ISSN 2177-2096

WESAAC 2015

9º Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações

Proceedings WESAAC 2015

Nono Workshop-Escola de Sistemas de Agentes, seus Ambientes e ApliCações

01 - 03 de Junho 2015 | Niterói • Rio de Janeiro

Editores:
Baldoino Fonseca (UFAL)
Viviane Torres da Silva (UFF)
Ricardo Choren (IME)

CNPq FAPERJ CAPES FAAMAS UFF Instituto de Computação UNIVERSIDADE FEDERAL DE ALAGOAS IME

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária Responsável: Maria Helena Mendes Lessa

W926 Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (9. : 2015 : Niterói, RJ)/
Anais do IX Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações – IX WESAAC / Baldoino Fonseca, Viviane Torres da Silva, Ricardo Choren (Eds.). – Niterói: UFF, 2015.
232 p. : il.

Inclui bibliografia.
ISSN 2177-2096

1. Agentes. 2. Sistemas multi-agentes. 3. Simulações. 4. Aplicações. 5. Congresso. I. Fonseca, Baldoino. II. Silva, Viviane Torres da. III. Choren, Ricardo. IV. WESAAC (9. : 2015 : Niterói, RJ). V. Título.

CDU: 004(05)

I. Artigos Completos

II. Pôsters

Construindo Sistemas Multi-Agentes no Contexto de Internet das Coisas

Chrystinne O. Fernandes[1], Carlos J. P. Lucena[2]

[1]Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) CEP 22453-900 – Rio de Janeiro – RJ – Brasil

{cfernandes,lucena}@inf.puc-rio.br

Abstract. *Aiming to expand the set of disciplines that intersects the Multi-Agent Systems (MAS) area, this paper brings the concept of agents to the Internet of Things (IoT). It also presents the results of two case studies, involving a luggage tracking simulation in an airport and a prototype for sensing and monitoring patients and their environment. The solution showed that it is possible to make more proactive environments. The agents were able to detect problems in patients' health status and alert healthcare professionals responsible for their care. The studies assisted modeling and design of a framework for monitoring things based on an agent model.*

Resumo. *Visando ampliar o conjunto de disciplinas que tem interseção com a área de Sistemas Multi-Agentes (SMA), este artigo traz o conceito de agentes para a Internet of Things (IoT). Também apresenta os resultados de dois estudos de caso, envolvendo uma simulação de rastreamento de bagagens em um aeroporto e um protótipo para sensoriamento e monitoramento de pacientes e seu ambiente. A solução mostrou que é possível tornar os ambientes mais proativos. Os agentes foram capazes de detectar problemas no estado de saúde dos pacientes e alertar os profissionais responsáveis pelo seu tratamento. O experimento auxiliou na modelagem e projeto de um framework para monitoramento de coisas, baseado em um modelo de agentes.*

1. Introdução

Estamos vivendo um período em que a tecnologia está sendo cada vez mais usada para melhorar nossa qualidade de vida, seja aumentando nossa segurança, conforto e bem-estar, seja otimizando recursos humanos limitados como o tempo ou a capacidade de armazenar e processar dados, a fim de extrair deles informação útil e gerar novos conhecimentos. A *Internet of Things* (IoT) é uma área relativamente recente que vem garantindo seu espaço nesse processo de mudanças tecnológicas que se reflete em nossos hábitos cotidianos e em nossa cultura de modo geral.

A ideia básica por trás do conceito de IoT é ter um cenário onde coisas, pessoas e animas são identificados de maneira única e podem comunicar-se pela Internet, com pouca ou nenhuma intervenção humana. Vale salientar que as coisas, no âmbito da IoT, podem ser entendidas como entidades do mundo real que vão fazer parte da rede IoT, compreendendo desde simples objetos como uma cadeira, um eletrodoméstico ou um carro, até seres vivos como plantas, animais e pessoas.

Desse modo, interações na rede, que até então estavam restritas aos tipos *human-to-human* ou *human-to-computer*, estão sendo ampliadas para suportar outros conceitos de interação envolvendo coisas, como os tipos *human-to-thing* e *thing-to-thing*.

Materializando este conceito de Internet das Coisas, atualmente já temos exemplos de sua aplicação em diversas áreas das nossas vidas, como transporte e logística e cuidados médicos. A tendência é que haja uma expansão do alcance da IoT, de modo a possibilitar o surgimento de ambientes cada vez mais inteligentes, os quais vislumbramos que possam evoluir para a dimensão de cidades inteligentes, que é um dos temas de pesquisa amplamente explorados neste campo.

Podemos relacionar diversos problemas que são bastante comuns em nossas cidades e retardam a ocorrência de avanços sociais, como por exemplo: a gestão ineficiente de recursos públicos, que leva a desperdícios de verbas e à consequente insatisfação da população; O alto congestionamento de veículos que impacta negativamente a vida das pessoas, consumindo seu tempo e muitas vezes promovendo stress; O uso de processos manuais de coleta de dados propensos a erros; Além da utilização de técnicas obsoletas de identificação de coisas;

A busca de soluções para problemas desta natureza serve de motivação para este trabalho, cujo enfoque principal consiste no desenvolvimento de sistemas multi-agentes no contexto de Internet das Coisas, a fim de que o conceito de cidades inteligentes torne-se cada vez mais próximo da nossa realidade. Com o intuito de fornecer um suporte ao desenvolvimento de sistemas deste tipo, foi idealizado neste trabalho o projeto de um framework para construção de aplicações IoT, com base em um modelo de agentes de software.

A estratégia utilizada para alcançar este objetivo consistiu, inicialmente, em encontrar cenários com grande potencial para exploração dos conceitos de IoT e que também fossem alvos para a ampla aplicação de agentes. Nesta etapa, foram priorizados dois domínios de aplicação: o primeiro foi mais específico - *track and tracing* -, onde buscou-se analisar o rastreamento de coisas através de simulações; Já o segundo, além de mais geral, foi explorado através de um protótipo construído durante este trabalho para auxiliar no monitoramento de pacientes e correspondeu ao domínio de e-health.

Foram realizados dois estudos de caso, um para cada domínio, onde ambos foram planejados para investigar como se davam as interações *human-to-thing* e *thing-to-thing* em uma rede IoT, além de como os agentes de software iriam se comportar no contexto da IoT e se o uso dos agentes tornaria o ambiente mais proativo, interativo e inteligente. Além do levantamento destas e de outras questões de pesquisa, na fase de planejamento do experimento também foram definidas algumas hipóteses com base no conhecimento prévio sobre o assunto. Tanto as questões quanto as hipóteses serão apresentadas mais adiante, na seção 3. Ao final de cada estudo de caso, foram discutidos os resultados do experimento. Nesta etapa, foram analisadas as hipóteses levantadas, para confirmá-las ou refutá-las, através do seu confronto com os dados obtidos experimentalmente. Esta discussão foi feita com o intuito de tentar responder as questões de pesquisa e registrar possíveis conhecimentos empíricos que possam ter surgido acerca do tema.

A realização dos estudos de caso possibilitou o levantamento de um conjunto de *features* necessárias às etapas iniciais de modelagem e projeto do framework, cuja implementação é um dos trabalhos futuros sugeridos neste artigo.

2. Fundamentação Teórica

2.1. Sistemas Multi-Agentes

Um agente é um elemento de um sistema computacional que está situado em algum ambiente no qual é capaz de realizar ações autônomas, a fim de cumprir os objetivos que

lhe foram delegados. Apesar de não existir uma definição universal do termo agente, existe um consenso na literatura de que a autonomia é uma característica chave para a atuação do agente [Wooldridge 2009]. Neste contexto, autonomia significa a possibilidade de atuação sem a intervenção humana ou de outros sistemas.

Entretanto, mesmo tendo controle sobre seu estado interno e comportamento, o agente não tem controle total sobre seu ambiente. Ele possui um conjunto de ações que pode realizar e cuja execução pode resultar na modificação do ambiente. Por essa razão, considera-se que o agente tem controle parcial sobre seu ambiente, podendo influenciá-lo, de acordo com a ação que decidir realizar. Desse modo, um agente utiliza sua autonomia para decidir como agir de modo a satisfazer seus objetivos. Outras propriedades dos agentes, por sua vez, são desejáveis apenas em domínios específicos, como é o caso do aprendizado, ou seja, a capacidade que possui de aprender a partir da sua experiência.

Um agente pode ocupar diferentes tipos de ambientes, que se classificam de acordo com suas propriedades, em [Russell and Norvig 2013]: **Acessível ou inacessível**, dependendo da informação que o agente possui sobre ele. Diz-se que o ambiente é acessível quando o agente pode obter informações completas, precisas e atualizadas sobre o estado do ambiente; **Determinístico ou não-determinístico**, de acordo com a garantia de efeito que uma determinada ação terá neste ambiente. Um ambiente determinístico é aquele em que qualquer ação tem a garantia de um efeito único – não existe incerteza sobre o estado que irá resultar da execução de uma ação; **Estático ou dinâmico**, dependendo do controle que o agente tem sobre as mudanças no ambiente. Um ambiente pode ser considerado estático quando pode-se assumir que ele irá permanecer inalterado, exceto pela execução de uma ação por parte do agente. Em contrapartida, um ambiente dinâmico é aquele que tem outros processos operando sobre ele e que as mudanças ocorridas nele estão fora do controle do agente; E, por último, um ambiente pode ser **discreto ou contínuo**, considerando as possíveis ações e percepções que o agente pode realizar. Em um ambiente discreto, existe um número finito e fixo de ações e percepções.

Tomando como exemplo os agentes do estudo de caso na área de saúde, podemos considerar seu ambiente inacessível, não-determinístico, dinâmico e contínuo. Neste contexto, é inacessível por não ser possível ter conhecimento completo sobre os dados do ambiente, uma vez que eles refletem o estado de saúde do paciente e são coletados em tempo real. É não-determinístico porque não é possível garantir que cada ação estará associada à exatamente um único efeito possível. Dinâmico, pois o ambiente está em constante mudança, de acordo com eventos que ocorrem e que fogem ao controle dos agentes. E, por fim, contínuo, já que existem infinitas e variáveis ações que podem acontecer neste cenário.

Agentes inteligentes tem capacidades como reatividade, proatividade e habilidade social, as quais diferem pelo modo como os ajudam a satisfazerem seus objetivos. A reatividade diz respeito à capacidade de perceber e responder tempestivamente às mudanças no ambiente. Já a proatividade é a capacidade de ter um comportamento orientado ao seu objetivo, através da tomada de iniciativa. E a habilidade social é a capacidade de interagir com outros agentes e, possivelmente, com humanos. Ela não envolve apenas troca de informações, mas também a possibilidade de realizarem relações de negociação e cooperação [Russell and Norvig 2013].

2.2. Inteligência Artificial (IA)

Inteligência Artificial é o campo que tenta não apenas compreender, mas construir entidades inteligentes [Russell and Norvig 2013], que podem ser interpretadas como agentes de software. O conceito de agente é amplamente abordado em IA, bem como o uso de técnicas que podem ser utilizadas pelos agentes para execução de tarefas e tomada de decisões.

Outro aspecto bastante explorado em IA é o processo de aprendizagem. O aprendizado é uma das possíveis características dos agentes. Pode-se dizer que um agente irá aprender se conseguir melhorar o seu desempenho nas tarefas futuras a partir da observação do mundo. Dessa forma, o conhecimento sobre o mundo faz-se necessário para que os agentes possam tomar boas decisões. Este conhecimento é armazenado em uma base de conhecimento, através de uma linguagem de representação do conhecimento. [Russell and Norvig 2013].

2.3. Internet of Things

No final da década de noventa, tínhamos o seguinte cenário na Internet: um grande poder computacional estava acessível, mas a entrada de dados na rede era feita quase exclusivamente por seres humanos, que em geral tem recursos de tempo bastante limitados. Este fato gerava um gargalo no canal de entrada de dados na rede, já que a capacidade de processamento era muito superior à demanda de dados. Para tentar resolver limitações como esta, surgiu o conceito de *Internet of Things*, onde coisas também estariam habilitadas a realizar operações de entrada e saída de dados e processamento de informações na Internet.

A ideia era que as coisas fossem identificadas e conectadas à Internet, com a capacidade de se comunicarem de modo independente ou com intervenção mínima dos seres humanos. Esta mudança visava automatizar atividades humanas dispendiosas como a de coletar dados e inseri-los na rede. Desse modo, tarefas como esta passaram a ser delegadas às próprias coisas, que poderiam interagir entre si e também com pessoas e sistemas.

Aplicações IoT tem gerado impacto em diversas áreas da nossa vida: na área pessoal, por exemplo, surgiram os conceitos de *domotics* e *smart house*; No campo profissional, surgiram os *smart offices*; Na Indústria, aplicações no ramo de transporte e logística despontam como grandes promessas para otimização de recursos; Na Administração pública, estamos experimentando aplicações que tratam da gestão mais eficiente de recursos públicos. E, por último, temos a área de saúde, um dos domínios de aplicação dos estudos de caso deste trabalho, que vem explorando conceitos como *e-health* e *Assisted living*.

Existem diversas tecnologias que podem ser utilizadas no desenvolvimento de aplicações IoT. A concepção deste trabalho foi baseada no conceito de três tecnologias que foram fundamentais para embasar nossas simulações: RFID (*Radio-Frequency IDentification*), microcontroladores Arduino e Sensores.

RFID ou identificação por rádio frequência é um método de identificação automática que utiliza sinais de rádio, recuperando e armazenando dados remotamente através de dispositivos denominados etiquetas ou tags RFID. Estes dispositivos são usados para fins de identificação, sensoriamento e comunicação [Atzori 2010].

O arduino, por sua vez, é uma plataforma de prototipagem eletrônica, open-source, flexível, disponível na versão stand-alone ou para comunicação com software

executando em computador pessoal. Por serem baseados em hardware e software considerados fáceis de usar, microcontroladores Arduino são destinados a qualquer pessoa que queira desenvolver projetos interativos [Doukas 2012].

A terceira e última tecnologia utilizada foram os sensores. Diversos tipos de sensores podem ser utilizados para coletar dados úteis em aplicações IoT, como a frequência de batimentos cardíacos, nível de açúcar no sangue (para controlar pacientes diabéticos), taxa respiratória, temperatura, umidade, nível de líquido, nível de oxigênio, nível de gás carbônico, acelerômetro, entre outros.

3. Metodologia

3.1. Método Experimental

Conforme mencionado anteriormente, os métodos utilizados para responder as questões de pesquisa consistiram na realização de dois estudos de caso, abordando os domínios de *track and tracing e e-health*, os quais serão detalhados nas próximas subseções, 3.4 e 3.5.

3.2. Questões de Pesquisa (Q)

Q1. Como os agentes de software comportam-se no contexto da IoT?

Q2. O uso dos agentes pode tornar o ambiente mais proativo e interativo?

3.3. Hipóteses (H)

H1. O uso de agentes facilita a comunicação entre os elementos da rede IoT.

H2. O uso de agentes pode tornar o ambiente mais proativo.

H3. O uso de agentes pode tornar o ambiente mais interativo.

3.4. Estudo de Caso 1: Rastreamento de bagagens

3.4.1 Definição do Problema

Atualmente, a grande maioria dos aeroportos ainda não conseguiu superar problemas corriqueiros e que provocam uma série de transtornos aos usuários de seus serviços. Problemas como extravios de bagagens ainda são recorrentes, mesmo com todo o aparato tecnológico que dispomos hoje em dia. Além disso, os passageiros, muitas vezes, veem-se obrigados a acompanharem atentamente os avisos nos monitores espalhados pelo salão de embarque. Isto ocorre porque precisam se certificar de que não houve mudança de última hora em seu portão de embarque, fato que é bastante comum acontecer. Outro inconveniente constante é a poluição sonora provocada pelas últimas chamadas para embarque dos passageiros retardatários e avisos de atraso nos voos.

3.4.2 Solução Proposta

Como uma alternativa para tentar solucionar problemas como os citados, foi proposto o desenvolvido de um sistema multi-agentes, no âmbito de Internet das Coisas, através do qual seriam simuladas algumas das situações citadas na subseção anterior. Com isto, pretendia-se, principalmente: Aumentar a inteligência do ambiente simulado, bem como a capacidade de comunicação dos seus elementos (*Smart Environments domain*); Rastrear as bagagens dos passageiros (*Track and Tracing domain*); Monitorar os ambientes nos quais animais eram transportados durante os voos (*Monitoring domain*); Aumentar a confiança dos passageiros no transporte de seus animais, através do sensoriamento de

suas condições físicas e de seu ambiente *(Sensing domain*); Realizar, de forma autônoma, atividade como os alertas aos passageiros feitos pelas companhias aéreas, a partir do uso de agentes de software;

3.4.3 Objetivos

O objetivo deste estudo de caso foi desenvolver um sistema no qual fossem realizadas simulações de situações práticas que ocorrem em aeroportos, com o intuito de alcançar os seguintes resultados específicos: Fornecer estratégias para minimizar problemas recorrentes nos aeroportos; Identificar de forma eficiente os passageiros e suas bagagens; Evitar o extravio de bagagens, assim como a troca de bagagens entre passageiros; Agilizar a retirada das bagagens pelos passageiros na esteira; Permitir o transporte de animais de forma mais segura, com o uso de sensores de temperatura, bem como de oxigênio, de nível de água, entre outros; Diminuir a poluição sonora; Notificar passageiros de forma individual, no que concerne a avisos de atrasos nos voos, abertura de embarques, mudanças nos portões de embarque, últimas chamadas para embarque de passageiros retardatários; E, por último, realizar os créditos correspondentes aos voos dos passageiros, em seus planos de milhas, de forma automática, no momento do check-in;

3.5 Estudo de Caso 2: Monitoramento de pacientes

3.5.1 Definição do Problema

Muitos são os desafios evidenciados pela área de saúde, visto que se trata de um ambiente onde os profissionais lidam com uma questão delicada que é a vida dos pacientes e com os riscos envolvidos em seus tratamentos médicos, desde exames de rotina a procedimentos cirúrgicos de mais alta gravidade. Em consequência de razões como esta, os avanços tecnológicos acabam ocorrendo em um ritmo mais lento que em outras áreas, já que se exige mais cautela ao lidar com o estado de saúde dos pacientes, que muitas vezes já encontram-se debilitados, física e emocionalmente.

Além disso, os entraves ao desbravamento deste meio não se limitam ao campo de atuação da atividade estritamente médica. Entram em cena questões éticas que precisam ser resolvidas antes mesmo de dar início a qualquer método de pesquisa científica que tenha participação direta dos pacientes. Nas universidades, tem-se os comitês de ética para avaliar se o trabalho respeita as regras preconizadas pelos regulamentos de ética. Desse modo, o aluno precisa submeter seu trabalho a este comitê para conseguir aprovação para avançar em suas pesquisas. Uma vez autorizado pelo comitê da academia, o pesquisador irá precisar também da autorização do responsável pelo hospital ou unidade de tratamento hospitalar em que ocorrerá a pesquisa. Todas estas questões devem ser consideradas na etapa de planejamento do método de pesquisa, para que seja viabilizada a aplicação de seu experimento no ambiente real.

De modo mais específico, a motivação para realizar este estudo de caso veio de problemas como a predominância da reatividade, em detrimento da proatividade, no ambiente em que o paciente está sendo tratado. O que significa, por exemplo, que, em muitos casos, medidas são tomadas pela equipe médica somente depois que o paciente já teve o seu quadro piorado de alguma forma.

Fatos como este são bastante problemáticos, principalmente porque, muitas vezes, em virtude do tempo decorrido entre o momento em que o paciente teve complicações em seu estado de saúde e o momento em que este fato foi detectado pelos profissionais de saúde responsáveis por agir nestas circunstâncias, que aqui estamos chamando de

Detection Anomaly Interval (DAI), é grande o suficiente para que se tenham consequências indesejáveis, podendo provocar sequelas nos pacientes ou até mesmo leva-lo à óbito, no pior dos casos.

Um dos fatores que contribuem para que o DAI tenha uma forte tendência em apresentar um alto valor é o fato de que a inspeção do estado de saúde dos pacientes ainda é comumente realizada de modo presencial. Desta forma, o profissional da saúde precisa se deslocar até o local em que se encontra o paciente e inspecioná-lo, coletando seus dados, através da medição de seu pulso, temperatura, dentre outros dados vitais.

3.5.2 Solução Proposta

A solução planejada neste experimento consiste no levantamento de alternativas para tratar de problemas da ordem dos que foram mencionados anteriormente, como o grande lapso de tempo que se verifica no processo de detecção de anomalias. Neste contexto, anomalias devem ser interpretadas como qualquer anormalidade perceptível no estado de saúde do paciente, da qual decorra a necessidade de intervenção pelo profissional da saúde responsável pelos seus cuidados, no instante em que elas ocorrem.

3.5.3 Objetivos

De modo mais geral, deseja-se tornar o ambiente de atendimento hospitalar mais proativo. Também pretende-se aumentar a capacidade de interação entre as coisas do ambiente, que serão apresentadas na subseção 4.2.1. Com relação aos objetivos específicos, temos, por exemplo, a pretensão de tentar diminuir o intervalo de detecção de anomalias através da atuação dos agentes de software.

