElementType type="attribute" />
        <internalElementId id="Rules Concern::max resource units to harvest" />
      </connector>

    </connectors>
    <elementTypes>
      <attribute name="max resource units to harvest" type="INTEGER" cardinality="1">
        <initSource type="static value" value="5"/>
        <updateSource type="empty"/>
        <capability id="empty"/>
        <entity type="empty"/>
      </attribute>
    </elementTypes>
</concern>
```

Basic
Required Section
**Provided Section**

Providing Connectors

| Connector Id | Type | Internal Element Id |
|---|---|---|
| rule data provided | data | Rules Concern::max resource units to harvest |

custos esperados para coletar o recurso, inicializados com um *static value*; (iii) energia ou condição financeira do agente, inicializada com um *static value*. Este valor e atualizado com um *static value*; (iv) lista de estratégias para a coleta do recurso, inicializadas com uma *expression*; (v) normas internas, equivalentes aos parâmetros culturais do agente,

Figura 6.5: *Concern* reusável que representa os coletores do recurso

**Harvester Concern**

| Harvester | | | |
|---|---|---|---|
| CAP... | | | |
| ATTRIBUTES | **Name** | **Init.** | **Update** |
| | expected benefits | V | |
| | expected costs | V | |
| | energy | V | V |
| | strategy list | E | |
| | internal norms | E | |
| | resource available units | EA | |
| | max units to harvest | EA | |
| | resource to harvest | EE | |

inicializadas com uma *expression*; (vi) unidades do recurso disponíveis para coleta, inicializadas com um EA ou *external attribute*; (vii) número máximo de unidades que podem ser extraídas, inicializadas com um *external attribute*; e (vii) recurso a coletar, inicializado com um EE ou *external entity*. Os três últimos atributos necessitam de dados externos ao *concern*. O arquivo descritor do *concern* com os conectores da interface requerida, referentes a estes atributos, pode ser visto na Figura 6.6. O *concern* que representa os agentes coletores possui três conectores onde dados são requeridos para completar as funcionalidades dos atributos mencionados anteriormente. Por fim, a Tabela 6.3 mostra um resumo de todos os conectores de interfaces providas e requeridas definidos nos três *concerns* reusáveis.

Figura 6.6: Arquivo descritor do *concern* que representa os coletores do recurso

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<concern>
    <id>Harvester Concern</id>
    <description>Concern that holds the harvester agents</description>
    <connectors>

      <connector id="resource data required" direction="required" type="data">
        <internalElementType type="attribute" />
        <internalElementId id="Harvester::resource available units" />
      </connector>

      <connector id="rule data required" direction="required" type="data">
        <internalElementType type="attribute" />
        <internalElementId id="Harvester::max units to harvest" />
      </connector>

      <connector id="resource entity required" direction="required" type="data">
        <internalElementType type="attribute" />
        <internalElementId id="Harvester::resource to harvest" />
      </connector>

    </connectors>

    <elementTypes>
      <agent name="Harvester">

        <attributes>
          <attribute name="resource available units" type="EXTERNAL" cardinality="1">
            <initSource type="external attribute"/>
            <updateSource type="external attribute"/>
            <capability id="empty"/>
            <entity type="agent" name="Harvester"/>
          </attribute>

          <attribute name="max units to harvest" type="EXTERNAL" cardinality="1">
            <initSource type="external attribute"/>
            <updateSource type="empty"/>
            <capability id="empty"/>
            <entity type="agent" name="Harvester"/>
          </attribute>

          <attribute name="resource to harvest" type="EXTERNAL" cardinality="1">
            <initSource type="external entity"/>
            <updateSource type="empty"/>
            <capability id="empty"/>
            <entity type="agent" name="Harvester"/>
          </attribute>
        </attributes>
      </agent>
    </elementTypes>
</concern>
```