4. Resultados

4.1 AirportTrack&Trace System

O primeiro estudo de caso resultou no desenvolvimento de um sistema multi-agentes que consistiu em uma simulação das interações das coisas no aeroporto. Para isto, foram modelados e implementados agentes de software reativos, os quais serão relacionados a seguir.

4.1.1 As Coisas no AirportTrack&Trace System

Foram modeladas as seguintes coisas no AirportTrack&Trace System: 1- Passageiros com cartões de embarque eletrônicos; 2- Bagagens com identificadores; 3- Animais com identificadores; 4- Voos; 5- Portões de embarque; 6- Esteiras rolantes.

4.1.2 Os Agentes no AirportTrack&Trace System

Foram modelados os seguintes agentes de software no sistema:

Tabela 1: Agentes modelados no AirportTrack&Trace System com seu tipo de comportamento baseado no modelo de comportamento do JADE

Identificador do Agente	Descrição	Behaviour
AnimalEnvironmentReportAgent	Agente de transmissão de informações de sensoriamento do ambiente onde animais são transportados nos voos	*TickerBehaviour*
UpdateGateAgent	Agente de notificação de mudança no portão de embarque	*TickerBehaviour*
CallForPassengerAgent	Agente de chamadas de embarque para passageiros retardatários	*TickerBehaviour*
CreditMilesAgent	Agente de crédito automático de milhas no programa de milhas dos passageiros após check-in	*OneShotBehaviour*
BaggageClaimAgent	Agente de notificação de proprietários de bagagens	*OneShotBehaviour*
NotifyBeginBoardingAgent	Agente de notificação de início de embarque	*TickerBehaviour*
NotifyDelayInFlightAgent	Agente de notificação de atrasos no voo	*OneShotBehaviour*
BaggageAgent	Agente de alerta de desvio de rota da bagagem	*TickerBehaviour*
AnimalEnvironmentSensorAgent	Agente de sensoriamento do ambiente onde estão sendo transportados os animais	*TickerBehaviour*

Para construção do AirportTrack&Trace System, foi utilizada a linguagem de programação Java. A arquitetura do sistema foi projetada de modo que este fosse composto por módulos com responsabilidades bem definidas. Os agentes foram modelados no sistema a fim de compor um módulo específico, que conteve apenas agentes reativos e foi implementado utilizando-se a versão 4.3.0 da ferramenta JADE (*Java Agent DEvelopment Framework*). Jade é um framework para desenvolvimento de aplicações de agentes em Java, que simplifica a implementação de sistemas multi-agentes através de um middleware que está em conformidade com o padrão FIPA (*Foundation For Intelligent, Physical Agents*) e utiliza um conjunto de ferramentas gráficas que suportam as fases de depuração e implantação.

A API JADE disponibiliza dois tipos de classes de comportamentos que podem ser estendidos para os agentes: Primitivos e Compostos. O comportamento dos agentes do AirportTrack&Trace System foi implementando a partir deste modelo de comportamentos, mais especificamente da classe de comportamentos primitivos (*Primitive Behaviours*). Cada agente do AirportTrack&Trace System é uma extensão da classe *Agent* e possui um comportamento que corresponde à extensão da classe *Behaviour* do JADE. Todo agente do sistema tem seu comportamento definido através do seu método *setup*, onde seu comportamento é configurado, através do método *addBehaviour*.

O AirportTrack&Trace System possui agentes reativos com dois tipos de comportamentos: *TickerBehaviour* e *OneShotBehaviour*. Como comportamentos do tipo *TickerBehaviour* executam suas ações de forma cíclica, escolhemos os agentes do nosso cenário que precisam realizar atividades de monitoramento contínuo para implementar este comportamento, como é o caso dos agentes: 1- *CallForPassengerAgent*: Agentes que precisam monitorar continuamente se todos os passageiros que fizeram check-in em determinado voo já embarcaram, a fim de notificar retardatários; 2- *NotifyBeginBoardingAgent*: Agentes que checam, de tempos em tempos, o início de embarques; 3- *UpdateGateAgent*: Agentes que verificam constantemente a ocorrência de mudanças no portal de embarque;

Outros agentes, por sua vez, são encarregados de executar tarefas que não são realizadas em um intervalo de tempo predefinido, ou seja, que ocorrem sob demanda, em

resposta a um determinado evento. É o caso do agente *CreditMilesAgent*, que credita as milhas no plano de milhas do passageiro no momento em que este realiza o check-in;

4.1.3 Discussão

As simulações realizadas no sistema mostraram que muitos dos problemas apresentados neste cenário poderiam ser minimizados adotando-se estratégias simples, como o uso de um sistema que abrangesse funcionalidades como as do AirportTrack&Trace System: Identificação de bagagens dos passageiros com etiquetas RFID, para serem monitoradas e rastreadas a fim de evitar extravios; Uso de identificação por radiofrequência também para os animais que estão sendo transportados, para que informações sobre as suas condições físicas e de seu ambiente possam ser coletadas, transmitidas e monitoradas para aumentar a segurança dos animais durante os voos e, consequentemente, a confiança dos seus donos; Criação de um cartão de embarque eletrônico para passageiros (*BoardingPass*), para facilitar o processo de comunicação das empresas aéreas com os passageiros dos seus voos. Esta troca de informações em um canal direto passageiro-companhia aérea poderia diminuir a poluição sonora decorrente das chamadas para embarque de passageiros retardatários e demais avisos. Também evitaria que os passageiros verificassem continuamente a ocorrência de alterações em seus horários de voos, mudanças em seus portões de embarque e outras notificações importantes.

4.2 MonitoringPatient System

4.2.1 As Coisas no MonitoringPatient System

Os seguintes elementos foram modelados no sistema para representar o conceito das coisas envolvidas na unidade de tratamento hospitalar onde o paciente irá receber o atendimento médico: Pacientes; Profissionais de saúde; Ambiente do paciente (UTI, CTI, Sala de cirurgia, apartamento). Identificadas no sistema de forma única, as coisas serão capazes de comunicarem-se entre si para aumentar a inteligência do ambiente.

4.2.2 Os Agentes no MonitoringPatient System

Foram modelados os seguintes agentes de software no sistema:

Tabela 2: Agentes modelados no MonitoringPatient System com seu tipo de comportamento baseado no modelo de comportamentos do JADE

Identificador do Agente	Descrição	Behaviour
VerifySensorDataAgent	Procura anomalias nos dados sensoreados	*TickerBehaviour*
SendSMSAgent	Notifica anomalias, através do envio de mensagens	*OneShotBehaviour*

A construção do sistema MonitoringPatient System também foi feita a partir da linguagem Java. O sistema também dispunha de um módulo de agentes que foi implementado utilizando o framework JADE. Este módulo é composto pelos agentes reativos: VerifySensorDataAgent e SendSMSAgent.

O agente VerifySensorDataAgent, conforme seu identificador sugere, possui a tarefa de monitorar os dados vitais que estão sendo sensoreados. Para que esta solução funcione, é preciso que se utilize uma tecnologia muito comum no âmbito da IoT, que são os microcontroladores. A ideia é que que sejam adicionados ao microcontrolador, sensores de diferentes tipos, como sensor de batimento cardíaco e temperatura, por exemplo, formando, assim, um dispositivo IoT capaz de coletar os dados paciente automaticamente (Figura 1-a). Uma vez coletados, os dados são armazenados em uma

base de dados, de modo que o agente *VerifySensorDataAgent* tenha acesso e consiga monitorá-los para detectar possíveis anomalias. A figura 2 mostra um exemplo de atuação deste agente, onde ele detecta um valor anormal na temperatura de um dado paciente.

Uma vez detectada uma anomalia, o agente *VerifySensorDataAgent* comunica-se com o agente *SendSMSAgent*, enviando uma mensagem que informa sobre a necessidade de alertar o profissional da saúde responsável por agir no tratamento da anomalia detectada. Deste modo, o agente *SendSMSAgent* envia o alerta para o responsável, através de e-mail ou SMS. A Figura 1-c mostra um exemplo de mensagem SMS enviada pelo agente *SendSMSAgent* ao detectar anormalidade na temperatura de um dado paciente.

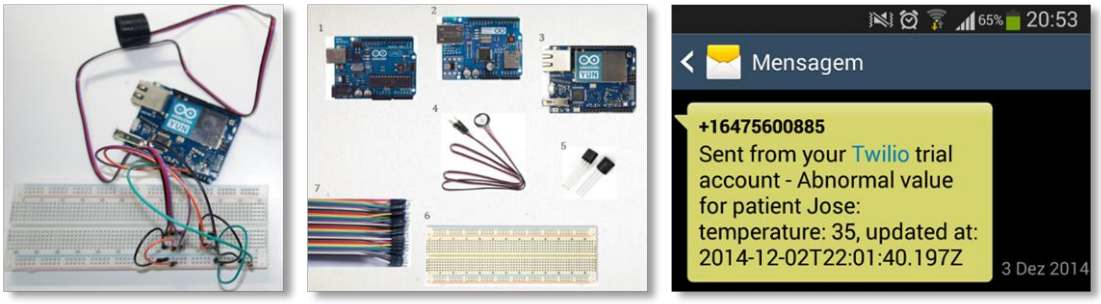


Figura 1-a. Protótipo IoT para o monitoramento de pacientes. Figura 1-b. Elementos do protótipo: 1-Arduino Uno R2; 2- Shield Ethernet compatível com Arduino; 3-Arduino Yún; 4-Sensor de pulso; 5-Sensores analógicos de temperatura; 6-Protoboard; 7-Fios do tipo macho-macho; Figura 1-c. Alerta enviado ao profissional de saúde pelo agente SendSMSAgent, via SMS.

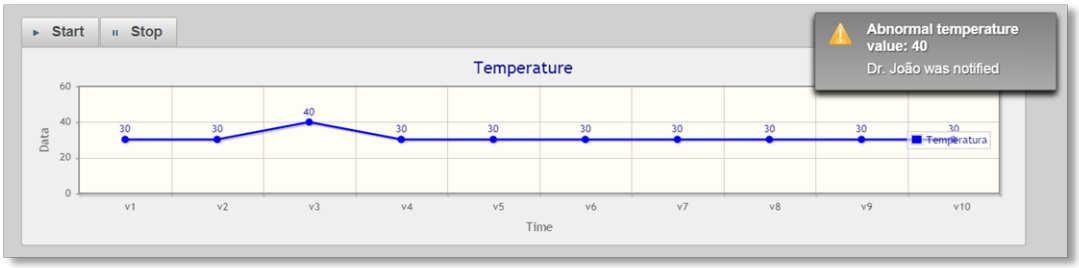


Figura 2. Gráfico usado para plotar os dados de temperatura do paciente. Pode ser acessado a partir do módulo de visualização do MonitoringPatient System. No canto superior direito, detecção de anomalia na temperatura do paciente pelo agente *VerifySensorDataAgent*.

4.2.3 Discussão

Através das simulações realizadas no sistema, pudemos observar que o DAI poderia ser diminuído com o uso do MonitoringPatient System. Esta diminuição traz como consequência o aumento da característica de proatividade do ambiente.

4.3 Framework

4.3.1 Features (F)

Após a realização dos estudos de caso, foi projetada a primeira versão do framework para construção de sistemas multi-agentes no âmbito da IoT. Nesta versão inicial, modelamos as seguintes *features*, para o núcleo das aplicações:

F1. Identificação automática das coisas do domínio através de tags RFID;

F2. Coleta de dados automática através do uso de protótipos de hardware compostos por microcontroladores Arduino e sensores de diversos tipos e com finalidades variadas;

F3. Armazenamento e recuperação dos dados sensoreados na base de dados;

F4. Visualização dos dados da base de dados no sistema, sob a forma de *line charts*;

F5. Inserção de conhecimento do especialista no domínio na base de dados;

F6. Definição de estratégias de negociação por recursos para os agentes cognitivos;

4.3.2 Arquitetura dos Sistemas Multi-agentes

O framework foi projetado de modo que a arquitetura dos sistemas construídos a partir dele tenha uma estrutura modular. Basicamente, o núcleo principal de cada SMA será composto por três módulos principais:

Módulo 1. Coleta de Dados: responsável por receber os dados coletados pelo protótipo, formado pelo microcontrolador e conjunto de sensores específicos da aplicação, e armazená-los na base de dados.

Módulo 2. Visualização: corresponde à parte do sistema multi-agentes que irá recuperar os dados sensoreados na base de dados e exibi-los, via interface gráfica. Os dados serão exibidos na forma de gráficos do tipo *line chart*, conforme ilustrado na figura 2.

Módulo 3. Atuação dos Agentes: irá tratar das tarefas específicas dos agentes. Pode compreender a atuação de agentes reativos e cognitivos.

No caso de agentes reativos, seu comportamento deverá ser inteiramente mapeado na aplicação, previamente. Desta forma, a reação do agente deve ser definida previamente para possibilitar a sua atuação. Tomando como exemplo o sistema MonitoringPatient, à cada anomalia detectada, precisaria ser definida na configuração inicial da aplicação, o comportamento que deveria ter o agente. A realização desta etapa de configuração precisa ser feita com o auxílio de um especialista no domínio. Os agentes cognitivos, por sua vez, poderão atuar em processos de negociação por recursos.

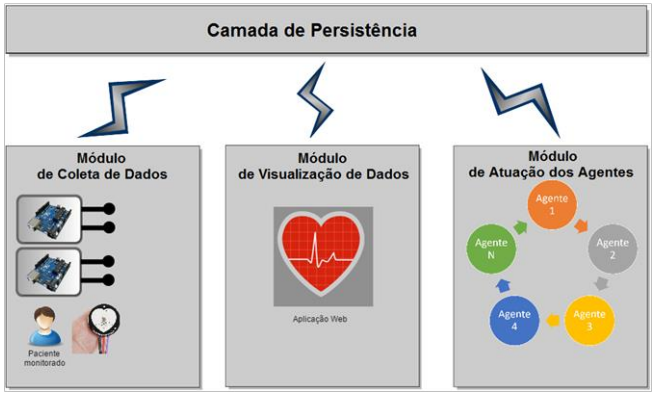


Figura 3. Visão geral da arquitetura dos SMAs construídos a partir do framework.

5 Conclusões e trabalhos futuros

Os estudos de caso mostraram que o DAI poderia ser diminuído com o uso do MonitoringPatient System, aumentando a proatividade do ambiente. Também concluiu-se que aumentou a interação *human-to-thing*, visto que os agentes, considerados coisas do domínio, foram capazes de alertar os profissionais da saúde sobre anomalias no estado de saúde dos pacientes. Além do aumento na interação *thing-to-thing*, já que os agentes

VerifySensorDataAgent e *SendSMSAgent*, ambos considerados como coisas do domínio, comunicaram-se efetivamente na realização de suas ações.

Consideram-se, como trabalhos futuros: implementação do framework projetado e aprofundamento nas investigações através do uso de técnicas de *data mining* nos dados coletados para extração de conhecimento útil, bem como de *machine learning*, para que os agentes tenham capacidade de fazer predições sobre o estado de saúde dos pacientes;

Também pretende-se implementar o comportamento de agentes cognitivos para atuarem na negociação por recursos. No caso de sistemas como o MonitoringPatient System, o agente poderia ser responsável por adotar estratégias de negociação para conseguir recursos hospitalares para um dado paciente. Esta funcionalidade poderia ser modelada de modo que, à cada paciente fosse associado um agente que seria responsável por representá-lo no processo de negociação. Este agente teria acesso às informações referentes aos sinais vitais que estão sendo coletados e, com isso, poderia usar estes dados para gerar conhecimento sobre o estado de saúde do paciente. De posse desse conhecimento, o agente seria capaz de atuar em benefício do paciente que ele representa e requerer acesso a um dado recurso, à medida que este se fizesse necessário.

Referências

Atzori, Luigi; Iera, Antonio; Morabito, Giacomo. (2010) "The Internet of Things: A survey", In: Journal Computer Networks: The International Journal of Computer and Telecommunications Networking, vol.54, pages 2787-2805

Bui, Nicla; Zorzi, Michele. (2011). "Health Care Applications: A Solution Based on The Internet of Things". In: ISABEL '11 Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologie.

Weiser, Mark. (1993). "Some Computer Science Issues in Ubiquitous Computing".

Kuniavsky, Mike. (2010). "Smart Things: Ubiquitous Computing User Experience Design".

Russell, Stuart; Norvig, Peter. (2013). "Inteligência Artificial". Elsevier.

Doukas, Charalampos. (2012). "Building Internet of Things with the Arduino".

Wooldridge, Michael. (2009). "An Introdcution to MultiAgent Systems". Wiley.

Padgham, Lin; Winikoff, Michael (2004)."Developing Intelligent Agent Systems". Wiley

McRoberts, Michael. (2011). "Arduino Básico". Novatec.

Finkenzeller, Klaus. (2003). "RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification". John Wiley & Sons, Inc., NY, USA, 2nd edition.

Nguyen, Stéphanie. (2009). "RFID for Track & Trace of Baggage in Airports. Business Aspects of the Internet of Things, Seminar of Advanced Topics", In: ETH Zurich, FS2009, Florian Michahelles.

Zhang, Ting; Ouyang, Yuanxin; He, Yang. (2008). "Traceable Air Baggage Handling System Based on RFID Tags in the Airport". In: Journal of Theoretical and Applied

http://arduino.cc/

http://jade.tilab.com/

A Robotic-agent Platform For Embedding Software Agents using Raspberry Pi and Arduino Boards

Nilson Mori Lazarin, Carlos Eduardo Pantoja

[1]CEFET/RJ – UnED Nova Friburgo – Av. Gov. Roberto da Silveira, 1900 – Nova Friburgo – RJ – Brasil

pantoja@cefet-rj.br, nlazarin@cefet-rj.br

Abstract. *This paper presents a robotic-agent platform to embed software agents into hardware devices. The platform consists in embed Jason framework in Raspberry Pi, allowing directly control of its pins, in order to control hardware devices, and Arduino to control sensors/actuators. So, it is necessary to prepare both hardware and software, and establish a communication between them. For this, it was developed the Javino library, which is a communication protocol for exchange messages between Java and Arduino using a serial port. It is also presented a three step methodology for supporting the robotic-agent development. An example using the proposed platform and methodology is presented. It was chosen a vehicle chassis, as the hardware robot, that is able to move and deviate from obstacles.*

1. Introduction

Intelligent Agents are autonomous entities capable of reasoning and are situated into an environment which can be physical or virtual [Wooldridge, 2000]. Nowadays, robotics is one field application for intelligent agents where a robotic agent can have actuators and sensors to interact with the physical environment. Besides, it is necessary a reasoning system embed into hardware to provide some intelligence to the robot. Multi-agent systems (MAS) are software or hardware systems that deals with subjects such as cooperation, distributed control, communication and fault-tolerance, in real-time situations. So, to implement a robotic agent microcontrollers are necessary, in order to control the actuators and sensors; an agent reasoning system using a program language; and a bridge between the hardware and the software layers. .

There are several microcontrollers like the ATMEGA328, part of Arduino board [Gertz; Justo, 2012], which is a widespread board for small automation projects; and the Raspberry Pi board that is a tiny computer with high processing capability [Upton; Halfacree, 2012]. However, the Arduino processing is very slow for an embed agent reasoning; and the Raspberry Pi although has a higher processing and memory power, it does not have an analogic interface, avoiding some sensors to be used. For the development of reasoning systems, there are several agent-oriented program languages like Jade [Bellifemine et al., 2013], a Java-based agent-oriented programming language, and JaCaMo [Boissier et al., 2011], which integrates three technologies for a Multi-Agent System (MAS) development: the Jason for agent reasoning programming; the CArtAgO for environments' artifacts programming; and the Moise+ for organizational programming.

Some projects try to integrate and embed a robotic reasoning into hardware such

as [Barros et al., 2014] that is an automated grounded vehicle, which uses the ATMEGA328 microcontroller to program the hardware basic functions; a java library for serial communication between the hardware and the simulated environment programmed in Java; and Jason framework for the agent programming. In [Calce et al., 2013] it is proposed an autonomous aquatic robot, which uses Arduino together with BeagleBoard and can move point-to-point deviating from obstacles.

The objective of this paper is to propose a platform for robotic agents, which uses the Raspberry Pi and Arduino together to provide the hardware controls, and uses Jason framework for the agent reasoning. Besides, an improvement of [Barros et al., 2014] platform is also presented. For the platform construction it will be developed: an action-ready hardware file for the 4WD Robot Chassis vehicle; The Javino, a library for serial communication between the Arduino and the Java Environment; a direct link between Jason and the Raspberry Pi using Pi4J library (available at http://pi4j.com/); and a simple example using the robotic-agent programmed with Jason controlling the 4WD Robot Chassis vehicle.

This paper is structured as follows: section 2 presents the methodology for supporting the robotic agent programming; in section 3, a simple example using the 4WD Robot Chassis vehicle and Jason framework is presented; section 4 discusses some related work; section 5 concludes the work; and finally the references used in this paper are shown.

2. The Robotic-Agent Platform

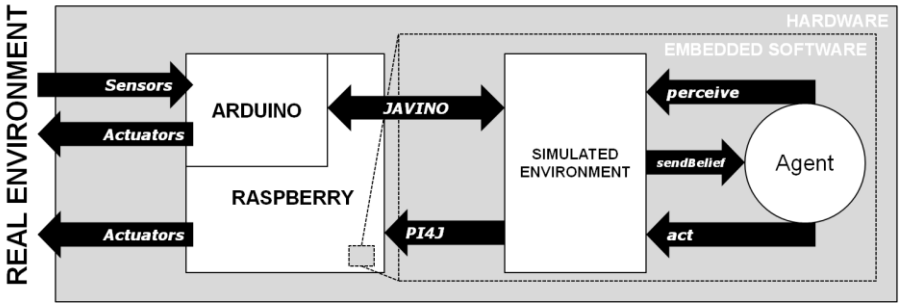
In this section, it is presented the proposed robotic-agent platform and a methodology to support the robotic-agent programming. The robotic-agent platform consists in an embedded software (agent) into a hardware platform (robot). The hardware platform is composed of the Arduino and Raspberry Pi boards, where Arduino is connected on the top of Raspberry Pi (using a USB port) to provide analogic hardwares to be used once Raspberry does not provide analogic pins. Therefore, it is possible to connect actuators devices using both boards, but is only allowed to use sensors connected with Arduino. One of the advantages of using Raspberry Pi is that it is possible to connect up to 127 Arduino boards (the USB device limit) in a single board.

The software agent is programmed using Jason framework and it is embedded into Raspberry because of its processing power and storage capacity. The platform uses the Javino to provide bidirectional serial communication between the hardware (Arduino) and the software agent; and the PI4J library for a direct control over Raspberry's digital pins. So, the agent is able to switch the Raspberry's pins values without any microcontroller intervention (the Raspberry uses the Python programming language to control digital pins), and senses the real environment or act upon it using Javino which transfers data perceived or sends a command for some action to be executed in real world. The platform is initialized once Raspberry starts. The proposed platform can be seen in figure 1.

The Raspberry can be used to manage both actuators and sensors, however this is not the main objective of the board since the GPIO are delicate and do not accept analogic inputs on its project. The Arduino's pins are more resistant then GPIO, besides it has available analogic pins. Furthermore, it is possible to communicate with Raspberry from an USB port. Therefore, in this platform, the Arduino manages sensors and actuators,

while Raspberry hosts the reasoning and control only some actuators with GPIO.

Figure 1. The proposed robotic-agent platform.



2.1. The Communication Protocol Using Javino

Here it is introduced the Javino which is a double-side library protocol for exchanging messages between an Arduino and Java program using a serial port. When a software agent needs to act in the environment, it is necessary to transmit its command to the microcontroller (Arduino ATMEGA328) where the actuators are connected. Since the agent cannot perform an action directly to the Arduino, because it is not possible to embed an agent software into a limited microcontroller (Arduino storage is due to 32Kb), it is necessary to provide a bridge between agent's external actions and microcontroller's functions. In the same way, when the agent needs to sense the environments, the data perceived from sensors has to be sent to the agent.

There are some libraries that use serial port to deal with one-side messages such as RxTx and IO library (both for Java). However, these libraries just provide message treatment for one platform side (environment) leaving the other side to the programmer. The Javino aims to fill this gap because it offers a double-side communication library based on the platform functioning: the Javino for Arduino and Javino for Java. They work together to provide a higher level of correctness in message exchange.

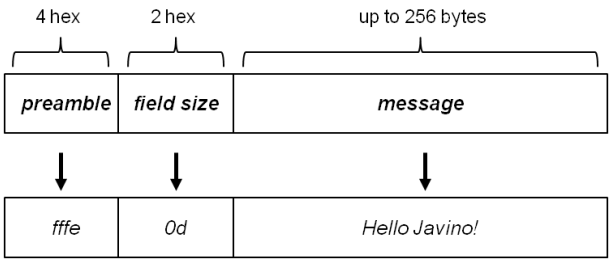


Figure 2. The message format.

For this, every message is composed of a preamble, a field size and the message content (figure 2). The preamble is a field composed of four hexadecimal characters that are used to identify the beginning of a message sent by an agent. The field size is composed of two hexadecimal characters that are used to calculate the message extension.

Finally, the last field is the message content up to 256 bytes. The preamble and the field size are used together to avoid errors in case of loss of information during the message transmission. For the sake of practice, Javino automatically mounts the message.

When a message is sent (either from agent to Arduino or vice versa), the Javino library starts to listen the serial port for arriving char-to-char messages. If there is any information arriving, the Javino stores this character analysing if it is part of the expected preamble. So, this process is repeated until the message has been completely received. Once the preamble is not confirmed, the Javino discards all information received until it finds a valid preamble. Otherwise, the Javino verifies the field size value to identify the message length. This process avoids error insertions and defines where a message starts and ends. Javino still has a pre-defined time-out to every character received, ending the actual process (if the time-out has been reached) and starts a new message listening.

After all done, the string message is mounted and returned. For the Arduino-side Javino, the message received will activate a hardware function or ask for data from sensors while for the environment-side Javino, the received message can be transformed into beliefs and sent to the software agent.

2.2. The Platform Methodology

In this section it is explained the methodology to support the robotic-agent programming. The methodology aims to guide the programmer between phases that will need some programming intervention, since the platform uses Arduino, Raspberry Pi and Jason. The methodology is composed by three steps that can be seen in figure 3.

The STEP 1 phase is composed by the selection of the hardwares (robot, sensors and actuators) to be connected in the Raspberry and/or Arduino boards. The selection consists in choose one pre-defined file for Arduino robots available with the methodology. Until now, there are two pre-defined robot chassis available: the Rover 5, a tank style vehicle; and the 4WD, a four-wheel drive vehicle. These pre-defined files guarantee all basic movements (forward, reward, left, right, turn, etc.) for those robot chassis. In the same way, the sensors and actuators selection consist in import a communication file where functions for connecting some devices (like GPS devices and compass) are defined. The methodology tries to provide a plug-and-play style, where the programmer do not need to interfere too much into the programming.

However, if the programmer desires to develop his own robot (or improve an existing one), it has to be programmed, in the Arduino board, all functions for the connected sensors and the actions that will be performed when the software agent executes an external action (both using the Javino library).

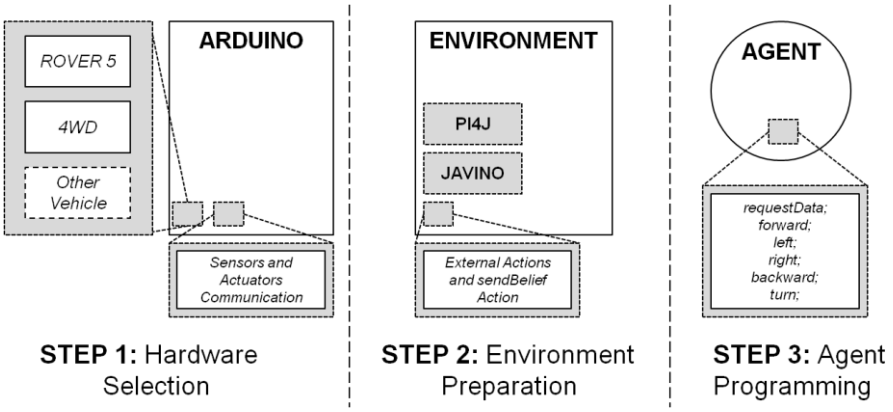


Figure 3. The robotic-agent programming methodology.

In STEP 2 it is necessary to prepare the agent's simulated environment. As the platform uses the Jason framework, it is used the basic Java environment where the agent's external actions (the actions that an agent wants to execute in real environment through the hardware layer) are programmed. So, when an agent wants to activate the actuators to move left for example, it is necessary to program the message (it is used "left" for the robot pre-defined files) that will be sent to the hardware. If the agent wants to get data from sensors, it is necessary to request it using an external action, where the hardware will return the data via Javino.

In order to have the communication between the software agent and the hardware it is necessary to import two libraries into Jason's simulated environment: i) The Javino library for Java which provides communication between Arduino and the environment, and; ii) The Pi4J library which provides functions for directly controlling the Raspberry pins (without any intervention in the microcontroller programming).

At last, the STEP 3 is the agent programming using AgentSpeak. In this phase it is just necessary to program normal agents using Jason framework. The agent's plan should have external actions to perform actions into the simulated environment. The methodology do not provide how to connect electronic devices for neither the robots chassis nor sensors and actuators. For more information about how to use the devices and pre-defined files cited in this paper look for http://sourceforge.net/projects/javino/.

3. Controling a 4WD vehicle with Jason

In this section it is presented a simple example using both proposed platform and methodology to implement a vehicle robotic-agent. The vehicle is able to move forward until it finds an obstacle when it will change its directions turning to right, left or reward, depending if there is another obstacle or not. For this, it was chosen the 4WD vehicle and an ultrasonic distance sensor. The 4WD chassis is equipped with four 7.2V motors encoders. By means of exemplification and to fully follow methodology, was chosen to connect the vehicle and the sensor at Arduino board using the Raspberry only for embed the software agent.

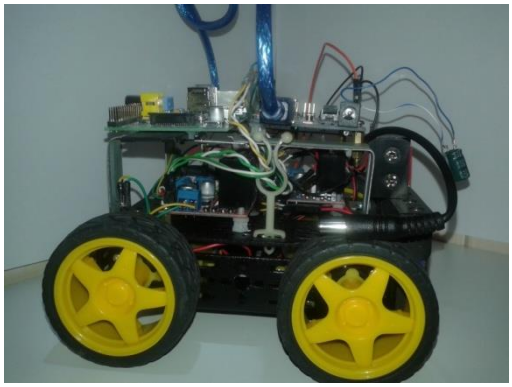


Figure 4. The vehicle robotic-agent chassis.

Following the methodology, in STEP 1 was selected the pre-defined file for the 4WD vehicle and the communication file for the ultrasonic distance sensor (both available at http://sourceforge.net/projects/javino/.). The pre-defined files are ready to use and have the Javino library setted. The pre-defined 4WD file provides movements functions (move, reverse, left, and right).

After that, the environment has to be prepared for the agent's external actions. First of all, it is mandatory to import Javino library into Jason's environment. The Pi4J library is not necessary in this example because it was chosen to connect the hardwares in Arduino board. All movements functions present in 4WD file must be programmed (if the programmer desires to use all possible movements). It was chosen to use the request-send method to update the agent's belief base: the agent requests to update its beliefs using an external action. Finally, the agent is programmed using Jason and AgentSpeak. The environment programming and the agent code can be seen in figure 5.

```
if (action.getFunctor().equals("front")) {
    logger.info(agName + ": sent front to Arduino.");
    javino.sendmsg("front");
}

if (action.getFunctor().equals("refresh")) {
    logger.info(agName + ": refreshing sensors.");
    javino.sendmsg("refresh");
    if (javino.availablemsg()) {
        addPercept(Literal.parseLiteral(javino.getmsg()));
    }
}

if (action.getFunctor().equals("turnLeft")) {
    javino.sendmsg("turnLeft");
    logger.info(agName + ": sent TurnLeft to Arduino.");
}
```

```
!start.

+!start: connected <-
    !move.

-!start: not connected <-
    .print("Waiting for hardware response...").

+!move : not obstacle <-
    .print("Go ahead.");
    front;
    !refreshSensor;
    !move.

+!move : obstacle <-
    .print("Obstacle ahead... turning left!");
    turnLeft;
    !move.

+!refreshSensor <-
    refresh.
```

Figure 5. The environment and agent implementation.

Now, the Raspberry Pi has to be configured to start with the embedded software (the project's jar file). It was realized two basic experiments: using a room without obstacles; and a room with three obstacles. The examples could show that the platform

works as expected, providing a reasoning for a hardware trying to reduce the programmer interventions. Some problems could be identified such as the agent executing time to send messages while it was still waiting data from sensors. The serial buffer could loss the information if the agent sends another serial command to the hardware. For now, this problem can be solved by both programming: *wait* action in Jason or *Thread.Sleep* in Java.

4. Related works

In this section it is discussed the proposed platform and methodology in comparison with the platform for grounded-vehicles [Barros et al., 2014] and the aquatic robotic using and BeagleBoard [Calce et al., 2013]. The platform for grounded-vehicles proposes an integration between Jason framework and the Arduino that uses RxTx library for communication between Arduino board and Jason's environment. Although the platform can operate a ground-vehicle and be adaptable for any kind of vehicles, it uses transmitters and receivers for each side (Jason and Arduino) that could cause data loss, interferences and depends on an external processing station (e.g. computer). The proposed platform truly embed Jason into a hardware without using external devices or transmitters/receivers for communication since it uses Raspberry, which provides an operational system that can be directly connected to an Arduino board.

The methodology for the grounded-vehicle platform consists in four programming steps: the hardware selection; the firmware programming (for the sensors/actuator actions and transmitters/receivers actions); the environment preparation (using RxTx and Jason's external actions); and the agent programming. The proposed methodology reduces the programmer interference into the code, because it offers pre-defined files for some hardwares, that can be changed based on the hardware selection; it uses Javino which provides a ready-to-use communication protocol between Java and Arduino; and can use the Pi4J library, that directly controls the Raspberry digital pins (decreasing the methodology in one step). Basically, the programmer only has to worry with the agent and the simulated environment.

In [Calce et al., 2013], an aquatic robot (boat) using Arduino BeagleBoard is presented, where a robot can move from one point to another deviating from obstacles. The robot platform consists in connecting via USB port Arduino and BeagleBoard, which holds a standard robot middleware called ROS (for movements). The proposed robotic-platform uses Raspberry Pi, a lower price board with similar specifications. It also provides pre-defined files for Arduino (such as ROS), which communicates with a message protocol library that can be used to exchange messages between hardware and software. The aquatic robot does not use agent-reasoning, being only an action-reaction robot while the proposed methodology uses Jason framework.

5. Conclusion

This paper presented a robotic-agent platform that uses Arduino and Raspberry to automate hardware functions and Jason to provide intelligent reasoning. Besides, it also presented the Javino library which is a communication protocol to exchange messages between Java and Arduino using serial port. A simple example using 4WD chassis was implemented to evaluate both platform and methodology.

For future works it will be proposed an architecture and several internal actions

for Jason, in order to provide to agents directly control vehicles movements. It will reduce the programmer interference into code, since he will not need to program the simulated environment anymore (for vehicles movements). It is also possible to directly update the agent's belief base without any interference into environment. The platform will be extended to guarantee communication between two or more robotic-agents. Furthermore, more pre-defined files for other kind of vehicles and sensors/actuators will be developed too.

References

Barros R. S., Heringer V. H., Pantoja C. E., Lazarin N. M., and Moraes L. M. (2014) "An Agent-oriented Ground Vehicles Automation Using Jason Framework" In: Proceedings of 6th International Conference on Agents and Artificial Intelligence: volume 1, ICAART'14, Angers, France.

Bellifemine, F., Caire, G., and Greenwood, D. (2007). "Developing multi-agent systems with JADE". Wiley series in agent technology.

Boissier, O., Bordini, R. H., Hubner, J. F., Ricci, A. e Santi, A. (2011) "Multi-agent oriented programming with jacamo" Science of Computer Programming.

Bordini, R. H., Hubner, J. F. e Wooldridge, W. (2007) "Programming Multi-Agent Systems in AgentSpeak using Jason" Jonh Wiley and Sons, London.

Calce A., Forooshani P. M., Speers A., Watters K. ,Young T., and and Jenkin M. (2013) "Autonomous Aquatic Agents" In: Proceedings of 5th International Conference on Agents and Artificial Intelligence: volume 1, ICAART'13, Barcelona.

Gertz E. and Justo P. D. (2012). Environmental monitoring with Arduino. EUA: Maker Press, 2012.

Upton, E. and Halfacree, G. (2012). "Raspberry pi user guide". United Kingdom: John Wiley & Sons Ltd,.

Wooldridge, M. (2000). "Reasoning about rational agents". Intelligent robotics and autonomous agents, MIT Press.

LOCUS: An environment description language for JASON

**Ramon Fraga Pereira, Maurício Cecílio Magnaguagno,
Felipe Meneguzzi, Anibal Sólon Heinsfeld**

[1]School of Computer Science (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

{ramon.pereira, mauricio.magnaguagno, anibal.heinsfeld}@acad.pucrs.br

felipe.meneguzzi@pucrs.br

Abstract. JASON *is an* AGENTSPEAK *interpreter for multi-agent system development, in which agents are described in the* AGENTSPEAK *language. Therefore, we only have to describe the agent behavior, but the environment does not follow this style, it requires a Java description of how the actions and perceptions operate. This choice of implementation guarantees that even complex environments can be created for* JASON*, but it requires knowledge about both Java and* JASON*'s Application Programming Interface (API). In this paper we aim to fill the gap between the languages with an* AGENTSPEAK*-like description of the environment. To overcome this gap we propose* LOCUS*, a source-to-source compiler which generates a Java description of the environment for* JASON*, providing the user with an easier starting point to create complex environments with a consistent description for both the agents and the environment. The output of* LOCUS *can be further modified if required, not limiting the user to the features already provided by* LOCUS.

1. Introduction

A multi-agent system consists of a population of autonomous computational entities situated in a shared structured environment. From this definition, we emphasize the importance of the environment, agents are not isolated entities since they share the same space. In several situations we want to simulate agents that perceive, reason, and act to achieve their goals within an environment. To describe agent behavior concisely a specific paradigm is used, Agent Oriented Programming (AOP) [Shoham 1993]. This behavior must match the restrictions of the environment to enable an agent to act successfully while pursuing its interests. Therefore, it is extremely important to model the environment correctly in multi-agent systems. However, most work on AOP, focuses on the formal description of the agents, rather than the environment. Some frameworks opt to use an AOP language to better describe the agents, while the environment remains described in an imperative language. Many researchers neglected to integrate the environment as a primary abstraction in models and tools for multi-agent systems, and much research is concerned exclusively with agents.

In this paper, we describe a declarative approach to create virtual environments in an AGENTSPEAK-like language. We target JASON, a platform for the development of multi-agent systems, due to the familiarity with the platform and current use of AGENTSPEAK to describe agent behavior. Currently, the environment specification in JASON must

be encoded in Java, which mix levels of abstraction used between agent and environment description. In this way, we propose LOCUS[1], a source-to-source compiler that converts an AGENTSPEAK-like description of the environment into a Java version with the corresponding bindings to the JASON API[2]. Although there is no gain in expressivity of the environment behavior, our aim is to allow designers to write simpler code that is easier to read. Without the common worries of API usage, the user is able to focus on what matters the most, i.e. the correct operation of the desired environment. Thus, our main contribution is an approach to describe the environment without mixing abstractions while maintaining the possibility to do so if required. We demonstrate the applicability of our approach through a subset of the standard JASON application examples. Our proposed environment description is concise and yields the Java description of the environment based on the JASON API, when the description is correct, and warns the developer otherwise.

This paper is organized as follows. Section 2 reviews the background on agents and environments, AGENTSPEAK, and JASON. Section 3 presents LOCUS and some practical examples of its usage. In Section 4 we address related work on environment description for multi-agent systems. In Section 5 we conclude with final considerations and give directions for future work.

2. Background

This section introduces essential background on agents, and important concepts related to multi-agent system environments.

2.1. Agents and Environments

Agents act through actuators and perceive through sensors within an environment [Russell and Norvig 2009]. Actuators and sensors allow agents to interact with the environment, these interactions are represented by actions and perceptions of the agent [Weiss 2013]. More specifically, actions are executed through actuators and produce an output that affect the environment. Actions are fundamental because they represent the mechanism through which agents change the environment. Perceptions are given as input to agents through sensors from the environment. In general, the interactions with the environment describe the agent behavior, for example, given an input of perceptions the agent can reason to act within an environment in order to achieve a desired goal [Wooldridge 2009].

Multi-agent environments are typically modeled depending on a variety of properties, for example: the environment can be deterministic or non-deterministic; fully or partially observable; and interactions can happen instantly or with a time duration. These properties tend to impact on how these agents interact with the environment, and consequently on how agents have to reason in order to act in an environment. Agents commonly represent the environment internally as a belief base, i.e, what the agents believe to be true from the interactions with the environment and other agents. Conversely, the environment contains an internal representation of the result of interactions between agents as a state, holding the properties of every object.

[1]The word locus is latin for place.
[2]http://jason.sourceforge.net/wp/

2.2. AGENTSPEAK(L) and JASON

AGENTSPEAK(L) is a programming language based on logic programming for the Belief-Desire-Intention (BDI) agent architecture which provides support for events and actions [Rao 1996]. The BDI architecture is used to provide components that bring about a notion of "mental state" for agents. It is characterized by implementation of computational analogues of human beliefs, desires, and intentions. The current state of the agent, which is a model of itself, its environment, and other agents, can be viewed as its current belief state. The states which the agent wants to accomplish are desires, based on its external or internal motivations. The adoption of programs to satisfy such stimuli can be viewed as intentions. The behavior of the agent, i.e, its interaction with the environment is encoded by the programs written in AGENTSPEAK(L). Thus, an AGENTSPEAK agent is created by the specification of a set of base beliefs and a set of plans[3] [Bordini and Hübner 2005].

JASON is an AGENTSPEAK interpreter for multi-agent system development, in which agents are described in an extension of the AGENTSPEAK language, but the environment is described in Java [Bordini et al. 2007]. JASON infrastructure is developed in Java and allows the customisation of the agent behaviors and reasoning. It provides three main constructs for agent programming and reasoning: beliefs, goals and plans, each described in turn below.