Tabela 6.3: Conectores das interfaces providas e requeridas

| ***Concern*** | **Interface** | **Elemento** | **Elemento que Requer** | **Elemento Requerido** | **Elemento Provido** |
|---|---|---|---|---|---|
| *Resource* | Provida | *entity* | - | - | A entidade *Resource* |
| *Resource* | Provida | *entity.attribute* | - | - | Atributo contido na entidade *Resource*: unidades do recurso disponíveis |
| *Rules* | Provida | *concern.parameter* | - | - | Parâmetro do *concern Rules*: quantidade máxima de unidades do recurso a coletar |
| *Harvester* | Requerida | *agent.attribute* | Atributo do agente: recurso a coletar | A entidade *Resource* do *concern Resource* - *entity* | - |
| *Harvester* | Requerida | *agent.attribute* | Atributo do agente: unidades do recurso disponíveis | Atributo da entidade *Resource* do *concern Resource*: unidades do recurso disponíveis - *entity.attribute* | - |
| *Harvester* | Requerida | *agent.attribute* | Atributo do agente: quantidade máxima de unidades do recurso a coletar | Parâmetro do *concern Rules*: quantidade máxima de unidades do recurso a coletar - *concern.parameter* | - |

## 6.3 Resultados

Após a modelagem dos *concerns* reusáveis, são modeladas as simulações alvo na ferramenta ABStractme juntamente com as injeções de dependência entre *concerns*. As simulações definidas na Tabela 6.1 possuem cada uma o seu projeto em que os *concerns* modelados anteriormente foram importados. No caso do exemplo modelado a partir do trabalho de Schlüter, Tavoni and Levin (2016), a visão geral da simulação pode ser vista na Figura 6.7, onde é possível visualizar os *concerns* da simulação que foram modelados conforme a definição dos mesmos que aparece na Tabela 6.2. Todos os *concerns* foram importados a partir dos *concerns* reusáveis modelados anteriormente. A Figura 6.8 mostra uma visão geral de todas as conexões entre *concerns* da simulação feitas através da aba *Required Section* das propriedades padrão da IDE Eclipse.

A visão geral das conexões destaca os relacionamentos estabelecidos entre os *concerns*: (i) o agente coletor precisa saber a quantidade máxima de unidades do recurso que pode coletar conforme as regras de coleta. Dessa forma, o *concern* do agente coletor re-