2.2.1. Beliefs

In JASON, beliefs are represented as predicates and stored in a collection called belief base. Unlike classical logic, beliefs are not an absolute truth. Each agent has an individual belief base and each stored belief is only true for its agent. As the logic programming language Prolog, a belief is represented by an atom (a sequence of characters starting with a lower-case letter) or a structure. Structures are useful to represent complex beliefs composed by an atom followed by arguments, for example door(closed).

JASON handles negation in two ways: using 'not' and $\sim$ operators. The first will lead the interpreter to result *true* about some formula if it cannot be derived using beliefs and rules of the agent. The second is called *strong negation* and brings the notion that a belief is explicitly false.

A difference from PROLOG syntax is the belief annotations. Annotations provide meta-level information about beliefs, like the source of beliefs. The source is annotated automatically in each belief by JASON, but custom annotations can be created in order to provide useful information for agent reasoning, such as the moment that the agent adds a belief. However, annotation are meaningless to the JASON interpreter, so the programmer must develop an expected behaviour for the annotations. Actually, annotations could be represented as other beliefs, but JASON's syntax provides a better readability and linkage of information.

Another structure provided in JASON is the Rule. A rule is a logic formula that the agent uses its belief base to reason about its truth. Rules bring brevity to the code, since it is not necessary to always represent the formula that is entailed by the rule.

[3]Plans represent a sequence of actions that an agent is able to perform in a environment to achieve a goal.

```
1 triggeringEvent : context <- body.
```

Listing 1. The three plan parts in JASON.

2.2.2. Goals

In JASON, goals represent the properties of the states of an agent's environment that the agent wishes to bring about. JASON has two types of goals: achievement goals and test goals. Achievement goals ares denoted by an exclamation mark and a predicate. If the goal is declarative, the agent believes that the predicate is not true and will act in order to modify the state of its environment to make it true, otherwise it is a procedural goal, the proposal of the goal is basically to group actions together, which is useful when these actions are often used. Test goals are normally used to retrieve information from belief base and are denoted by interrogation mark. When an agent adopts a goal, it executes a Plan, that is composed by a sequence of actions to change environment and agent states.

2.2.3. Plans

A plan in JASON consists of three parts: the triggering event, the context and the body, Listing 1. The triggering event can be an addition or a deletion of beliefs or goals, due to environment changes. So, the plan will be executed as a sequence of actions as a consequence of such event. The context is used to check the current circumstance in order to determine if the plan is expected to succeed. Due to different circumstances, a plan that better fits in the current state is chosen according to its context specified by agent designers. The body is composed by a sequence of actions, containing sub-goals that the agent must assume to proper handle the triggering event.

3. LOCUS Environment Description Language

LOCUS implements the counterpart of Agent Oriented Programming (AOP), targeting a concise and correct environment description. In the same way an agent requires a belief base to create an internal representation of the world, the environment requires a structure to hold its own current state. The difference is the point of view, the agent beliefs have a known or unknown degree of truth, due to the sensing capabilities of the agent who perceives part of the world at specific moments. The environment represents the entire state without errors, it contains the ground truth, and communicates parts of this truth as perceptions to the corresponding agents. The agent with the perceptions and reasoning may act, calling an environment action. The environment deals with the outcome of actions, those actions can only happen if the preconditions are satisfied, i.e, the agent class or the agent accuracy. With those concepts in mind, we create AGENTSPEAK-like constructs to represent them.

In order to be easier to describe we divided the environment in three main parts: an initial state (*init*), actions (*beforeActions, +action, afterActions*), and a stop call (*stop*), they can be seen in Listing 2. Although the environment parts can be defined in any order, we believe this order to be easier to understand. Firstly, we define an initial setup, in which the initial state of the environment is defined along with the perceptions they create.

```
1  init                                    <- body.

2  beforeActions                           <- body.
3  +action(name[, terms]) : context        <- body.
4  afterActions                            <- body.

5  stop                                    <- body.
```

Listing 2. The environment main parts.

```
1  +percept(agent|all, predicate[, terms]) : context.
2  -percept(agent|all, predicate[, terms]) : context.

3  +state(predicate[, terms]);
4  -state(predicate[, terms]);
5  -+state(predicate[, terms]);
```

Listing 3. Modifying perceptions and the environment state.

This creates a common starting point to the simulation. Secondly, we define how the actions affect the environment and the restrictions imposed on specific action executions. Additionally, in the second part, it is possible to define what happens before and after the execution of actions. This part is essential to keep environment and agents interacting, modifying the state and which perceptions happen during the simulation. Finally, the stop setup defines what happens at the end of the simulation. Although not explored so far we believe to be important to log data at the end of simulation.

The actions are the main part, as they may be called several times per execution. The *beforeActions* controls what happens before any action takes place, usually clearing a set of percepts for some or all agents. After that the action that was called by name by an agent takes place and any term given as an argument is forwarded. If the context is satisfied the body of the action is applied. We provide macros to test agent name and agent class, **agentName** and **agentClass** respectively, to make it easier to test agent attributes.

Inside each part we find their respective body, a set of commands to describe what happens with the environment. The commands in Listing 3 can be used in the body of the following statements: *init, actions, before and after actions, and stop*. We focus on commands about perceptions and state at the current stage of development. It is possible that perceptions are broadcast to all agents or are restricted to a specific agent or subset of agents. However, the context must be true to apply the desired perception, like a sensor working properly and in range to capture the data. In order to handle the state of the environment, it is possible to specify features of the environment as propositions to be added or deleted from the state. The only impact of this approach is the generic structure used to hold the state configuration, while handmade versions of the environment would use tailored structures, like a single boolean variable to store a door state. The symbol plus ($+$) represent addition, and the symbol minus ($-$) represent deletion of a feature. These symbols are used as a prefix of the statement **percept**. Moreover, to update an environment state, the symbol ($-+$) is declared as a prefix of the statement **state**. In Listing 3 we exemplify how to add, remove, and update perceptions and states in LOCUS.

```
1 +!locked(door)[source(paranoid)] : ∼locked(door) <- lock.
2 +!∼locked(door)[source(claustrophobe)] : locked(door) <- unlock.
```

Listing 4. Porter agent.

```
1 +locked(door) <-
2     .send(porter,achieve,∼locked(door)).
```

Listing 5. Claustrophobe agent.

```
1 +∼locked(door) <-
2     .send(porter,achieve,locked(door)).
```

Listing 6. Paranoid agent.

We also use the plus symbol to add actions to the environment, further investigation is required to see how useful it would be to remove actions using a $-action$ versus using a context.

3.1. Examples

In this section we use examples in JASON to show how our source-to-source compiler can be used. Those examples describe small environments, but handle with state control and agent perception. We make use of the room application, one of the JASON examples, and Bakery application, an example created specifically to show how our approach handle with different environments.

3.1.1. Room

This application consists of a room with a door, and contains the following agents: a porter, a claustrophobic and a paranoid. The only object in this room is the door that may be closed or not. The porter (Listing 4) is the only agent that modifies the door state, using lock and unlock actions, while all agents can perceive the current state of the door. When the claustrophobic agent (Listing 5) perceives the door closed the only option is to ask the porter to open the door, after all this agent does not like to be confined. The paranoid (Listing 6) does not share the same fear, instead this agent prefers to be safely closed inside the room. Once the paranoid perceives the door opened the only option is to ask the porter to close it. The porter only does as told.

The environment is extremely simple: it contains two different perceptions and two different internal actions about the same door. In the Prometheus diagram [Winikoff and Padgham 2004], in Figure 1, we see only one agent able to interact with the door, the porter. Yet, in the current version of JASON these actions have no test towards the agent class or name. This lack of test makes any agent eligible to close the door, which creates no problem for this small example, but shows how hard it can be to follow a specification when you need to code in two different styles. Our description in LOCUS (Listing 7) covers almost the entire Prometheus design, it only lacks a description of the message passing mechanism between agents, since message passing is done in JA-

```
1  init <-
2      +state(doorLocked);
3      +percept(all, locked, door).

4  beforeActions <-
5      -percept(all).

6  +action(lock) : agentClass(porter) <-
7      -+state(doorLocked).

8  +action(unlock) : agentClass(porter) <-
9      -+state(~doorLocked).

10 afterActions <-
11     +percept(all,  locked, door) : state( doorLocked);
12     +percept(all, ~locked, door) : state(~doorLocked).
```

Listing 7. LOCUS description for Room application

SON without environment restrictions. The generated Java code matches the original Java environment in JASON – it is able to replace that code without further modifications.

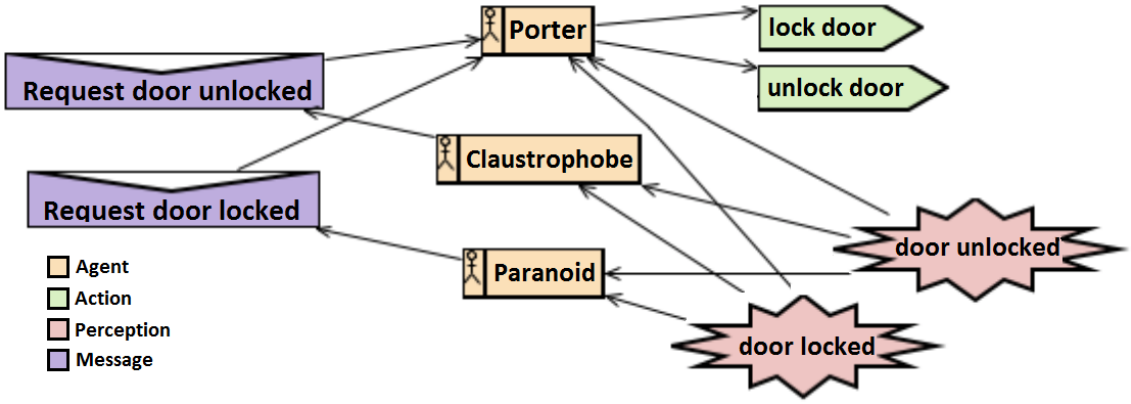


Figure 1. An overview of the room example.

3.1.2. Bakery

In our second example we use a bakery application, containing two types of agents: a boss and one or more bakers. The Prometheus diagram of this application is shown in Figure 2. Our goal with this example is to create and perceive the existence of items in the environment. The boss (Listing 8) perceives when there is one item missing from the shelf, and since the boss likes to maintain the shelf full, pins a note with the task to bake more of some item. The baker (Listing 9) perceives there is a new task, and starts baking the new item. When finished the task, the new item is put directly in the shelf. The boss can now perceive the task was done. We could continue and add an agent to sell and other to buy items randomly, and keep the system in a loop, but we believe this subset of the application already shows how useful our source-to-source compiler can be. Listing 10 contains the environment, with several perceptions targeting the boss

```
+~have(C) <- pinTask(C).
+have(C) <- .print("Done␣", C).
```

Listing 8. Boss agent.

```
+newTask(X) <- bake(X).
```

Listing 9. Baker agent.

```
init <-
    +state(~have(pie));
    +state(~have(cake));
    +state(~have(donut));
    +percept(boss, ~have, pie);
    +percept(boss, ~have, cake);
    +percept(boss, ~have, donut).

beforeActions <-
    -percept(all).

+action(pinTask, C) : agentName(boss) <-
    +percept(all, newTask, C).

+action(bake, C) : agentClass(baker) <-
    -+state(have, C).

afterActions <-
    +percept(boss, ~have, pie)   : state(~have(pie));
    +percept(boss, ~have, cake)  : state(~have(cake));
    +percept(boss, ~have, donut) : state(~have(donut));
    +percept(boss,  have, pie)   : state( have(pie));
    +percept(boss,  have, cake)  : state( have(cake));
    +percept(boss,  have, donut) : state( have(donut)).
```

Listing 10. LOCUS description for Bakery application

and explicitly declaring the existence or lack of each item. It is possible to observe the redundancy that appears in the **afterActions** block, in which we are testing both positive and negative cases. The lack of an ”else” construct is not a problem, but takes repeated conditions to take care of. The good side effect is to have perceptions with independent causes, but we hope to address the problem of code repetition as bigger applications reveals them.

4. Related Work

Regarding the environment description for multi-agent systems, we highlight two related work: ELMS (Environment Description Language for Multi-Agent Simulation) and CArtAgO (Common Artifact infrastructure for Agent Open environments).

CArtAgO provides the infrastructure to develop environments based on the A&A (Agent and Artifacts) meta-model [Ricci et al. 2010, Ricci et al. 2011]. This meta-model describes three basic abstractions. First, the agents that represent pro-active components

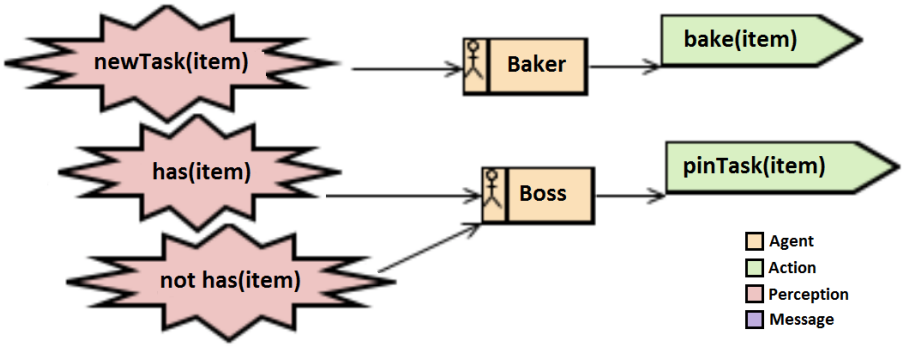


Figure 2. An overview of the bakery example.

of the system. Second, the artifacts, which represent passive components of the system and support agents' activities. Third, the workspaces, that provide a notion of topology for the environment. In CArtAgO, artifacts are not autonomous and can not follow their own course of action, serving as tools for agents. The programming of CArtAgO artifacts is made using a Java-based API. This API provides Java annotations and classes to make accessible specific methods to JASON, aiming at the interaction of agents with the artifacts. CArtAgO is part of the framework JaCaMo. JaCaMo [Boissier et al. 2013] combines the environment description of CArtAgO with JASON and Moise, an organisational model for multi-agent systems. This union of technologies provides a robust development environment, allowing cooperative and distributed agent behaviours. However, CArtAgO and Moise require the use of different languages and paradigms, which can difficult the development and the linkage of design concepts.

In order to describe the environment, [Okuyama et al. 2004] propose ELMS to use a markup language syntax, namely XML. Due to generality of XML, the authors defined a set of elements and attributes to describe the parts covered by the work. First, the structures to define agent bodies specify "physical" aspects of the agents, such as the properties that may be perceived by other agents and the actions that agents can perform in the environment. Second, the work provides a structure to define 2D or 3D grids, if the designer has chosen to have one. Attributes can be defined for each cell of the grid to be perceived by agents. Reaction of actions also can be defined, as a response to the actions of agents and changes to the attributes. Objects can be placed on the environment via the notion of resources. Resources, as grid cells, have their own attributes and reactions. Third, some elements are defined in order to specify the initialisation process and parametrizations of the environment. As noted by the authors, describing the environment with XML can be cumbersome and may require a graphical interface to ease the development process.

5. Conclusions

As JASON's API gets more complex the direct use of Java constructs requires a good understanding of the inner workings of Java, which takes time and is often not the goal of JASON users. New users of JASON are LOCUS target audience, being able to use simpler constructs without knowledge about JASON's internal structures. Two other advantages became clear during the development process, the first is the ability to include the environment description, now much shorter, in texts without having to explain Java code first or select slices of the original description. The second advantage is to be independent

from the internal structures used by JASON and its extensions, which may prevent updates in JASON's API from breaking legacy code (as is often the case even with minor JASON updates). There is a long way before LOCUS supersedes the direct Java descriptions used today. Some applications may require specific constructs available in Java (File input, optimized structures, random outcomes from the same action) and the limitations of our solution to recreate the behavior in the current state. Those constructs could be solved right away, but then our implementation would replicate the imperative paradigm of Java, telling how to solve the problem, instead of representing the environment.

We now focus on several constructs to be added, being implemented to match JASON's AGENTSPEAK behavior, adding unification and lists to describe more complex environments than what is currently possible. The goal is to use Java as a last resort, keeping the description free of an internal architecture. In the current version we see the current models and views as a problem to our goal. Models are tailored structures to hold the state, while views are graphical interfaces to show the state. Model and view usually appear together and we search for an abstraction to give as good results as them. The model provides speed with the correct structures while the interface provides a visualization method specifically for the application. Their only problem is the complexity to maintain both working correctly, since they vary for each application. We hope to address the view in the future and create some optimizations to our generic state structure to yield a tailored one. Today the environment does not consult the agent information, like beliefs and intentions, we believe this information must remain encapsulated. Maybe a specific application will require an extension to do so. Another discussion that we had is related to time-based events, happening at some point in time or with a duration, but we have not explored this enough to create an abstraction. The current set of commands is small, but shows it is possible to describe the environment without mixed levels of abstraction while opening the subject of an AGENTSPEAK-like language to describe environments. The LOCUS project[4] is available for download in the GitHub website. In this website we provide instructions on how to use the LOCUS description language to create environments for JASON.

References

[Boissier et al. 2013] Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent Oriented Programming with JaCaMo. *Sci. Comput. Program.*, 78(6):747–761.

[Bordini et al. 2007] Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons.

[Bordini and Hübner 2005] Bordini, R. H. and Hübner, J. F. (2005). BDI Agent Programming in AgentSpeak using Jason. In *Proceeding of 6th International Workshop on Computational Logic in Multi-agent Systems (CLIMA VI). Volume 3900 of LNCS*, pages 143–164. Springer.

[Okuyama et al. 2004] Okuyama, F. Y., Bordini, R. H., and da Rocha Costa, A. C. (2004). ELMS: An Environment Description Language for Multi-agent Simulation. In *Envi-*

[4] https://github.com/Maumagnaguagno/Locus

ronments for Multi-Agent Systems, First International Workshop, E4MAS 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers, pages 91–108.

[Rao 1996] Rao, A. S. (1996). AgentSpeak(L): BDI Agents Speak out in a Logical Computable Language. In *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-agent World : Agents Breaking Away: Agents Breaking Away*, MAAMAW '96, pages 42–55, Secaucus, NJ, USA. Springer-Verlag New York, Inc.

[Ricci et al. 2011] Ricci, A., Piunti, M., and Viroli, M. (2011). Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192.

[Ricci et al. 2010] Ricci, A., Viroli, M., and Piunti, M. (2010). Formalising the Environment in MAS Programming: A Formal Model for Artifact-based Environments. In *Proceedings of the 7th International Conference on Programming Multi-agent Systems*, ProMAS'09, pages 133–150, Berlin, Heidelberg. Springer-Verlag.

[Russell and Norvig 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[Shoham 1993] Shoham, Y. (1993). Agent-oriented Programming. *Artif. Intell.*, 60(1):51–92.

[Weiss 2013] Weiss, G., editor (2013). *Multiagent Systems*. MIT Press, Cambridge, MA.

[Winikoff and Padgham 2004] Winikoff, M. and Padgham, L. (2004). *Developing Intelligent Agent Systems: A Practical Guide*. Halsted Press, New York, NY, USA.

[Wooldridge 2009] Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition.

Simulação baseada em agentes para atendimentos em saúde com eventos estocásticos

Nécio de Lima Veras[1], **Mariela I. Cortés**[2], **Gustavo A. Lima de Campos** [2]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
Rodovia CE-075, s/n - Aeroporto – Tianguá - Ceará

[2]Universidade Estadual do Ceará (UECE) - Fortaleza, CE - Brasil

necio.veras@ifce.edu.br, {mariela, gustavo}@larces.uece.br

Abstract. *Tentative of put in practice a planning to the offer of health services commonly has problems because of the occurrence of undesirable events. These events happen in a random way and they influence in a negative form the access and quality indicators of the planned services. This work presents a agent-based (computer) simulation to execute of schedules planned by an agent in a virtual environment for treatment, considering different scenarios to the evolution of treatments in an intermission of one planned year. The solution specifies three environments based on artifacts envolving in scenarios registering the statistic behavior of the health indicators.*

Resumo. *A execução de um planejamento para a oferta de serviços em saúde constantemente sofre desvios por conta da ocorrência de eventos indesejados. Esses eventos acontecem de maneira estocástica e influenciam negativamente indicadores de acesso e qualidade para os serviços planejados. Este artigo apresenta uma simulação (computacional) baseada em agentes para simular a execução de agendas planejadas por um agente em um ambiente virtual de atendimentos, considerando diferentes cenários para a evolução dos atendimentos em um intervalo de tempo compatível com um ano de planejamento. A solução especifica três ambientes baseados em artefatos que evoluem em cenários registrando o comportamento estatístico dos indicadores de saúde.*

1. Introdução

Simulações computacionais nas ciências sociais englobam o conhecimento oriundo de teorias e fenômenos sociais para incorporá-los a um modelo científico-social em computadores, normalmente definidos por meio de sociedades de agentes inteligentes. Com essa infraestrutura artificial é possível simular ações e interações de agentes, recriando comportamentos sociais para analisar os efeitos nos próprios agentes [David and Sichman 2013]. Neste sentido, nos últimos anos diversas aplicações para simulação baseada em agentes têm sido exploradas, tais como: sistemas econômicos, robóticos, ecológicos, tráficos e transportes, dentre outros [Davidsson et al. 2007].