Figura 6.7: Visão geral da simulação

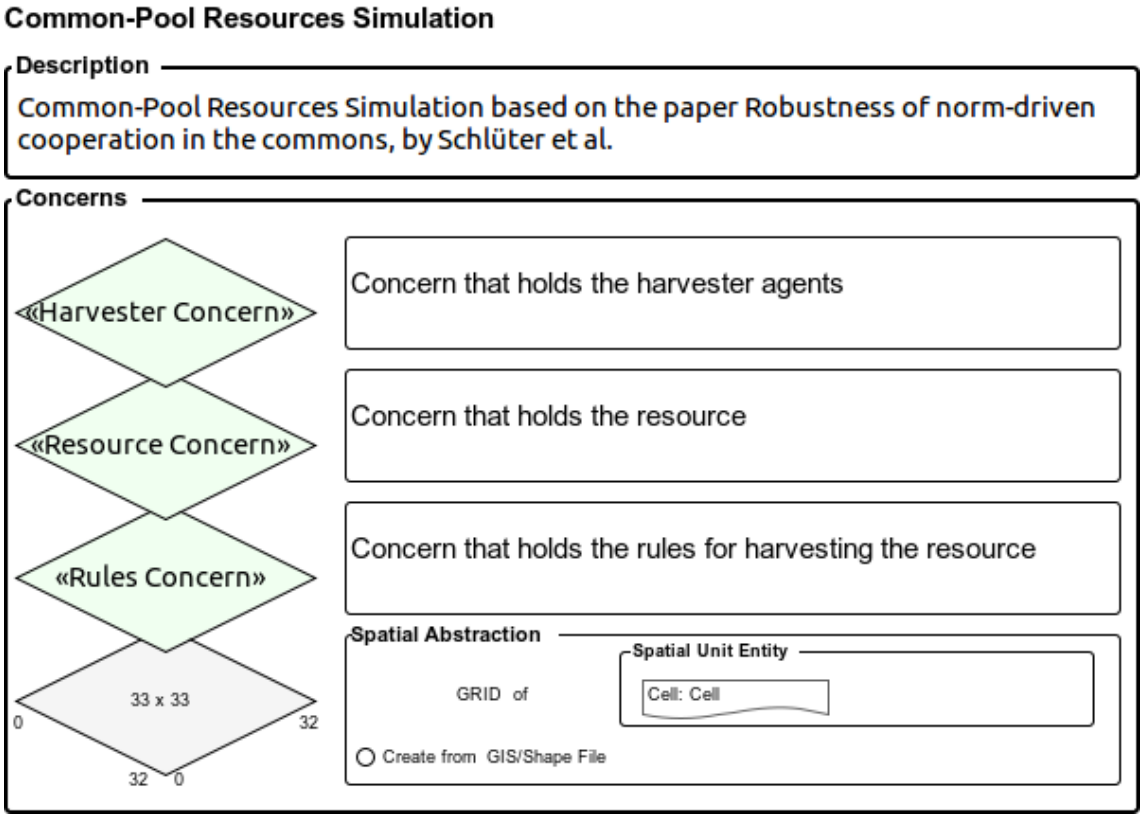
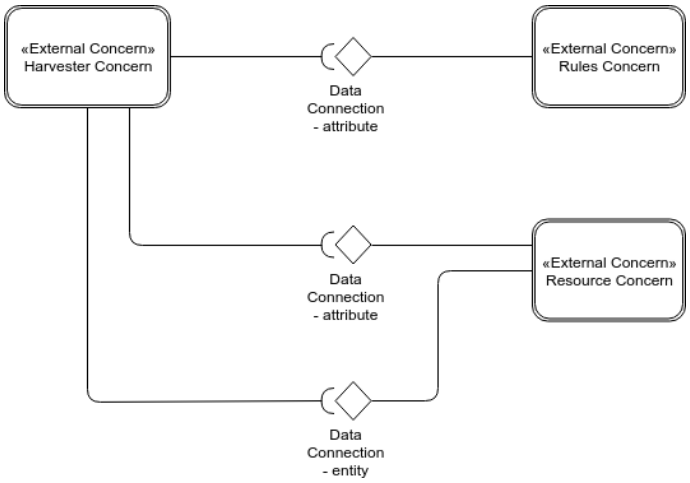


Figura 6.8: Visão geral das conexões entre *concerns*



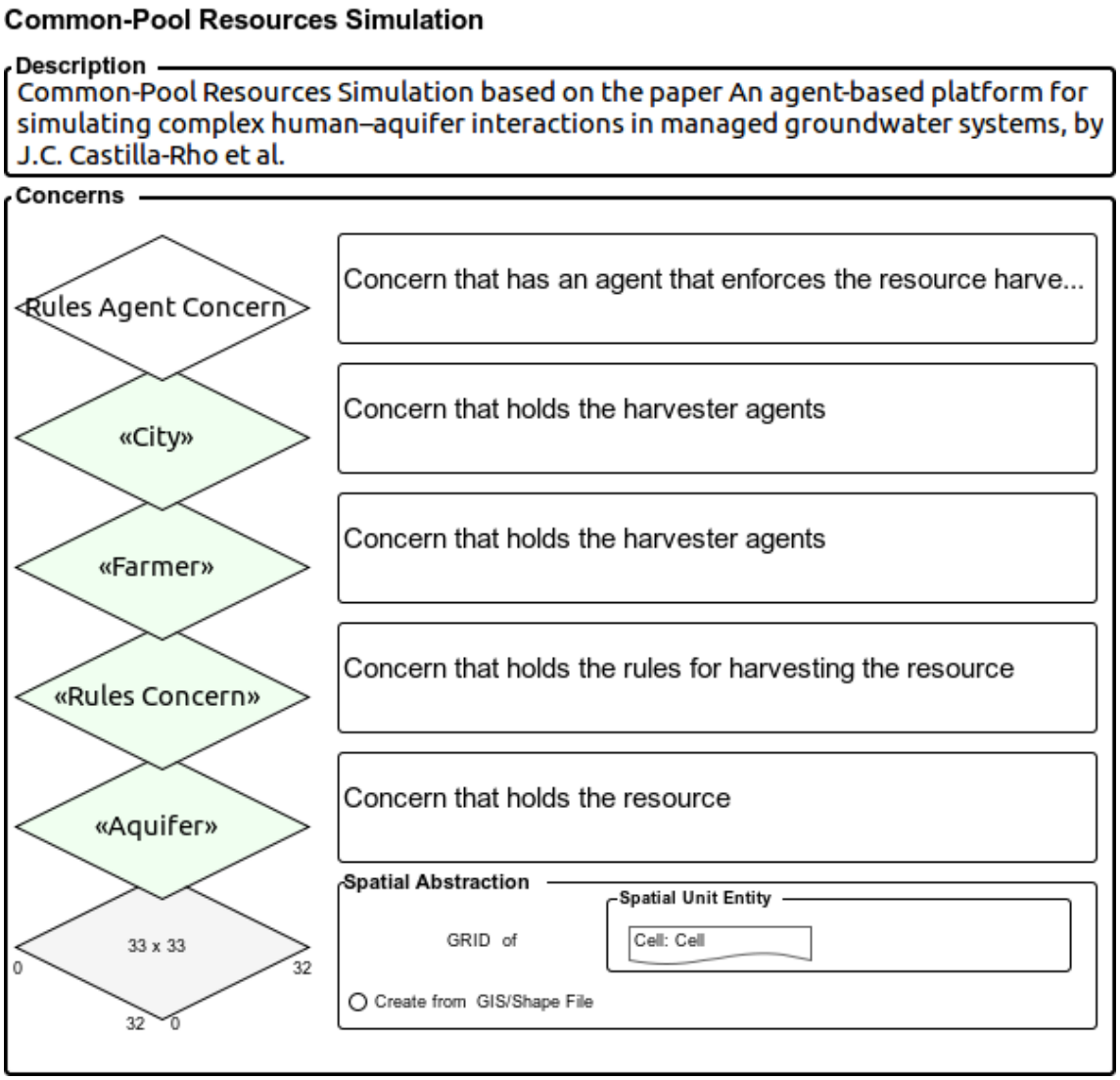

quer este dado do *concern* que possui as regras de coleta do recurso; (ii) para que o agente coletor saiba qual recurso deve coletar, o *concern* do agente coletor requer a identificação da entidade que representa o recurso que se encontra no *concern* do recurso; e (iii) o agente coletor precisa saber as unidades disponíveis do recurso, neste caso o *concern* do agente coletor requer esse dado do *concern* do recurso.

No exemplo feito a partir do trabalho de Castilla-Rho et al. (2015), a visão geral da simulação pode ser vista na Figura 6.9, onde são mostrados os *concerns* da simulação que foram modelados conforme a definição dos mesmos que aparece na Tabela 6.2. Foi necessário criar um *concern* para o agente que representa os reguladores dos poços para coleta de água. Este *concern* requer os dados da entidade que representa o aquífero, que se encontra no *concern Aquifer*, que é um *concern* do tipo *Resource Concern* importado, que teve seu título alterado para *Aquifer*. Os *concerns Farmer* e *City* são ambos *concerns* importados do tipo *Harvester Concern* e tiveram seus títulos alterados para diferenciar um do outro. A Figura 6.10 mostra uma visão geral de todas as conexões entre *concerns* da simulação feitas através da aba *Required Section* das propriedades padrão da IDE Eclipse.