Na saúde, segundo [Jacobson et al. 2013], nos últimos 40 anos as organizações de saúde tem enfrentado constantes pressões para prestarem melhores cuidados associado com custos crescentes, reembolsos mais baixos e exigências regulamentares adicionais. Para os autores, as simulações de eventos discretos tornaram-se uma aliada eficaz no

processo de tomada de decisão, pois a alocação ótima de recursos escassos pode melhorar problemas como o fluxo de pacientes, minimizar os custos de prestação de cuidados de saúde e aumentar a satisfação dos pacientes.

No Brasil, objetivando criar um parâmetro comparável para promover a melhoria do acesso e da qualidade da atenção à saúde, estabeleceu-se em 2011 o Programa Nacional de Melhoria do Acesso e da Qualidade da Atenção Básica (PMAQ-AB). O programa contém indicadores que podem determinar os valores dos parâmetros de qualidade para o acesso e para a oferta de serviços em saúde [BRASIL 2012].

O planejamento e o monitoramento de valores para os indicadores de saúde estabelecidos pelo PMAQ não é uma tarefa trivial, tendo em vista que são valorados diariamente conforme a oferta dos serviços de uma equipe multiprofissional. Em [Veras et al. 2014b] é apresentada uma arquitetura computacional inteligente para dar suporte ao planejamento de serviços orientado por metas de forma a auxiliar na avaliação do desempenho das equipes e na sua melhoria progressiva levando em conta recursos e demandas locais. O artigo especifica agentes capazes de planejar e monitorar os serviços para melhorar o desempenho das equipes conforme os indicadores do PMAQ. O problema do planejamento inteligente orientado por metas proposto no artigo é formalmente definido em [Veras et al. 2014a], porém em nenhum dos dois trabalhos foram realizadas simulações considerando eventos inspiradas no mundo real capazes de desviar a execução do planejamento dos atendimentos de maneira a coletar dados sobre o comportamento dos indicadores em cenários específicos.

Assim, o presente artigo objetiva apresentar uma simulação baseada em agentes capaz de retratar em um ambiente computacional a execução de serviços em saúde planejados por um agente que usa técnicas de otimização orientada por metas conforme os indicadores do PMAQ-AB. O trabalho incorpora na simulação eventos que acontecem estocasticamente e influenciam de forma negativa os indicadores. Visa-se com isso, retratar fatores do mundo real para coletar dados sobre o comportamento dos indicadores em relação ao planejamento e, com isso, analisar a eficiência da abordagem diante desses cenários. Para tanto, são considerados diferentes cenários que simulam a evolução dos atendimentos em um intervalo de tempo compatível com um ano de planejamento.

2. Trabalhos relacionados

A medicina tem recebido contribuições da Inteligência Artificial desde o final dos anos 60 [Patel et al. 2009] e no início dos anos 90 foi classificada como "adolescente" por [Shortliffe 1993]. Logo percebeu-se que paradigmas computacionais clássicos não seriam suficientes para representar sistemas com ambientes dotados de processos e interações complexos [Fox et al. 2003]. A tecnologia dos agentes inteligentes tem emergido nos últimos anos como um paradigma promissor que objetiva o desenvolvimento de sistemas complexos e a sua utilização no âmbito da medicina tem mostrado melhorias no desempenho destes sistemas em relação à interoperabilidade, escalabilidade e reconfigurabilidade, especialmente pelas características favoráveis para a criação de modelos de simulação [Isern et al. 2010] [Cardoso et al. 2014].

Em [Isern et al. 2010] foi realizada uma revisão de literatura entre os anos de 2002 e 2008 sobre a utilização da tecnologia de agentes na área da saúde. A simulação baseada em agentes foi uma das áreas categorizadas por meio dos sistemas identificados pela pes-

quisa. Dentre os sistemas descritos no trabalho, o *framework* **Agent.Hospital** destaca-se por ser designado para aplicações distribuídas na área da saúde e por fornecer diferentes interfaces para integrar sistemas de informação pré-existentes. Existem diversos componentes dentro da arquitetura do sistema e, dentre eles, o **SeSAm**, que é um ambiente de simulação para ser utilizado por profissionais da saúde capazes de criar cenários para simulações. Um outro trabalho simula o ambiente dinâmico de um hospital para a criação de um agendamento de planos de tratamentos complexos [Wiesman et al. 2006]. Para isso, é realizada uma coordenação entre todos os departamentos por meio de agentes de software autônomos capazes de simular uma negociação entre eles para a criação de consultas para o tratamento de pacientes.

Esta seção relaciona alguns trabalhos que têm sido desenvolvidos aplicando técnicas inteligentes na saúde e mostra que, resultados positivos em contextos internacionais, tem sido alcançados. As pesquisas evidenciam simulações com arquiteturas de sistemas inteligentes aplicados à área da saúde, no entanto, não são estendidos à atenção primária de saúde.

3. Um ambiente virtual de atendimentos baseado em artefato

Um serviço de saúde alocado estrategicamente em uma agenda de atendimentos torna-se efetivamente um atendimento quando o mesmo é executado por um profissional em algum ambiente. Um ambiente virtual de atendimentos em saúde deve simular a execução dos serviços contidos em uma agenda mensal qualquer. Para isso, o ambiente virtual criado na presente pesquisa baseia-se em artefatos de ambiente de agentes da tecnologia Cartago [Ricci et al. 2009] e considera duas estruturas idênticas de agendas para representar, respectivamente, as agendas planejadas e os atendimentos simulados. O ambiente possui além disso duas propriedades observáveis por agentes que correspondem (1) a ocorrência de eventos negativos que atrapalham a execução das agendas planejadas e (2) uma estrutura para indicar o exato momento (semana, dia e instante k) da evolução do ambiente. Na Figura 1 é possível visualizar o ambiente virtual criado. O modelo visual usa a representação gráfica dos elementos de um artefato ambiental conforme [Ricci et al. 2011].

Figura 1. Modelo visual para o Ambiente Virtual de Atendimentos.

A operação $Simular$ é uma ação externa (motivada por agentes) que autoriza o início da simulação, transformando os serviços planejados em atendimentos simulados. A simulação aplica os serviços encontrados nas agendas planejadas na estrutura "espelho" que representa os atendimentos simulados, considerando a ocorrência de eventos. Dessa forma é possível simular a ocorrência de eventos negativos capazes de desviar a execução dos serviços planejados e, consequentemente, prejudicar a valoração dos indicadores. A Figura 2 exemplifica a ocorrência de um evento e demonstra como acontece o desvio em

um serviço planejado. No exemplo de um atendimento realizado conforme o planejamento, um serviço com $id = 3$ foi alocado na agenda planejada e também foi alocado na agenda simulada. No outro exemplo destacado pela figura, o serviço com $id = 7$ foi alocado na agenda planejada, porém, na agenda simulada foi alocado um outro com $id = 2$, registrando a ocorrência de um desvio na execução do serviço originalmente planejado.

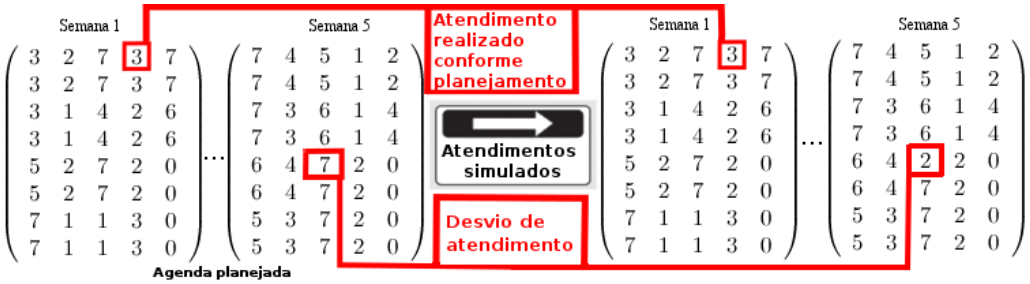


Figura 2. Exemplo de desvio na execução de um atendimento planejado.

A operação $Gerar\_eventos$ usa eventos associados à cada serviço ofertado para, durante a simulação dos atendimentos, dispará-los aleatoriamente ou estocasticamente, dependendo do serviço planejado para o instante k e do cenário a ser simulado. As simulações propostas nessa pesquisa utilizam eventos idealizados a partir de observações e análise de documentos (livros de ocorrências) de uma unidade básica de saúde do município de Viçosa do Ceará.

A $Funcao\_evolucao$ corresponde à forma como o ambiente simulado de atendimentos evolui em função do tempo. Cada transição do ambiente marca uma evolução de um momento m, constituído por instantes k, dias e semanas. O ciclo completo da evolução do ambiente denota o final de um mês de atendimentos simulados. Em uma visão simplificada do processo evolucionário do ambiente nota-se que para cada momento do mês correspondente à simulação do serviço alocado na agenda planejada é executado (alocado) na agenda simulada (dos atendimentos), caso nenhum evento negativo aconteça.

É importante destacar que nas simulações, durante o processo de alocação dos serviços na agenda dos atendimentos simulados, foram realizados dois grupos de testes. No primeiro, nenhum serviço alternativo em relação ao que foi planejado será executado caso um evento aconteça. No segundo, um serviço alternativo foi alocado na agenda de atendimentos simulados caso um evento aconteça. Dessa forma, os eventos possibilitam retratar fatores do mundo real para observar o comportamento dos indicadores em relação às metas e ao planejamento proposto.

4. Simulação de cenários com eventos estocásticos

Objetivando usar a abordagem proposta em [Veras et al. 2014b] no ambiente virtual de atendimentos descrito na Seção 3, criou-se diferentes cenários para simular a evolução do ambiente de atendimentos, levando em conta semelhanças em relação à forma como eventos contingentes ocorrem no mundo real. Para isso, considerou-se que as agendas mensais são estáticas em relação à remarcação de atendimentos que ocorrem no mesmo mês. Isso deu-se por conta de uma dificuldade presente no ambiente real em reagendar os pacientes em um curto período de tempo.

Todas as simulações foram realizadas em um único computador e executada trinta vezes, gerando assim uma amostra dos dados sobre cada cenário. Os testes estatísticos

objetivaram verificar se a similaridade estatística entre os dados das simulações, em qualquer uma das execuções, poderiam ser comparadas com um nível de significância de 95% em relação à média dos dados produzidos em cada cenário. Para isso, analisou-se toda a amostra em busca de verificar se a média dos serviços executados em cada cenário simulado é estatisticamente igual. Se comprovado então as trinta execuções de cada um dos cenários gera dados estatisticamente semelhantes.

O fluxo geral da arquitetura proposta para a simulação dos cenários pode ser visualizado na Figura 3. Na proposta, um Agente de Simulação assume o papel da Equipe e coordena a configuração das variáveis que determinam as demandas e metas. Além disso, o agente interage com o Ambiente de Atendimentos coordenando todos os ciclos da simulação, independente do número de meses que serão gerados.

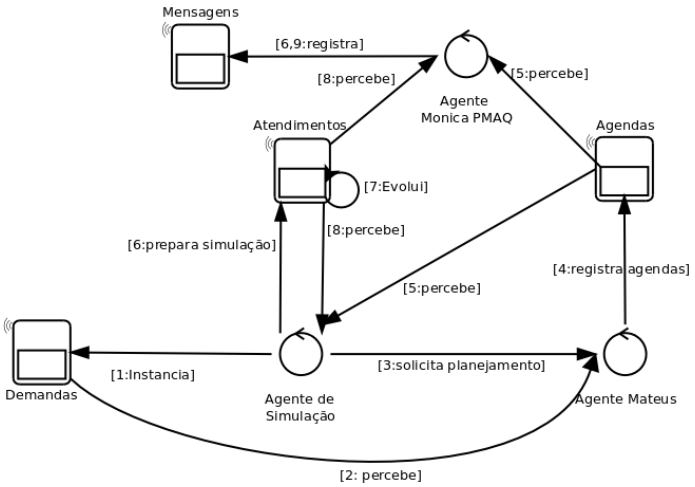


Figura 3. Fluxo para a simulação de atendimentos. Fonte: primária.

Assim, foram criados nesta pesquisa três cenários para os dois grupos de testes (com e sem serviços alternativos) com diferentes formas de aplicação dos eventos negativos na simulação dos atendimentos. Nesse momento, objetivou-se nos testes dos cenários a observação do comportamento dos indicadores diante da influência dos eventos em um ciclo de evolução equivalente à um mês de planejamento/simulação.

Em todos os cenários simulados o Agente Mateus usou como instância os dados populacionais de uma equipe de saúde conforme apresentado em [Veras et al. 2014a], assim como, configuração de recursos humanos a combinação de profissionais indicada pelos autores como a melhor para a instância apresentada, a saber: 2 enfermeiros, 2 médicos e 3 odontólogos.

4.1. Cenário 1: Probabilidade fixa

No Cenário 1, o ambiente de atendimento evolui alocando os serviços na agenda simulada conforme o planejamento, considerando uma probabilidade única de 30% para que ocorra um evento qualquer associado ao serviço ofertado em cada momento da evolução.

Os resultados das simulações com este cenário indicam uma tendência média dos desvios negativos em torno de 30% para os testes sem os serviços alternativos e 20% com os serviços alternativos em relação à oferta dos serviços planejados. O gráfico da Figura 4 mostra o comportamento dos indicadores nos testes sem serviços alternativos

demonstrando que alguns indicadores (1.6, 1.2, 2.1, 3.3, 3.4 e 4.3) obtiveram maiores valores em relação aos testes com os serviços alternativos, no entanto, apenas dois acima da meta (1.6 e 1.2). O gráfico atesta a regularidade fixa programada da influência dos eventos no planejamento, detectada pelo agente Monica PMAQ e registrada no ambiente de Mensagens.

Em um dos testes com e sem serviços alternativos, haviam sido planejados pelo Agente Mateus, respectivamente, 1043 e 1057 serviços distribuídos entre os sete profissionais da equipe. Ao final da evolução em ambos os casos, o Ambiente de Atendimentos registrou a execução de 838 (80,34%) e 756 (71,52%) serviços e a ocorrência de 205 (19,55%) e 301 (28,48%) eventos negativos. Dessa maneira, mesmo com a ocorrência dos eventos no teste sem os serviços alternativos, dois indicadores (1.6 e 1.2) permaneceram acima da meta, enquanto que nove sofreram queda em seus valores. Já no teste com os serviços alternativos, quatro indicadores (1.6, Visitas domiciliares, 5.1 e 5.10) superaram a meta e sete ficaram abaixo do desejado.

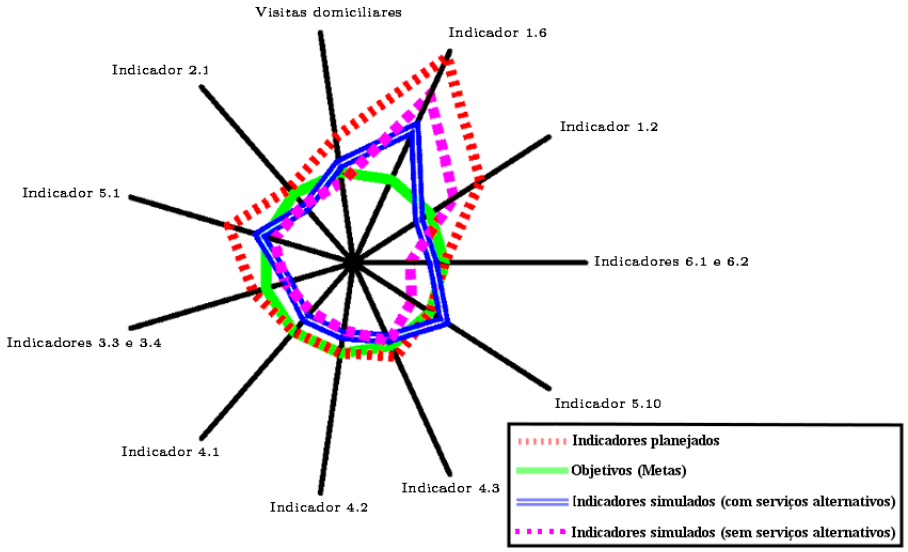


Figura 4. Indicadores após simulação no cenário 1. Fonte: primária.

Assim como neste cenário, qualquer outro o gráfico possuirá as metas configuradas a partir da atuação do Agente de Simulação no ambiente de Demandas. Os indicadores planejados advém do Ambiente das Agendas geradas por meio das ações do Agente Mateus, enquanto que os indicadores simulados surgem durante a evolução do Ambiente de Atendimento. Os dois tipos de indicadores são calculados pelo Agente Monica PMAQ e registrados no Ambiente de Mensagens. Objetivando visualizar os dados gerados, foram plotados gráficos para cada cenário de evolução do ambiente, demonstrados a seguir.

4.2. Cenário 2: Probabilidade fixa aleatória

O Cenário 2 evolui como uma variação do Cenário 1, porém, considera uma probabilidade aleatória com um valor entre 1 e 99 configurada a cada momento da evolução do ambiente. As simulações do cenário resultaram em um desvio negativo médio de 49% em relação aos serviços planejados para os testes sem os serviços alternativos e de 33% com os serviços planejados, como podem ser percebidos no gráfico da Figura 5. O cenário

buscou demostrar o que poderia acontecer com os indicadores em um ambiente consideravelmente caótico, tendo em vista que a todo momento há uma chance aleatorizada que algum problema ocorra.

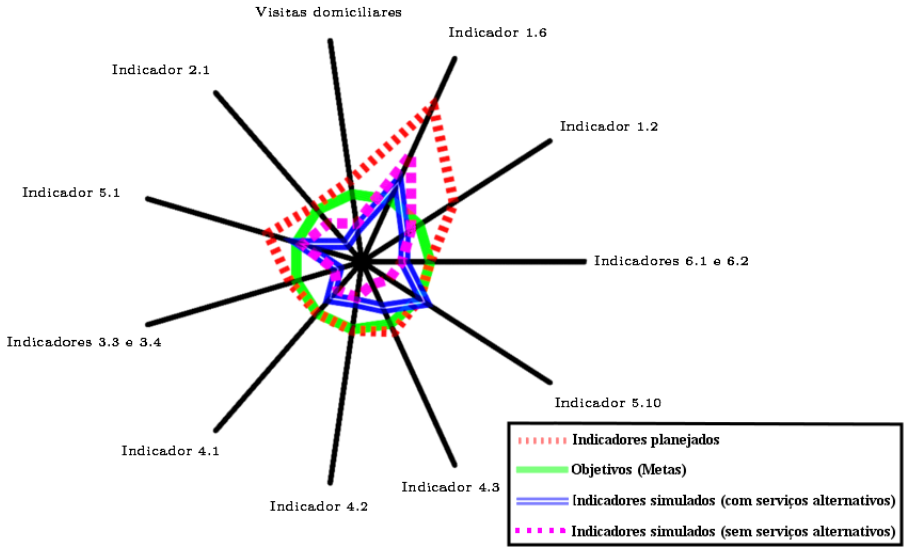


Figura 5. Indicadores após simulação no cenário 2. Fonte: primária.

Em uma das simulações, o Agente Mateus planejou a execução de 1040 (com serviços alternativos) e 1049 (sem serviços alternativos) serviços, porém, ao final das simulações, o Agente Monica PMAQ registrou no ambiente de mensagens 565 (63,94%) para os testes com serviços alternativos e apenas 575 (54%) serviços executados no segundo teste. Durante as simulações ocorreram 375 (36,06%) e 474 (46%) eventos negativos, respectivamente. É importante destacar que, dentre os onze indicadores englobados na presente pesquisa, no primeiro teste apenas dois (5.10 e 1.6) mantiveram seus valores acima da meta da equipe. No segundo caso, um único indicador (1.6) foi valorado acima da meta. Com isso, o cenário de fato conseguiu reproduzir um ambiente caoticamente adverso e observar o comportamento dos indicadores por meio da atuação do Agente Monica PMAQ.

4.3. Cenário 3: Probabilidades associadas aos eventos

De posse dos resultados gerados com os cenários 1 e 2, faz-se necessário refletir sobre eles elencando o seguinte questionamento: até que ponto um cenário com probabilidade única (fixa ou aleatória) para a ocorrência de eventos pode retratar a realidade local de uma equipe de saúde? Neste sentido, parece ser mais realista estabelecer uma probabilidade individual para cada evento e, desse modo, subsidiar formas para compor, calcular e associar os riscos com a ocorrência dos eventos quando considerado um cenário maior que um mês. Partindo disso, o terceiro cenário objetiva aproximar as simulações de um ambiente real de uma Unidade Básica de Saúde. Em busca de fiabilidade para os resultados da simulação nesse novo cenário, consultou-se dez profissionais oriundos de três Equipes de Saúde da Família do município de Viçosa do Ceará. Foi solicitado que os profissionais determinassem valores individuais para as probabilidades de cada evento. Ao final, as equipes sugeriram probabilidades uniformes com valores globais variando entre 1 e 20%, dependendo do serviço ofertado e do evento associado.