A visão geral das conexões evidencia quais são os relacionamentos estabelecidos entre os *concerns*. Os *concerns Farmer* e *City* se relacionam com os *concerns Aquifer* e *Rules* de forma análoga ao relacionamento entre *concerns* apresentado na simulação alvo modelada a partir do trabalho de Schlüter, Tavoni and Levin (2016). O *concern* que contém os reguladores dos poços para coleta de água requer a identificação da entidade que representa o aquífero contida no *Aquifer concern* para monitorar o seu estado.

Em ambos os casos, a maior parte dos elementos da simulação encapsulados nos

Figura 6.9: Visão geral da simulação

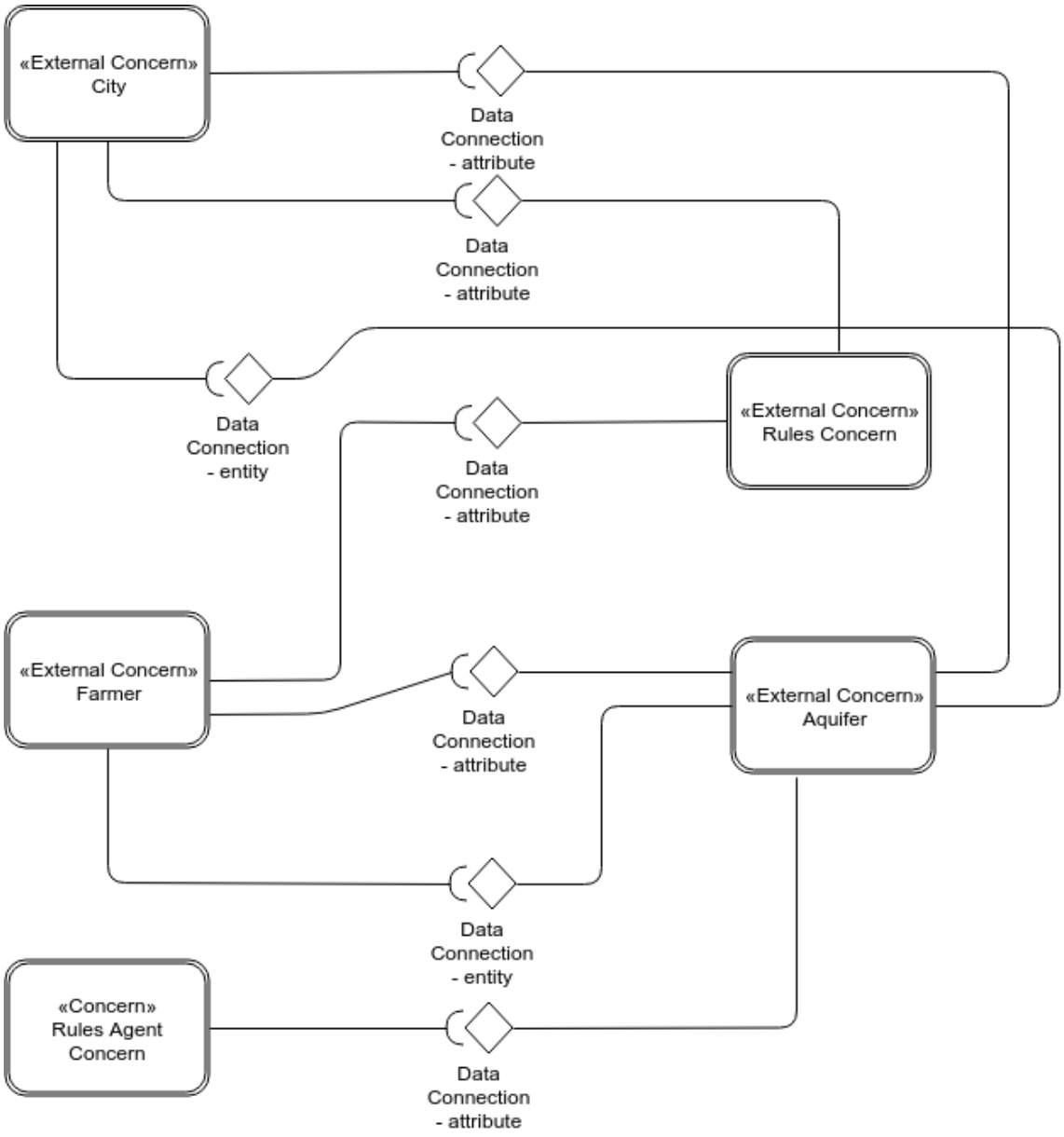



*concerns* e dos conectores das interfaces requerida e provida são definidos a partir dos *concerns* reusáveis modelados na seção 5.2, o que resultou em um reuso significativo de software na modelagem de cada simulação alvo. Os projetos foram feitos com uma versão simplificada das simulações, e podem ser expandidos com outros *concerns* externos e mais conectores de interface, conforme a necessidade do projetista.

## 6.4 Considerações Finais

A partir da definição de simulações que possuam pontos em comum, no caso o compartilhamento de um recurso que é coletado e gerenciado pelos agentes, é possível

Figura 6.10: Visão geral das conexões entre *concerns*




exemplificar o reuso de software ao modelar *concerns* reusáveis que encapsulam os pontos em comum destas simulações na ferramenta ABStractme. A modelagem das simulações com a importação destes *concerns* reusáveis deixa explícita a possibilidade de reuso de software a partir do refinamento da definição de *concerns* proposto neste trabalho.