O gráfico da Figura 6 traduz os resultados de uma das simulações no cenário atual, demonstrando um desvio médio aproximado de 3,3% para os testes com serviços alternativos e de 5% para os testes sem os serviços alternativos. Com essa configuração, o ambiente de atendimento evoluiu e simulou a execução de 1018 dos 1052 serviços planejados pelo Agente Mateus para o primeiro caso de testes, correspondendo a aproximadamente 96,7% da oferta. No segundo caso, foram executados em torno de 95% dos serviços planejados (993 de um total de 1045). Ao final das simulações no primeiro teste, nove indicadores ficaram acima da meta e apenas dois (4.1 e 4.2) finalizaram com seus valores minimamente abaixo da meta. Para o segundo caso (sem os serviços alternativos), seis indicadores encerraram com seus valores acima da meta e cinco abaixo (4.1, 4.2, 4.3, 5.10 e os indicadores 6.1 e 6.2).

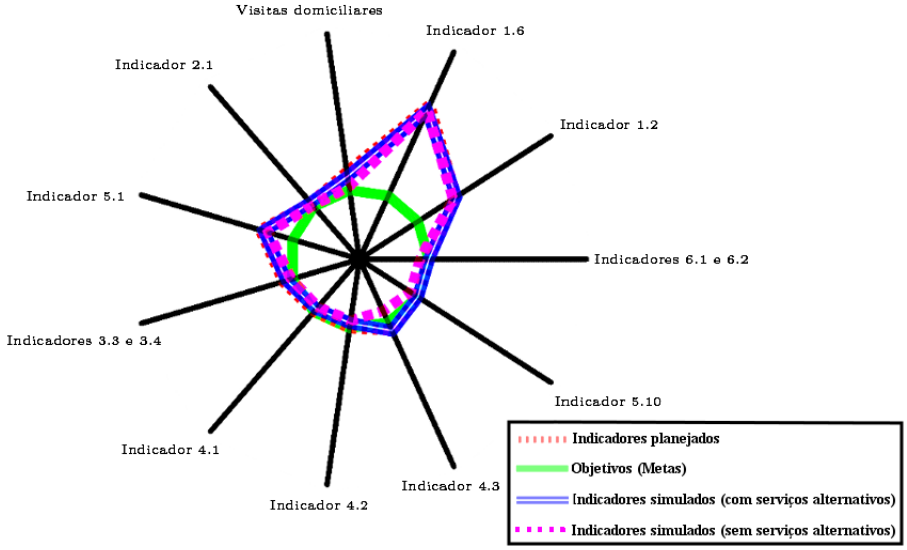


Figura 6. Indicadores após simulação no cenário 3. Fonte: primária.

5. Ampliação da simulação

Os cenários apresentados ressaltam resultados interessantes sobre o comportamento dos indicadores em situações distintas, resumidos pela Tabela 1.

Tabela 1. Resumo dos resultados da simulação de um mês nos três cenários[1].

Cenário[2]	Serviços Alternativos	Total de indicadores monitorados	Indicadores acima da meta	Indicadores abaixo da meta	Percentual de serviços executados (%)
C1	Com	11	4	7	80,34
	Sem		2	9	71,52
C2	Com		2	9	63,94
	Sem		1	10	54,81
C3	Com		9	2	96,76
	Sem		6	5	95,02

[1]Fonte: Primária
[2]Cenários: C1: Probabilidade fixa; C2: Probabilidade fixa aleatória; C3: Probabilidades associadas aos eventos

A Tabela 2 exibe dados da Estatística Descritiva sobre a situação dos indicadores ao final das simulações, assim como, o resultado (*p-valor*) do teste estatístico de *t-student* sobre o percentual dos serviços executados, que mede a influência dos eventos negativos sobre o planejamento. A escolha desse teste deu-se a partir dos resultados positivos sobre

os testes de normalidade para a distribuição do percentual de serviços executados em toda a amostra em relação à sua média. O teste *t-student* para média de uma amostra consiste em medir a probabilidade do valor para a média da amostra em questão apresentar um valor observado ou, ainda, um outro valor mais extremo quando dada a média da amostra.

Tabela 2. Dados estatísticos sobre os resultados da simulação de um mês nos três cenários[1].

Cenário[2]	Serviços Alternativos	Indicadores		Serviços executados			
		DP[3]	Variância	Média (%)	DP[3]	Variância	*p-valor*
C1	Com	1.10	1.21	80.3	1.52	2.32	0.9388
	Sem	1.25	1.56	70.9	1.26	1.60	0.8097
C2	Com	0.81	0.67	67.4	1.42	2.04	0.9005
	Sem	0.54	0.30	51.4	1.28	1.64	0.9935
C3	Com	0.91	0.83	97.0	0.57	0.33	0.6826
	Sem	0.91	0.83	94.8	0.69	0.48	0.8809

[1]Fonte: Primária
[2]Cenários: C1: Probabilidade fixa; C2: Probabilidade fixa aleatória; C3: Probabilidades associadas aos eventos
[3]Desvio padrão

A análise realizada visou atestar a semelhança estatística dos resultados gerados pelas simulações em relação à influência dos eventos negativos no planejamento, provando que não há alterações relevantes entre os dados da amostra (trinta execuções da simulação) no tocante à média dos serviços executados. Vale ressaltar que foi utilizada como apoio aos testes estatísticos a ferramenta de computação estatística R [Gentleman et al. 2015].

Com isso, a Tabela 2 permite concluir que os dados oriundos das amostras (trinta execuções) das simulações nos três cenários considerando 1 mês de planejamento são estatisticamente iguais com significância de 95% podendo atingir uma probabilidade de 99,35% para o valor da média dos serviços executados ser de fato o valor encontrado pelas simulações, como foi o caso do cenário 2 sem serviços alternativos. Essa probabilidade pode ser percebida por meio do *p-valor* de cada cenário. Além disso, os baixos valores de variância e desvio padrão para os indicadores confirmam os valores gerados pela abordagem apresentados na Tabela 1.

Embora a maioria dos indicadores permaneceram acima da meta no último cenário, dito mais realista, isso não significa que eles irão manter o mesmo comportamento em um espaço de tempo equivalente a um ano de atendimentos. Além disso, é conveniente lembrar que o processo de avaliação do PMAQ poderá punir equipes cujos indicadores piorem por três meses ou mais. Portanto, faz-se necessário a ampliação dos ciclos da simulação para, no mínimo, 12 meses de planejamento e execução. Ao ampliar os ciclos da simulação para doze meses obtivemos dados idênticos aos citados nas subseções anteriores, porém, para cada mês simulado. Com isso, o Agente Monica PMAQ registra informações sobre o acompanhamento de cada indicador ao longo dos meses que compõem o ciclo da simulação. A Tabela 3 resume os resultados do monitoramento de todos os indicadores nos três cenários, considerando o ciclo com doze meses de evolução.

Na Figura 7 é possível visualizar um exemplo do monitoramento exercido pelo Agente Monica PMAQ sobre o indicador 2.1, executado durante a evolução completa do ciclo de 12 meses para o Cenário 3 nos testes com serviços alternativos. O gráfico revela que o planejamento manteve-se estável durante praticamente todo o ciclo evolucionário. Como os eventos acontecem estocasticamente as influências negativas variaram bastante durante o ciclo e em alguns momentos (4 meses) o indicador não possuiu seu valor estabilizado acima da meta, mesmo com os serviços alternativos. No mês 2, observa-se

Tabela 3. Resumo dos resultados da simulação com um ciclo de doze meses nos três cenários[1].

Cenário[2]	Serviços Alternativos	Total de indicadores monitorados	Indicadores acima da meta	Indicadores abaixo da meta	Percentual de serviços executados
C1	Com	132	49	83	80,07
	Sem		31	101	80,34
C2	Com		28	104	67,41
	Sem		13	119	51,95
C3	Com		83	49	94,46
	Sem		79	53	90,51

[1]Fonte: Primária
[2]Cenários: C1: Probabilidade fixa; C2: Probabilidade fixa aleatória; C3: Probabilidades associadas aos eventos

uma pequena elevação no planejamento, dado durante a geração das agendas pelo Agente Mateus ao usar suas heurísticas de otimização. Não coincidentemente, no mesmo mês, o valor simulado do indicador também obteve um valor crescente. Esses dois fatores evidenciam que os indicadores sofrem influências tanto do planejamento quanto dos eventos negativos.

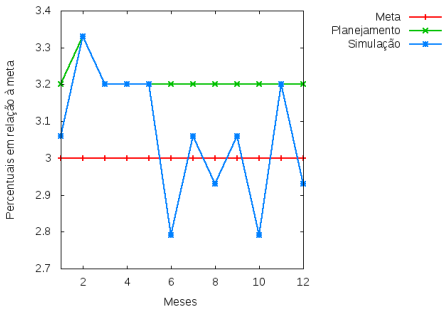


Figura 7. Monitoramento do indicador 2.1 no Cenário 3 com serviços alternativos. Fonte: prmária.

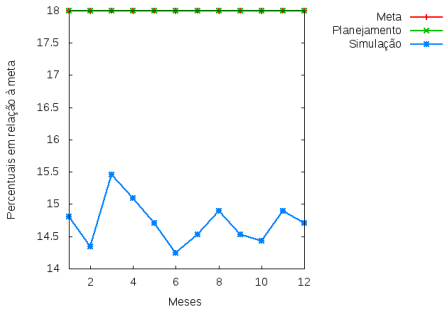


Figura 8. Monitoramento do indicador 4.1 no Cenário 3 com serviços alternativos. Fonte: primária.

Em um outro exemplo, o indicador 4.1, sofre demasiadamente com a ação negativa dos eventos, pois em nenhum momento do ciclo o indicador foi valorado próximo à meta. Vê-se pelo gráfico da Figura 8 que, mesmo diante das ações negativas exercidas pelos eventos, o Agente Mateus não é capaz de recalcular seu planejamento com base nos dados gerados pelos eventos e, ao contrário disso, ele mantém seu planejamento fixo durante o ciclo completo da evolução.

Os gráficos da Figuras 7 e 8 mostram um certo grau de variação dos indicadores em função da evolução do ambiente, principalmente, por conta da influência negativa com características estocásticas sofrida por meio dos eventos que "desviam" a execução do planejamento. O ambiente simulado de atendimentos dispara probabilisticamente os eventos e estes exercem verdadeira influência nos indicadores durante a evolução do ambiente, causando como consequência do disparo probabilístico de eventos, valores mais altos para a variância dos indicadores em comparação com a simulação de 1 mês. Concluise que esses valores são evidenciados com o aumento do tempo das simulações para o correspondente a doze meses. A Tabela 4 mostra os resultados estatísticos do monitoramento dos indicadores nos três cenários, comprovado as características estocásticas para

a ocorrência dos eventos durante as simulações.

Tabela 4. Dados estatísticos sobre os resultados da simulação de doze meses nos três cenários[1].

Cenário[2]	Serviços Alternativos	Indicadores acima da meta			Indicadores abaixo da meta		
		Média	DP[3]	Variância	Média	DP[3]	Variância
C1	Com	45.1	4.07	16.62	86.1	3.35	11.26
	Sem	29.9	3.12	9.74	100.6	4.20	17.09
C2	Com	24.9	2.39	5.75	103.7	4.13	16.52
	Sem	11.5	2.01	4.04	118.2	4.11	16.37
C3	Com	92.9	4.08	16.68	36.8	2.99	8.64
	Sem	75.5	4.25	18.11	53.9	3.33	10.75

[1]Fonte: Primária
[2]Cenários: C1: Probabilidade fixa; C2: Probabilidade fixa aleatória; C3: Probabilidades associadas aos eventos
[3]Desvio padrão

É importante frisar que, com o aumento do tempo da simulação, não houve alteração estatística em relação a média do percentual dos serviços executados. Isso implica que a quantidade de serviços executados em cada cenário é praticamente a mesma em toda a amostra, consequentemente, as variações se dão nas execuções dos serviços ofertados. Esse comportamento certifica o padrão na evolução do ambiente simulado nos cenários descritos em relação à ocorrência dos eventos e solidifica o comportamento estocástico desses eventos em relação à influência negativa dos indicadores, pois a primitiva básica dos indicadores (os serviços alocados nas agendas) sofre desvios não determinísticos.

6. Considerações finais

Este artigo objetivou apresentar uma simulação baseada em agentes para retratar em um ambiente virtual a execução de serviços de saúde planejados por um agente inteligente usando agendas de atendimento. A simulação considerou diferentes cenários para a evolução dos atendimentos em um intervalo de tempo compatível com um ano de planejamento e, para tanto, um modelo virtual foi definido contendo uma função de evolução capaz de simular cenários específicos englobando a ocorrência de eventos contingentes capazes de influenciar negativamente a execução do planejamento. Neste trabalho, três diferentes cenários foram idealizados e incorporados ao ambiente virtual.

Usando a abordagem proposta por [Veras et al. 2014b] foram realizados testes com a simulação apresentada neste artigo que enumeraram resultados satisfatórios em relação ao fortalecimento do uso da orientação por metas como estratégia de planejamento inteligente. No entanto, os avanços da abordagem obtidos a partir dos resultados das simulações apontam para uma necessidade em buscar o equilíbrio dos indicadores diante dos cenários descritos na Seção 4, pois comprovou-se que apenas o planejamento orientado por metas não é suficientemente eficaz para manter, por meio de agendas planejadas, os indicadores equilibrados em relação às suas metas em situações nas quais ocorrem eventos negativos. Como trabalhos posteriores, almeja-se equilibrar os indicadores diante desses cenários hostis ou, ainda, aproximar seus valores das metas desejadas pela equipe.

Agradecimentos

Agradecemos os suportes financeiros fornecidos pela Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico (FUNCAP) por meio do edital PPSUS-11/2013 e pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) por meio de bolsa de estudo.

Referências

BRASIL, M. d. S. (2012). *Programa nacional de melhoria do acesso e da qualidade da atenção básica (PMAQ): manual instrutivo*. Ministério da Saúde: (Série A. Normas e Manuais Técnicos).

Cardoso, L., Marins, F., Portela, F., Abelha, A., and Machado, J. (2014). Healthcare interoperability through intelligent agent technology. *Procedia Technology*, 16:1334–1341.

David, N. and Sichman, J. S. (2013). O papel da emergência em simulações de sociedades de agentes artificiais. *Ciência e Cultura*, 65(4):36–41.

Davidsson, P., Holmgren, J., Kyhlbäck, H., Mengistu, D., and Persson, M. (2007). Applications of agent based simulation. In *Multi-Agent-Based Simulation VII*, pages 15–27. Springer.

Fox, J., Beveridge, M., and Glasspool, D. (2003). Understanding intelligent agents: analysis and synthesis. *AI communications*, 16(3):139–152.

Gentleman, R., Ihaka, R., Bates, D., et al. (2015). The r project for statistical computing.

Isern, D., Sánchez, D., and Moreno, A. (2010). Agents applied in health care: A review. *International Journal of Medical Informatics*, 79(3):145–166.

Jacobson, S. H., Hall, S. N., and Swisher, J. R. (2013). Discrete-event simulation of health care systems. In *Patient Flow*, pages 273–309. Springer.

Patel, V. L., Shortliffe, E. H., Stefanelli, M., Szolovits, P., Berthold, M. R., Bellazzi, R., and Abu-Hanna, A. (2009). The coming of age of artificial intelligence in medicine. *Artificial intelligence in medicine*, 46(1):5–17.

Ricci, A., Piunti, M., and Viroli, M. (2011). Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192.

Ricci, A., Piunti, M., Viroli, M., and Omicini, A. (2009). Environment programming in cartago. In *Multi-Agent Programming:*, pages 259–288. Springer.

Shortliffe, E. H. (1993). The adolescence of ai in medicine: will the field come of age in the'90s? *Artificial intelligence in medicine*, 5(2):93–106.

Veras, N. d. L., Cortés, M. I., and Campos, G. A. L. (2014a). Planejamento de atendimentos em saúde orientado por metas. In *Proceedings do XLVI Simpósio Brasileiro de Pesquisa Operacional*, pages 2781–2792. SOBRAPO.

Veras, N. d. L., Cortés, M. I., and Campos, G. A. L. (2014b). Uma abordagem baseada em agentes para planejamento e monitoramento de serviços de saúde. In *Proceedings do VIII Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 25–36. PUCRS.

Wiesman, F., Hasman, A., Braun, L., and van den Herik, J. (2006). Information retrieval in medicine: The visual and the invisible (information retrieval im gesundheitswesen: Das visuelle und das unsichtbare). *it-Information Technology (vormals it+ ti)*, 48(1/2006):24–32.

AnthillRL: Multi-agent Reinforcement Learning

Anibal Sólon Heinsfeld, Felipe Meneguzzi

[1]School of Computer Science (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

anibal.heinsfeld@acad.pucrs.br, felipe.meneguzzi@pucrs.br

Abstract. *Ants are essentially social insects, and their organizational dependency reflects directly on their survival. Due to their social nature, ant society provides a rich model for analysing properties of multiagent systems such as collaboration and effectiveness of collective action. Modelling natural behaviour of ants can help understand their actions, as well as studying better forms of cooperation and competition for other multiagent models. In this paper, we model an ant society within a stochastic environment in which ant behaviour is generated using reinforcement learning to generate paths towards their goal. We test this model using a variable number of agents and learning parameters and report on the results.*

1. Introduction

Ants societies are described as a superorganism because of their social organization and cooperation [Wilson and Hölldobler 2005]. Besides the mutual care of their brood, there is a division of labour based in the ant's reproduction capacity and behavioural groups. According to this morphological and behavioural differences, the ants are classified in caste such as Queens, Workers and Soldiers. To achieve this status as a superorganism, the communication of ants is a differential point and occurs via secreted chemicals, sounds and touch [Wilson and Hölldobler 2005]. Pheromones are chemical substances used for purpose of communication and they are secreted from the body of an individual to affect the decision making of other individuals that detect it. Different types of pheromones are produced, such as to attract mates, to signal danger to the colony or to give directions about a location.

Given that ants are a fundamentally a social species, its modeling in a multi-agent platform is an interesting way to verify the collective behaviour of this decentralized and self-organized systems and to apply its results on optimization heuristics. Many algorithms were created inspired on social behaviour, as of insects: ants [Shtovba 2005] and honey bees [Sonmez 2011], and even from other phylum, as bats [Yang and He 2013] and gray wolves [Mirjalili et al. 2014].

In this paper, we have two main objectives. First, we want to model an anthill in order to better understand behaviours of ants collaborating as a group. Using JaCaMo architecture, each ant is modelled as an agent in order to act collaboratively to achieve the global goal: find the shortest path to environment resources. Second, we aim to test Reinforcement Learning methods with multi-agents system in order to evaluate how agents distributivity affects performance in our scenario. Using Markov Decision Processes to

model environment states and actions that agents can take, we aim to achieve randomness of real-world environment, and with Reinforcement Learning, we want to simulate pheromones that reinforce the shortest path to a resource.

This paper is organized as follows. Section 2 reviews the background on MDP and Reinforcement Learning. In Section 6 we address related work using Multi-agents and Reinforcement Learning. Section 3 and Section 4 describe our method to achieve the proposed objectives and the tests methodology. In the Section 5 we present our tests results and discussion. Finally, in Section 7, we conclude with final considerations and point the direction for future work.

2. Background

2.1. Markov Decision Process (MDPs)

Markov decision processes provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker, thus, it is a stochastic process. Using this control process, the decision maker is in a known state and can choose an action available for the current state. The action has a probability to occur and lead to the subsequent state, so the process brings about a random state to the decision maker. The probability to reach the subsequent states depends only on the current state and the action taken, satisfying the Markov Property. The previous taken actions do not interfere in the probability distributions of the current possible actions. For example, given an exploratory robot agent that must recognize its territory and control its battery level, the actions available to this robot are: search (as its mission), recharge its battery and wait for the recharging process. In a very simple agent model, the agent just has two states: high level of battery and low level of battery. In Figure 1, the stochastic automaton of an an exploratory robot agent demonstrates how the taken actions lead to states with a certain probability. The automaton has probabilities for an action to happens and a reward. At the *high* state, the agent can take the *wait* action, keeping the agent in the same state with a probability of 100% (denoted as 1) with reward R^{wait}. Otherwise, the action search can lead to both described states with a probability of α (or its complementary value 1-α) with reward R^{search}. It is important to note that, when in *low* state, the search action can lead to a negative reward (a penalty). This penalty for an agent is important in order to make it avoid this action.

In order to achieve better results, agent must find a policy which specifies which action to take in each state, so as to maximize the sum of values from a specific reward function. The problem is to specify a function to find a policy with maximum expected reward, called the optimal policy, based on a reward function. In this work, with regard to find the optimal policy, or the shortest path to a resource, we use Reinforcement Learning methods.

2.2. Reinforcement Learning

Reinforcement learning is defined as learning what to do in order to maximize some notion of cumulative reward [Sutton and Barto 1998]. In the context of MDP modelling, the intelligent agent does not know what actions it must take in its environment. Instead it must explore the environment by exploring actions and states and measuring the resulting reward. The chosen action has two consequences: first, it affects the reward of the decision and second, it changes the subsequent possible actions and its rewards.