# 7 CONCLUSÃO

O *concern* foi introduzido na abordagem MDD4ABMS para facilitar a modularização e a escalabilidade do modelo da simulação. Este trabalho estendeu o conceito de *concern* definido na abordagem para permitir o seu reuso em diferentes simulações. A partir do estudo de trabalhos relacionados, foi constatada a importância dada à modularidade em sistemas multiagente juntamente com a falta de foco na necessidade de haver um baixo acoplamento das abstrações da simulação para possibilitar o seu reuso. Com base nesta lacuna, foi proposto um refinamento da definição do *concern* que introduziu um baixo acoplamento para este elemento sem comprometer a sua coesão. A ferramenta ABStractme foi estendida para permitir a especificação de *concerns* reusáveis, e também para reusar estes *concerns* ao projetar simulações com agentes. Um estudo de caso foi realizado para demonstrar como os *concerns* podem ser reusados em simulações diferentes, mas que possuem pontos em comum.

O objetivo deste trabalho é expandir o conceito de *concerns* para facilitar o seu reuso, que é uma lacuna na abordagem MDD4ABMS. Ao utilizar um *concern* abstrato com funcionalidades que devem ser completadas e uma interface que suporta conexões que completam estas funcionalidades, o projetista tem o seu retrabalho reduzido ao modelar simulações similares. Além disso, o reuso facilita ainda mais o aprendizado em ABMS, que é um dos objetivos da linguagem ABStractLang, e introduz um aumento na confiabilidade da ferramenta ABStractme ao disponibilizar *concerns* padronizados.