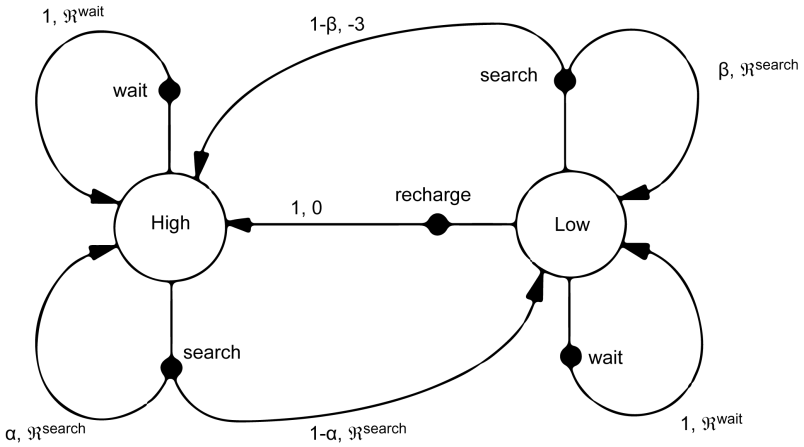


Figure 1. The Markov model of the robot states and actions

The feature of Reinforcement Learning of trial-and-error search makes it a distinguishable area of Machine Learning: it refers to learning from direct interaction with an environment. The learning agent receive just a reinforcement signal from the environment instead of a precise error signal measuring the discrepancy between the realized and a known expected performance. In Reinforcement Learning, the agent must explore the environment to learn. But also, aiming to converge the learned rewards towards an optimal solution, the agent must exploit its current knowledge about the environment, probably leading it to the search space closer to the solution of the problem. The agent should, for example, execute the same actions that in the past produce a good reward, assuming that reproducing the same behaviour will lead to the same reward. Otherwise, through the exploration, the agent can find a better reward than this known reward. So, it is important to define a proper weight between this two strategies, aiming the maximization of the cumulative reward.

There are several algorithms for Reinforcement Learning [Kaelbling et al. 1996]. The algorithm used in this work is the Q-learning. Q-learning is a form of model-free reinforcement learning [Sutton and Barto 1998], in which the agent does not need an internal model of environment to work with it. It works by learning an action-value function that provide the expected utility of taking a given action in a given state and its core is a simple value iteration update.

The remarkable characteristic of Q-learning is that it can determine the immediate rewards and delayed rewards, depending on its parametrisation. The algorithm creates a policy, that is a rule the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. Q-learning is an on-line reinforcement technique, which means that the policy of the agent is improved on the fly.

The update rule for Q of the state-action combination calculates an estimate value given a state and a chosen action by the agent with the formulae:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \times (R(s_t, a_t) + \gamma \times \Delta Q(s_t, a_t 0)) \quad (1)$$

$$\Delta Q(s_t, a_t) = maxQ(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \quad (2)$$

The learning rate (represented by α) defines how much the newly acquired Q-value influences the new Q-value. The learning rate can take values between zero and one. A rate close to zero makes the agent ignores the new information, so it does not learn anything. Otherwise, when the rate tends to one, the agent only considers the new Q-value and ignores the old one.

The discount factor (represented by γ) makes rewards earned earlier more valuable than those received later. It assumes values between zero and one.

Analysing Figure 2, it is possible to visually understand how information influence the algorithm. The assumed action (blue) will have its Q value changed, gathering the greater Q value of the possible actions (green) that the selected action will lead to.

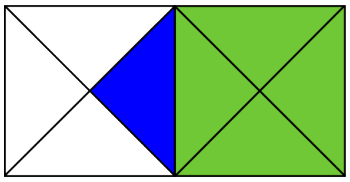


Figure 2. Taking an action (green) will trigger the Q(State, Action) calculus, using the neighbour's (blue) Q-value.

3. Proposed Approach

The proposed approach revolves around the search of an optimal action-selection policy for any given Markov Decision Process. Using a Multi-Agent System model, the goal is the convergence to the shorten path to a given destination. The environment surrounds the anthill world and, once the ant (the agent) is outside of the anthill, it must search the proposed grid for a source of sugar (modelled as a donut). The rewards of Q-Learning state-action pairs is modelled as pheromones. This system uses Multi-Agent approach in order to optimise the Reinforcement Learning algorithm by computing rewards concurrently and distributing their values across agents in the form of pheromones. Thus, it models correctly the ants behaviour by its perceiving, acting and sharing information through pheromones. The agents follow the Belief-Desire-Intention model, separating the activity of selecting a plan provided by its plan library from the execution of currently active plans.

We model the Multi-Agent System using JaCaMo [Boissier et al. 2013]. JaCaMo's joins three Multi-Agent technologies aiming the integration of agents, environments and organizations. These technologies are, respectively: Jason [Bordini et al. 2007], a agent-oriented programming language and platform that extends AgentSpeak; Cartago [Ricci et al. 2011], a platform that provides an general-purpose shared environment to agent interactions; and Moise [Hübner et al. 2010], an organisational model that provides structural, functional and deontic specifications. This integra-

tion aids on rewards synchronization between agents, since Cartago shares equally the environment information to Jason-programmed agents.

The movimentation of the agents across the environment was developed in the AgentSpeak-like language, Jason, with a set of conditionals which gives the direction of what plans must be executed for the current state. The decision process modeled over the Markov Decision Process for the learning and reasoning about the next actions and states is developed in Java, the language and environment that wraps the Jason, Cartago and Moise.

The map described in the Figure 3 was used for the tests. The modeled pheromones are the trail and danger pheromones, in order to stimulate ants follow or avoid a certain path, respectively. Each triangle, that represents the possible actions in a location of the grid, has its own measure of pheromone. So the ant can reason about which direction is the most valuable.

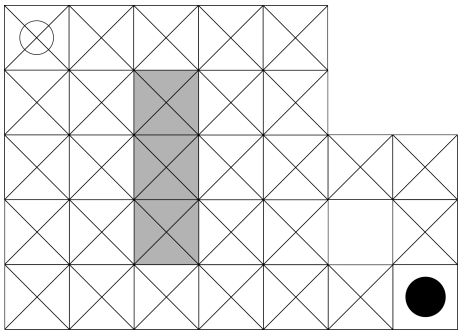


Figure 3. The map used for the tests. The white circle is the initial state, the black circle is the goal and the gray states are dangerous areas.

At this simulation, an ant can choose the actions UP, DOWN, LEFT, RIGHT when it is possible. In the borders of the scenario, where there is no surrounding state, it is impossible to assume an action aiming regions outside our scenario.

The agents use an ϵ-Greedy policy. For a given ϵ, between 0 and 1, the agent assume a greedy position choosing the most valuable utilities of the actions or choose a random action, based in a random distribuition. In the random mode, the action is described as exploratory, and when the next action is choosen based on its valuation, this actions is called exploitatory. In this system, ϵ assumes an initial value and, according to the current episode, the ϵ tends to an exploitatory value. It ensures that the ants take benefits of the learned knowledge. The ϵ value of the episode is described with the following formula:

$$\epsilon_{Episode} = \epsilon_0 + ((1 - \epsilon_0) * Episode/FinalEpisode) \tag{3}$$

The algorithm below describes how the action-selection through ϵ-Greedy works:

$\epsilon \leftarrow \epsilon_0 + ((1 - \epsilon_0) * Episode/FinalEpisode)$

$R \leftarrow Random[0, 1]$

if $\epsilon \geq$R **then**

Action ← Most-Valuable Action

else

Action ← Random Action

end if

Putting it all together, with the ϵ-Greedy policy and Q-learning in our MDP-modeled environment, the algorithm executed by the agents for each episode is:

CurrentState ← State_0

while CurrentState is not the Goal State **do**

Action ← Action chosen by ϵ-Greedy policy

Compute Q(CurrentState, Action)

CurrentState ← State

end while

4. Experimental Methodology

To measure the performance of the whole Multi-Agent System, the chosen metric is the number of episodes before convergence. An episode in this work is described as the traversing of the agent from initial position to the goal. So, an episode ending when some terminal state is reached. Convergence occurs when in the next 5 episodes the values remain unchanged. For a single agent, the episode of convergence is the episode that the agent reaches. The considered episode of convergence for multiple agents is the superior episode that one of the agents reaches. This metric supplies the lack of determinism on multi-agents systems because each agent can reach different episodes of convergence and the learning process is only finished when all agents reach the convergence.

In order to evaluate different scenarios for the proposed set of algorithms and architecture, the following parameters are changed across the tests:

- Number of agents: A single agent or multiple agents;
- Maximum number of episodes: The stop criteria for the learning algorithm. This metric influences the ϵ for each episode.
- Initial ϵ: The value of the ϵ variable at the ϵ-Greedy policy;

The test was executed five times for each parametrization, because of stochastic of the system. The multiple results are summarized through a mean operation. The α, the learning parameter of Q-Learning formula, is set to 0.8 in order to learn new information, but also consider the old one. The γ, the discount factor, is set to 1.0. This value was chosen to achieve additive rewards towards long-term rewards for a faster convergence.

5. Experimental Analysis

The experiments follow the variation of parametrization: the number of agents is varied in one, three and five agents; the maximum number of episodes that the run can reach is

varied in 30 and 100; and the initial ϵ of Q-learning algorithm is varied in 0.4, 0.6 and 0.8. The gathered results is shown in Table 5 and Table 5.

Number of Agents	Max Episode	Start Epsilon	Episode of Convergence
1	30	0.8	23
3	30	0.8	14
5	30	0.8	9
1	30	0.6	24
3	30	0.6	14
5	30	0.6	8
1	30	0.4	26
3	30	0.4	16
5	30	0.4	9

Table 1. Results of parameter permutation: Max Episode equals to 30

Number of Agents	Max Episode	Start Epsilon	Episode of Convergence
1	100	0.8	50
3	100	0.8	38
5	100	0.8	26
1	100	0.6	53
3	100	0.6	35
5	100	0.6	29
1	100	0.4	50
3	100	0.4	39
5	100	0.4	31

Table 2. Results of parameter permutation: Max Episode equals to 100

The number of agents influences convergence directly, because the knowledge is distributed among the agents and they execute multiple valuation of the actions at the same time. When there is just one agent, it takes more episodes to reach the converged policy.

It is clear that the decrease of the maximum number of episodes converges faster because of the influence of this parametrization in the ϵ value at each episode. Since the numerical difference between the current episode and the maximum episode is lesser, the tendency of the ϵ to one (describing 100% of chance to use the Greedy policy, in the exploitatory mode).

As showed in the results, the value of the initial Epsilon do not affect directly the convergence. This indifference probably occurs because of the high influence of the variation of ϵ during the execution.

The convergence of the map is showed at the Figure 4. The green marks indicate the possible actions of the position. It is possible to see that the most-valuable actions (marked as blue) leads to the food.

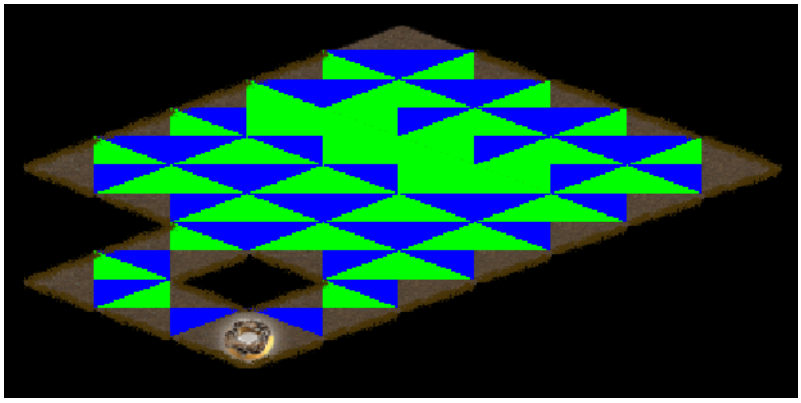


Figure 4. The policy generated by Q-learning.

6. Related Work

Several Multi-agent Systems use Reinforcement Learning with independent or cooperative approaches.

Tan [Tan 1993] uses both in a scenario with two sets of rivals: the hunter and the prey. He use Q-Learning to estimate the reward of each pair of state and action. He demonstrate that reinforcement learning agents can learn cooperative behaviour in a simulated social environment and, as this work, assume three types of interactions: a online communication of perceptions, an episodic communication and the sharing of the learned knowledge.

In the work of Subramanian et al. [Subramanian et al. 1998], the idea of mimicry of ants is used to find better routes in a dynamic computational network. They propose two algorithms that are resilient against random corruption of the current path: the first algorithm is a shortest path algorithm, while in their second approach, they introduce a concept of uniform ant. The probability of the uniform ant choose a route is equal to each other possible routes, so it does not suffer with route oscillation problems, which is a common problem in the first algorithm.

The work of Zhang et al. [Zhang et al. 2012] approaches the application of Reinforcement Learning to optimize the path-finding algorithm for robots. It is based on Markov Decision Process and use Q-Learning to learn the optimal policy through trial-and-error, an intrinsic property of Reinforcement Learning. They also explore the point of non-deterministic rewards and actions, with an interesting formula that consider the number of visits of the agent to that state as an learning rate.

In the work of Gambardella et al.[Gambardella et al. 1995], they propose a distributed algorithm based on Q-learning for combinatorial optimization using the metaphor of ant colonies, to solve the solution of symmetric and asymmetric instances of the traveling salesman problem, a combinatorial optimization problem. The symmetry is defined by the distance between two cities: if the distance to go from city A to city B is equal to go from city B to city A, it is assumed to be symmetric. Otherwise, it is called asymmetric. The remarkable modification made from Q-learning is the use of an heuristic value.

Finally, the survey of Busoniu et al. [Busoniu et al. 2008] brings about the use of Reinforcement Learning in a Multi-Agent System. It state that are two key points to the

design decision of the Multi-Agent Reinforcement Learning: the stability of the agents' learning dynamics and the adaption to the changing behaviour of the other agents. The survey introduces the single-agent and Multi-Agent approach for Reinforcement Learning using the Markov Decision Process and the Stochastic Game, an generalization of Markov Decision Process that in a environment for one or more agents it assumes probabilistic transitions. Surrounding this previous topic, it also cites the Nash Equilibrium, an important concept in Multi-Agent System that is an situation where no agent can benefit by changing its strategy as long as all other agents keep their strategies constant. In the section of benefits and challenges of the survey, it is important to note that the use of multi-agent can bring an speed-up to the algorithm, because of the parallel computation when the agents exploit the decentralized structure of the task. But also sharing the learning information among the agents can help to learn fast and better a converged solution: the authors argue about methods of learning from the knowledge of other agents. They cite the possibility of modeling this interaction as an skilled agents that may serve as teachers for the learner or the learner may watch and imitate the skilled agents to reason about its observation and learn. In the other part of the section, they bring about the importance of a good policy to trade-off between the exploratory and exploitatory strategy.

7. Conclusions and Future Work

This work demonstrates that intelligent agents based on reinforcement learning can cooperate to reach a good result on a multi-agent environment. Also the work involves a variety of subjects, from theoretical to technical, demonstration the application of complex learning on BDI-based multi-agents, as the simulation of a insect that uses a natural reinforcement learning methodology.

It is important to note that with a larger number of agents, the learner converges faster, since the reinforcement learning approach applied in this work distributes the knowledge among the agents and the agents can explore quickly the proposed map. The variable ϵ-Greedy action-selection policy gives a faster convergence, since the agents tended to follow the most-valuable path instead of explore even more the map.

For future work, we aim to use other reinforcement learning algorithms, as SARSA, an on-line Q-learning method. Comparison with other algorithms can bring different performance results and possible integrations and optimizations. The simulation of partially observable Markov Decision Process (POMDP), a generalization of Markov Decision Process that assumes probabilistic distributions of the underlying state, would be interesting, since it model the real world more precisely. The design of dynamic characteristics, such as the resource location and quantity, as well the competition between sets of agents to reach the resource, are also good cases to explore with reinforcement learning, since the applied algorithm and the agents must deal with changes in the environment.

References

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with jacamo. *Sci. Comput. Program.*, 78(6):747–761.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*.

Busoniu, L., Babuska, R., and De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172.

Gambardella, L. M., Dorigo, M., and Bruxelles, U. L. D. (1995). Ant-q: A reinforcement learning approach to the traveling salesman problem. pages 252–260. Morgan Kaufmann.

Hübner, J., Boissier, O., Kitio, R., and Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285.

Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey wolf optimizer. *Adv. Eng. Softw.*, 69:46–61.

Ricci, A., Piunti, M., and Viroli, M. (2011). Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall.

Shtovba, S. (2005). Ant algorithms: Theory and applications. *Programming and Computer Software*, 31(4):167–178.

Sonmez, M. (2011). Artificial bee colony algorithm for optimization of truss structures. *Applied Soft Computing*, 11(2):2406 – 2418. The Impact of Soft Computing for the Progress of Artificial Intelligence.

Sorici, A., Picard, G., Boissier, O., Santi, A., and Hübner, J. F. (2012). Multi-agent oriented reorganisation within the jacamo infrastructure. In *Proceedings of The Third International Workshop on Iinfraestructures and tools for multiagent systems: ITMAS*, pages 135–148.

Subramanian, D., Druschel, P., and Chen, J. (1998). Ants and reinforcement learning: A case study in routing in dynamic networks. In *In IJCAI (2*, pages 832–838. Morgan Kaufmann.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Tan, M. (1993). *Multi-Agent Reinforcement Learning Independent vs Cooperative Agents*. PhD thesis, GTE Laboratories Incorporated.

Wilson, E. O. and Hölldobler, B. (2005). Eusociality: origin and consequences. *Proceedings of the National Academy of Sciences of the United States of America*, 102(38):13367–13371.

Yang, X.-S. and He, X. (2013). Bat algorithm: Literature review and applications. *Int. J. Bio-Inspired Comput.*, 5(3):141–149.

Zhang, Q., Li, M., Wang, X., and Zhang, Y. (2012). Reinforcement learning in robot path optimization. *Journal of Software*, 7(3):657–662.

Incorporando Filtros de Percepção para Aumentar o Desempenho de Agentes Jason

Márcio F. Stabile Jr.[1]*, **Jaime S. Sichman**[1,2]†

[1]Programa de Pós-Graduação em Ciência da Computação
Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

[2]Escola Politécnica – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

mstabile@ime.usp.br, jaime.sichman@poli.usp.br

Abstract. *The use of simulation systems is gradually increasing due to its ability to reproduce the real world. In particular, we are interested in those systems where simulated humans interact with an environment. To better exploit the potential of these systems, we can make use of intelligent autonomous agents to simulate human behavior. The use of the Jason interpreter in the creation and implementation of intelligent agents can then reduce the effort required for its development. As agents become more sophisticated, also increases the processing time required for the agent to perceive its environment, to reason about their actions, and to decide the best action to take. Such time interval can be bigger than the limit required by the simulator for an agent to act on the environment. Hence, the objective of this research is to modify the Jason interpreter in order to filter his perceptions of the environment, so as to reduce the time taken to process them and consequently reduce the agent total running time. Finally, an empirical analysis is made in order to verify the impact of perception filters on the reduction in agent's time response and performance.*

Resumo. *Sistemas de simulação são cada vez mais utilizados devido à sua capacidade de reproduzir o mundo real. Em particular, estamos interessados em sistemas que simulam a interação entre humanos e o ambiente. Para melhor utilizar o potencial desses sistemas, podemos fazer uso de agentes autônomos inteligentes para simular o comportamento humano. Utilizar o interpretador Jason no processo de criação e execução dos agentes inteligentes pode então reduzir o esforço necessário para seu desenvolvimento. Conforme os agentes se tornam mais sofisticados, cresce também o tempo de processamento necessário para que o agente possa perceber o ambiente, raciocinar sobre suas ações, e decidir qual deve ser aplicada. Tal tempo pode ser maior do que o disponibilizado pelo simulador para que o agente efetue uma ação no ambiente. O objetivo desta pesquisa é então modificar o interpretador Jason para que este seja capaz de filtrar as percepções vindas do ambiente, de modo a reduzir o tempo levado para processá-las e consequentemente reduzir o tempo total de execução do agente. Finalmente, é feita uma análise empírica para verificar o*

*Márcio F. Stabile Jr. é financiado pelo CNPq.
†Jaime Simao Sichman é parcialmente financiado pelo CNPq, proc. 303950/2013-7.

impacto dos filtros de percepção na redução do tempo de processamento e do desempenho do agente.

1. Introdução

Os sistemas de simulação se fazem cada vez mais importantes pela sua capacidade de imitar a realidade. Em particular, estamos interessados em sistemas que simulam a interação entre humanos e o ambiente. Por permitir a avaliação de diferentes cenários, esses sistemas são uma ferramenta poderosa para avaliar os resultados de ações sobre um ambiente, já que eles permitem aos usuários compreender as características de um ambiente ou uma situação complexa sem a necessidade de reproduzi-la no mundo real, podendo identificar potenciais áreas problemáticas, e assim realizar modificações apropriadas no projeto final.

Os sistemas de simulação como o da RoboCup[1] e do MAPC[2] se tornam mais complexos a cada ano, fazendo com que seja necessária uma maior capacidade de tomada de decisão. Uma importante característica destes sistemas de simulação é que a execução do simulador é realizada em passos. No início de cada passo o simulador envia as informações do ambiente para os agentes. Os agentes devem então decidir suas ações e as enviar de volta ao servidor em um determinado tempo pré-fixado.

Modelar o comportamento humano para esse tipo de situação é extremamente difícil, pois além de modelar o raciocínio individual, é necessário também modelar a tomada de decisão em grupo. Uma das maneiras utilizadas é a criação de agentes inteligentes que imitam as características gerais dos seres humanos. Integrar tecnologias baseadas em agentes em mundos criados por sistemas de simulação permite aumentar o grau de realismo na representação dos comportamentos.