As principais contribuições deste trabalho são: (i) no Capítulo 4, refinamento da definição do *concern* que consiste na introdução de uma interface para o mesmo. A partir deste refinamento, foi apresentada uma linguagem para descrever os conectores da interface requerida e provida do *concern* para melhorar o seu encapsulamento, escondendo abstrações que não precisam ser visualizadas no nível global da simulação, permitindo um maior controle do projetista sobre o que é compartilhado entre *concerns* e facilitando o reuso dos mesmos; (ii) no mesmo capítulo, introdução de uma linguagem para descrever a injeção de dependência entre *concerns*, que ocorre externamente aos *concerns* e permite o desacoplamento dos mesmos; (iii) no Capítulo 5, extensão da ferramenta ABStractme para permitir a importação de *concerns* externos ao projeto, o que permite o reuso dos componentes contidos nestes *concerns*; e (iv) no Capítulo 6, estudo de caso de *concerns* reusáveis no contexto de compartilhamento e gerenciamento de recursos entre diversos agentes, para ilustrar as vantagens no reuso de *concerns* e seus descritores dos conectores

de interface.

Entre os trabalhos futuros está a necessidade de gerar uma biblioteca com *concerns* que possam ser usados em simulações de diversos domínios para demonstrar as vantagens no reuso de *concerns* e disponibilizar tutoriais de uso para esclarecer como são feitos os arquivos descritores dos conectores, a injeção de dependência através da aba de propriedades padrão da IDE Eclipse e a importação de *concerns*. Por fim, para possibilitar a geração de código para a plataforma NetLogo a partir dos *concerns* reusáveis e das injeções de dependência entre eles, é necessário introduzir mudanças no gerador de código da simulação. A seguir são descritas as mudanças que precisarão ser realizadas no gerador de código. Como um primeiro passo já implementado, foram criados tipos de atributos para indicar o atributo que requer dados de fora do *concern* que podem estar localizados em um componente do tipo atributo externo ou entidade externa. O gerador de código deve incluir um passo para ler os arquivos XML dos descritores dos conectores das interfaces provida e requerida de cada *concern* e dos descritores das conexões entre *concerns*. Por fim, a partir desta leitura, para cada elemento requerido serão geradas instruções de código que acessam os elementos providos que se encontram em outras partes da simulação, materializando desta forma a injeção de dependência.

# REFERÊNCIAS

AL-ZINATI, M. et al. Divas 4.0: A multi-agent based simulation framework. In: **2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications**. [S.l.: s.n.], 2013. p. 105–114. ISSN 1550-6525.

APELDOORN, D. A spatio-temporal multiagent simulation framework for reusing agents in different kinds of scenarios. In: **Revised Selected Papers of the 13th German Conference on Multiagent System Technologies - Volume 9433**. New York, NY, USA: Springer-Verlag New York, Inc., 2015. (MATES 2015), p. 79–97. ISBN 978-3-319-27342-6. Available from Internet: <http://dx.doi.org/10.1007/978-3-319-27343-3_5>.

BUSETTA, P. et al. Structuring bdi agents in functional clusters. In: JENNINGS, N. R.; LESPÉRANCE, Y. (Ed.). **Intelligent Agents VI. Agent Theories, Architectures, and Languages**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 277–289. ISBN 978-3-540-46467-9.

CASTILLA-RHO, J. et al. An agent-based platform for simulating complex human–aquifer interactions in managed groundwater systems. **Environmental Modelling Software**, v. 73, p. 305 – 323, 2015. ISSN 1364-8152. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S136481521530044X>.

EPSTEIN, J. M.; AXTELL, R. **Growing Artificial Societies: Social Science from the Bottom Up**. Washington, DC, USA: The Brookings Institution, 1996. ISBN 0-262-55025-3.

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.

GERSHENSON, C. Self-organizing traffic lights. **Complex Systems**, v. 16, n. 1, p. 29–53, 2005.

KLÜGL, F.; BAZZAN, A. L. C. Agent-based modeling and simulation. **AI Magazine**, v. 33, n. 3, p. 29–40, 2012.

KRAVARI, K.; BASSILIADES, N. A survey of agent platforms. **Journal of Artificial Societies and Social Simulation**, v. 18, n. 1, p. 11, 2015. ISSN 1460-7425.

MACAL, C.; NORTH, M. Introductory tutorial: Agent-based modeling and simulation. In: **Proceedings of the Winter Simulation Conference 2014**. [S.l.: s.n.], 2014. p. 6–20. ISSN 0891-7736.

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.

MOREIRA, D. et al. ABStractme: Modularized environment modeling in agent-based simulations. In: DAS, S. et al. (Ed.). **Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems**. São Paulo: IFAAMAS, 2017. p. 1802–1804.

NUNES, I. Capability relationships in bdi agents. In: **Proceedings of the 2nd International Workshop on Engineering Multi-Agent Systems (EMAS 2014)**. Paris: [s.n.], 2014. p. 56–72.

ORTIZ-HERNÁNDEZ, G. et al. A namespace approach for modularity in bdi programming languages. In: BALDONI, M. et al. (Ed.). **Engineering Multi-Agent Systems**. Cham: Springer International Publishing, 2016. p. 117–135. ISBN 978-3-319-50983-9.

OSTROM, E. **Governing The Commons: The Evolution Of Institutions For Collective Action**. [S.l.: s.n.], 1990. ISBN 0521371015.

RAILSBACK, S. F.; LYTINEN, S. L.; JACKSON, S. K. Agent-based simulation platforms: Review and development recommendations. **SIMULATION**, v. 82, n. 9, p. 609–623, 2006.

RAO, A. S.; GEORGEFF, M. P. Bdi agents: From theory to practice. In: **Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95**. [S.l.: s.n.], 1995. p. 312–319.

SANTOS, F. **Domain-specific Language for Agent-based Modeling and Simulation**. 2018. Available from Internet: <http://www.inf.ufrgs.br/prosoft/resources/dsl4abms/>.

SANTOS, F. **Model-Driven Agent-based Simulation Development**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2019.

SANTOS, F.; NUNES, I.; BAZZAN, A. Model-driven agent-based simulation development: a modeling language and empirical evaluation in the adaptive traffic signal control domain. **Simulation Modelling Practice and Theory**, April 2017. (submitted).

SANTOS, F.; NUNES, I.; BAZZAN, A. Model-driven engineering in agent-based modeling and simulation: a case study in the traffic signal control domain. In: DAS, S. et al. (Ed.). **Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)**. São Paulo: IFAAMAS, 2017. p. 1725–1727.

SANTOS, F.; NUNES, I.; BAZZAN, A. Supporting the development of agent-based simulations: a DSL for environment modeling. In: **Proceedings of the IEEE Computer Software and Applications Conference (COMPSAC 2017)**. Torino: [s.n.], 2017. p. 170–179.

SANTOS, F.; NUNES, I.; BAZZAN, A. L. C. A case study of the development of an agent-based simulation in the traffic signal control domain using an MDD approach. In: **Proceedings of the 5th International Workshop on Engineering Multi-Agent Systems (EMAS 2017)**. São Paulo: [s.n.], 2017. p. 97–112.

SCERRI, D. et al. An architecture for modular distributed simulation with agent-based models. In: **Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1**. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2010. (AAMAS '10), p. 541–548. ISBN 978-0-9826571-1-9. Available from Internet: <http://dl.acm.org/citation.cfm?id=1838206.1838283>.

SCHLÜTER, M.; TAVONI, A.; LEVIN, S. Robustness of norm-driven cooperation in the commons. **Proceedings of the Royal Society B: Biological Sciences**, v. 283, 01 2016.

SMITH, R. G. The contract net protocol: High-level communication and control in a distributed problem solver. **IEEE Trans. Comput.**, IEEE Computer Society, Washington, DC, USA, v. 29, n. 12, p. 1104–1113, dec 1980. ISSN 0018-9340. Available from Internet: <https://doi.org/10.1109/TC.1980.1675516>.

WARWAS, S.; KLUSCH, M. Making multiagent system designs reusable: A model-driven approach. In: **2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology**. [S.l.: s.n.], 2011. v. 2, p. 101–108.