Jason[3] [Bordini et al. 2007b] é um interpretador para uma versão estendida de AgentSpeak [Rao 1996] (linguagem de programação lógica com eventos e ações) que pode ser utilizado para o desenvolvimento de Agentes inteligentes. Jason implementa a semântica operacional dessa linguagem e proporciona uma plataforma para o desenvolvimento de sistemas multi-agentes. Jason tem seu código aberto e é construído na linguagem Java.

Dado que o ciclo de raciocínio do Jason é independente do simulador, realizar a integração dos dois é uma tarefa difícil, uma vez que aspectos como o tempo de resposta dos agentes pode influenciar diretamente no seu desempenho. Com isso temos que por muitas vezes o tempo levado para encontrar a melhor ação é maior que o tempo fixado pelo simulador, o que acarreta na perda da ação e inércia do agente durante todo o passo. Além disso, quanto mais complexos ficam os agentes, maior é o tempo utilizado para se processar, agravando mais o problema.

Uma possível solução para o problema seria a utilização de *filtros de percepção* de forma a reduzir o número de percepções recebidas e assim possivelmente reduzir o tempo de processamento do agente.

O objetivo desta pesquisa é então construir uma modificação do interpretador Jason que seja capaz de filtrar as percepções vindas do ambiente para que seja possível

[1]RoboCup: `http://www.robocup.org/`
[2]MAPC: `https://www.multiagentcontest.org/`
[3]Jason: `http://jason.sourceforge.net/wp/`

responder a perguntas como: (i) Dado um certo limite de tempo de resposta do simulador, é possível para o agente processar todas as informações provenientes das percepções? (ii) Ao utilizar um filtro de percepções, é possível melhorar o tempo de resposta? (iii) Caso melhore, como se identificaria o filtro adequado?

2. Estado da Arte

Em [van Oijen and Dignum 2011] os autores tem como objetivo a integração de agentes criados em plataformas multi-agentes como 2APL [Dastani et al. 2007], Jadex [Braubach et al. 2003] e Jason [Bordini et al. 2007a] com ambientes virtuais em tempo real, mais especificamente jogos de computador. Nesse artigo foi proposto um *middleware* para fazer a integração entre os dois sistemas. Esse *middleware* é responsável por receber sinais do ambiente virtual e fazer todo o processamento para entregar ao agente somente o conjunto de percepções que sejam de seu interesse. Isso acontece pois, como descrito no artigo, um dos problemas do paradigma BDI quando se une os agentes a ambientes virtuais é a falta de controle sobre as percepções. Se não houver alguma forma de percepção direcionada ao objetivo, o agente pode ser inundado com informações sensoriais que podem resultar no raciocínio sobre uma enorme quantidade de informações irrelevantes. Além disso, a falta de controle também não é realista quando olhamos para a fisiologia da percepção humana. Como a atenção é considerada um recurso limitado, não se pode atender a todos os aspectos do ambiente. Durante a execução de uma tarefa, os seres humanos, portanto, tendem a direcionar sua atenção para informações selecionadas do ambiente que podem apoiá-los na realização da tarefa. Isto sugere que uma abordagem similar deve ser utilizada também para agentes BDI.

Para atacar esse problema foi criado um Gerenciador de Assinaturas de Interesse que contém assinaturas para informações específicas do ambiente. Quando um agente adota um objetivo, o sistema cria automaticamente um conjunto de inscrições de informações sobre o ambiente usando o Gerenciador de Assinaturas de Interesse. Assim, quando um objetivo é alcançado ou descartado, o sistema automaticamente cancela a inscrição da informação correspondente. O sistema é capaz de fazer isso com base em uma tabela criada off-line que mapeia os objetivos do agente para conjuntos de assinaturas. Uma exemplo de tabela pode ser visto na figura 1. Nela é possível ver que existem condições específicas de quando e com que frequência se deve perceber determinados objetos como pessoas e carros.

O objetivo de [Bordeux et al. 1999] é criar uma extensão para uma arquitetura de software para a gestão das acções chamada AGENTlib, criada em [Boulic et al. 1997], com um mecanismo de percepção. Esse mecanismo de percepção conta com um filtro de percepções que tem como objetivo filtrar eficientemente informações valiosas da cena onde se encontra o agente. No artigo, o filtro de percepção representa a entidade básica do mecanismo de percepção. Esse filtro recebe entidades perceptíveis a partir da cena como entrada, extrai informações específicas sobre elas, e, finalmente, decide se envia a percepção referente a essa entidade ao agente ou não. Para diferentes finalidades, foram criados alguns tipos de filtro. Entre eles os que se destacam são:

- **Filtro de intervalo:** Este filtro seleciona somente os objetos que estão fisicamente ao redor do agente em um determinado intervalo de espaço.
- **Filtro de campo de visão:** Este filtro simula um campo de visão de um agente com uma determinada abertura angular.

Table I
OBJECT ATTRIBUTE SUBSCRIPTION EXAMPLES

OBJECT CLASS	PROPERTY	CONDITION	SOURCE	UPDATE POLICY	DESCRIPTION
PHYSICALOBJECT	LOCATION	-	-	FREQUENCY-1	LOCATION ALL OBJECTS AT 1 HZ.
HUMAN	LOCATION	-	-	FREQUENCY-5	LOCATION OF ALL HUMANS AT 5 HZ.
CAR	LOCATION	[COLOR=RED]	-	FREQUENCY-2	LOCATION OF RED CARS AT 2 HZ.
CAR	MODEL	-		ONE-TIME	MODEL OF A CAR
DOOR	STATUS	-		VALUE-CHANGE	DOORS OPENING/CLOSING
HUMAN	GENDER	-	HUMAN21	ONE-TIME	GENDER OF ONE SPECIFIC PERSON

Table II
EVENT SUBSCRIPTION EXAMPLES

EVENT CLASS	PARAMETER	CONDITION	SOURCE	DESCRIPTION
COMMUNICATION	-	-	-	ALL COMMUNICATION EVENTS
COMMUNCIATION	-	-	HUMAN21	COMMUNICATIONS FROM ONE SPECIFIC PERSON
COMMUNICATION	RECEIVER	HUMAN22	HUMAN21	COMMUNICATIONS BETWEEN TWO SPECIFIC SPEECH PARTNERS
PHONERINGING	-	-	CELLPHONE1	RINGING OF A SPECIFIC PHONE

Figura 1: Tabela de inscrições utilizada em [van Oijen and Dignum 2011].

- **Filtro de colisão:** Este filtro detecta os objetos sujeitos a potenciais impactos com o agente de acordo com sua trajetória.
- **Filtro detector de tipo:** Este filtro procura ao redor do agente por objetos com tipos e propriedades específicas.

Também é possível combinar diferentes filtros para atingir o objetivo desejado. Com isso, foi possível alcançar o objetivo inicial e permitir aos agentes perceber os objetos de interesse no ambiente.

Nossa abordagem é baseada nestes dois trabalhos. Da abordagem de [van Oijen and Dignum 2011], utilizamos a idéia de especificar documentos indicando as percepções a serem descartadas. Entretanto, em nosso caso não é adequado a construção de um middleware específico para esta tarefa, já que desejamos incorporar tais filtros ao Jason. Assim, nos inspiramos na abordagem de [Bordeux et al. 1999] para tal, além de também incorporarmos a idéia de tipos de filtros distintos. O detalhamento destas idéias é apresentado na seção 4.

3. Ciclo de Percepção e Raciocínio em Jason

Jason é um interpretador open-source implementado na linguagem Java para uma versão estendida da linguagem AgentSpeak, implementando a semântica operacional da linguagem e fornecendo uma plataforma para o desenvolvimento de sistemas multi agentes.

Em Jason, os agentes são executados por meio de um ciclo de raciocínio que pode ser dividido em 10 passos principais. Durante cada ciclo de raciocínio o agente recebe as percepções do ambiente, as mensagens de outros agentes e executa uma ação. [Bordini et al. 2007b]

O processo interno é ilustrado pela figura 2 se dá da seguinte forma:

1. **Percepção do ambiente:** O primeiro passo na execução do agente é perceber o ambiente. As percepções são um conjunto de literais onde cada literal representa uma propriedade no atual estado do ambiente.
2. **Atualização da Base de Crenças:** Uma vez em posse das percepções, a base de crenças é atualizada para refletir as mudanças do ambiente. Isso é feito através de

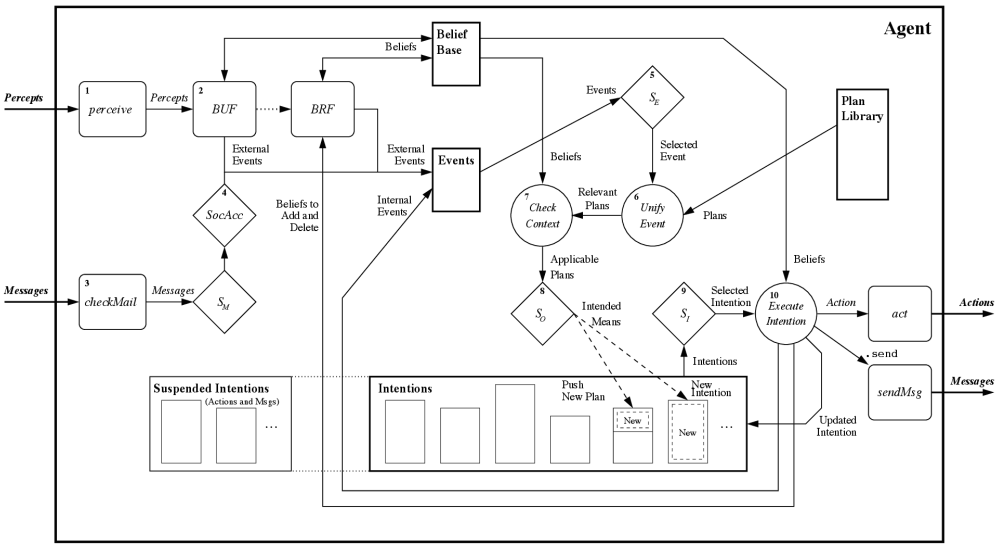


Figura 2: Ciclo de raciocínio de um agente Jason.

uma função de atualização personalizável. Na implementação padrão do Jason, assume-se que tudo o que pode ser percebido pelo agente está listado no conjunto de percepções. Assim, o método padrão percorre a base de crenças e a lista de percepções para incluir os literais novos, remover os literais que não foram mais percebidos e manter aqueles que já eram conhecidos. Cada inclusão ou exclusão de literais gera um evento que é adicionado à uma lista que será utilizada posteriormente.

3. **Recebendo comunicação de outros agentes:** Neste estágio, o interpretador verifica se o agente recebeu alguma mensagem de outro agente. Caso afirmativo, é selecionada uma delas para ser processada.
4. **Selecionando mensagens "Socialmente Aceitáveis":** Antes de processada, a mensagem passa por uma verificação para determinar se ela deve ou não ser aceita pelo agente, verificando o remetente e o conteúdo através de uma função personalizável.
5. **Selecionando um Evento:** Da lista de eventos criada na fase de atualização das crenças, é selecionado um evento para ser executado. Caso não haja alteração da função de seleção, o primeiro evento da lista é escolhido.
6. **Recuperando os planos relevantes:** Selecionado um evento, é necessário encontrar um plano que permita ao agente agir de forma a lidar com o evento. O primeiro passo é percorrer a lista de planos e encontrar aqueles cujo evento seja igual ao selecionado.
7. **Determinando os planos aplicáveis:** Com a lista de planos relevantes, é preciso verificar quais deles podem realmente ser executados, ou seja, cujas pré-condições são válidas.
8. **Selecionando um dos planos aplicáveis:** Neste ponto, já foram determinados todos os planos que podem ser utilizados para lidar com o evento selecionado. Assim, teoricamente, qualquer um dos planos pode ser utilizado para esse fim. Para decidir então qual deles executar é utilizada mais uma função de seleção personalizável que o adiciona à lista existente de intenções.
9. **Selecionando uma intenção para posterior execução:** Durante a execução, ti-

picamente um agente possui mais de uma intenção para executar, cada uma representando um foco de atenção diferente. No entanto, somente uma intenção pode ser executada por vez. Assim, é necessário escolher uma única intenção para ser executada.

10. **Executando um passo da intenção:** Neste último estágio, o agente já atualizou suas informações sobre o ambiente, lidou com um dos eventos gerados e deve agora, efetuar uma ação. Da intenção escolhida, é selecionada a primeira ação que ainda não foi executada e esta é efetuada no ambiente.

Dado que o interpretador Jason é implementado na linguagem Java, é possível utilizar conjuntamente com o AgentSpeak métodos desenvolvidos na linguagem Java para tarefas como por exemplo para cálculos matemáticos ou leituras de arquivos. Utilizando então estes mecanismos, o Jason é capaz de executar os agentes de forma que interajam com o ambiente quando necessário e realizem as tarefas especificadas.

Durante a análise do código-fonte, notamos que um dos pontos críticos de processamento é o segundo passo, onde ocorre a atualização das crenças. Por ter que verificar quais percepções são novas, remover as que não foram mais percebidas e manter aquelas que já eram conhecidas, no pior caso é realizada a comparação de todas as percepções recebidas com todas as que já estavam na base de crenças. Esse processo se torna muito custoso conforme se aumenta a quantidade de percepções. Além disso, cada inclusão ou exclusão na base de crenças gera um evento a ser analisado, fazendo então com que a quantidade de percepções recebidas do ambiente seja um fator ainda mais determinante para o tempo de resposta do agente, já que serão gastos mais ciclos para avaliar eventos que não irão gerar intenções antes de se obter uma resposta para o simulador.

Tendo esses pontos em vista, um filtro de percepções pode afetar diretamente o desempenho dos agentes e reduzir o tempo de processamento de diversas partes do ciclo de raciocínio.

4. Implementando Filtros de Percepção em Jason

Como dito anteriormente, na implementação padrão do Jason, assume-se que tudo o que pode ser percebido no ambiente está listado no conjunto de percepções recebido. Como nem sempre é viável para os agentes perceber todo o ambiente e ainda manter um alto nível de desempenho, é importante reconhecer as mudanças *relevantes* que ocorrem no ambiente.

Os filtros de percepção são então uma solução para esse problema, fazendo com que seja possível selecionar previamente e enviar ao agente somente aquilo que é interessante para ele naquele determinado momento.

Com base nos trabalhos de [van Oijen and Dignum 2011] e [Bordeux et al. 1999] e no funcionamento do Jason, foi idealizado um modo de incluir alguns dos tipos de filtro no mecanismo de percepção do agente Jason.

Um agente Jason recebe as percepções na forma de uma lista de literais em linguagem lógica no passo 1 do seu ciclo de raciocínio. Além disso, o Jason permite a adição de anotações aos literais, de forma que é possível adicionar informações extras às percepções. Por padrão, é adicionada a cada percepção a anotação *source* para que seja

possível saber se uma crença na base de crenças do agente surgiu de uma percepção, do raciocínio do próprio agente ou se foi uma informação vinda de outro agente. As anotações são, assim como as percepções, compostas por um predicado e parâmetros. Assim, um exemplo de percepção seria a seguinte lista:

```
posicao(inseto1,10,10)[source(self)]
posicao(inseto2,5,15)[source(percept)]
posicao(inseto3,18,4)[source(agente2)]
```

Desta forma, possuímos todas essas informações que podem ser utilizadas na hora de decidir se uma percepção deve ou não ser avaliada. De forma semelhante ao realizado com as tabelas em [van Oijen and Dignum 2011], para cada agente são criados documentos de restrição, que informam ao interpretador quais percepções devem ser descartadas. Estes documentos de restrição são construídos em formato xml, onde o exemplo a seguir mostra o documento de restrição que foi utilizado nos experimentos deste artigo:

```
<?xml version="1.0"?>
<PerceptionFilter>
  <filter>
      <predicate>posicao</predicate>
      <parameter operator="EQ" id="0">inseto.*</parameter>
      <parameter operator="GT" id="2">7</parameter>
      <anotation>
        <name>source</name>
        <limit operator="EQ" id="0">percept</limit>
      </anotation>
  </filter>
</PerceptionFilter>
```

Cada arquivo desse pode conter inúmeras tags *filter* para filtrar os diferentes predicados utilizados. Para que uma percepção seja descartada ela deve obedecer a todos os critérios especificados no filtro. Em cada filtro existem três critérios: o predicado, os parâmetros e as anotações, sendo que o único critério obrigatório é o predicado, dado que não é possível que exista um literal sem um predicado associado.

A restrição para o predicado é direta, é verificado se o predicado associado ao literal é o mesmo predicado especificado no filtro. Já para os parâmetros, existem duas informações necessárias, a primeira é o *id* que informa qual dos parâmetros deve ser comparado e o *operator*. Este último indica como deve ser feita a comparação do valor dentro da tag com o valor no literal vindo da percepção. *Operator* pode assumir cinco valores:

- **EQ:** O valor do parâmetro na percepção deve ser igual ao valor dentro da tag. Este último pode ser uma expressão regular para a linguagem Java.
- **GT:** O valor do parâmetro na percepção deve ser maior do que o valor dentro da tag (inteiro ou de ponto flutuante).
- **GE:** O valor do parâmetro na percepção deve ser maior ou igual ao valor dentro da tag (inteiro ou de ponto flutuante).

- **LT:** O valor do parâmetro na percepção deve ser menor do que o valor dentro da tag (inteiro ou de ponto flutuante).
- **LE:** O valor do parâmetro na percepção deve ser menor ou igual ao valor dentro da tag (inteiro ou de ponto flutuante).
- **NE:** O valor do parâmetro na percepção deve ser diferente valor dentro da tag (inteiro ou de ponto flutuante).

Dado que as anotações também são literais em linguagem lógica, é necessário que sejam comparados seu predicado e seus parâmetros. Assim, o valor dentro da tag *name* é comparado ao predicado da anotação para ver se são iguais. A tag *limit* funciona da mesma forma que a tag *parameter*, comparando os parâmetros contidos na anotação.

Mesmo com a possibilidade de se selecionar exatamente as informações que se deseja restringir, pode ser que dependendo da situação e do objetivo atual do agente, seja interessante que os filtros sejam alterados. Com essa finalidade, foi incluída uma ação interna *change filter* que pode ser incluída como parte de um plano para que o filtro atual seja alterado.

Em qualquer plano pode ser incluída a ação interna:

```
+!carry_to(R)
   <- !take(object,R);
        .change_filter(busca);
        -object(r1);
        !!search(slots).
```

Assim, depois que o agente executou o objetivo de pegar o objeto, o filtro é alterado para que haja o controle das percepções nos ciclos de raciocínio futuros. A palavra "busca"utilizada como parâmetro se refere ao nome do arquivo XML onde se encontra o filtro desejado. Assim, é possível ter diversos filtros para diversas situações.

Enquanto um filtro previamente selecionado não for alterado, sempre que o agente receber uma lista de percepções cada uma delas será analisada e eventualmente apagada caso corresponda aos critérios do filtro ativo. Sabendo qual o filtro a ser utilizado, foi inserido entre os passos 1 e 2 do ciclo de raciocínio o mecanismo de filtragem. Dessa forma, é possível verificar quais percepções recebidas pelo agente no passo 1 devem ser enviadas à função de atualização de crenças no passo 2.

5. Estudo de caso

Para tentar encontrar a resposta para as perguntas propostas na seção 1, foi implementado um ambiente onde um agente Jason deve perseguir, capturar insetos que se movem e levá-los de volta para sua base em um grid, como mostrado na Figura 3.

A execução do simulador é feita em passos: (i) o simulador envia os dados da percepção atual ao agente e espera a ação que ele irá executar no ambiente. (ii) ao receber essa ação ele a executa, atualiza a posição dos insetos (movimentando-os aleatoriamente para os lados, para cima ou ficando parados) e gera um novo conjunto de percepções para enviar ao agente no ciclo seguinte.

Para que houvesse impacto das percepções no funcionamento do agente, foi definido que o agente deve guardar a posição dos insetos no passo anterior para comparar

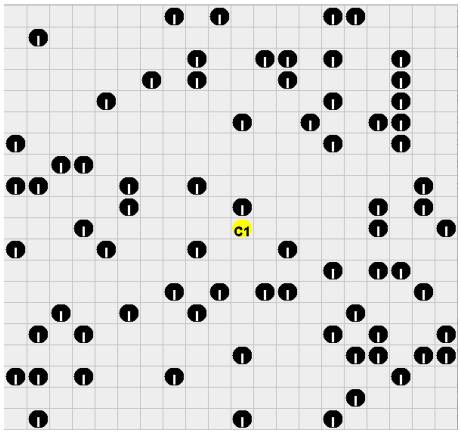


Figura 3: Ambiente utilizado para os testes de desempenho.

com a posição no passo atual. Assim ele se move para o quadrante onde houver a maior quantidade de insetos que se moveu em sua direção. Além disso, foi construído um filtro que ignora todos os insetos que se encontram a uma distância maior que um valor limite pré-definido.

Definimos como n a dimensão do grid, que representa a sua largura e altura (o grid contém n^2 espaços). Os experimentos aqui mostrados apresentam o valor de $n = 20$, permitindo uma variação significativa na quantidade de insetos.

Para os diferentes experimentos, diferentes parâmetros foram utilizados. São eles: (i) O alcance do filtro f, que representa a distância euclidiana máxima que o agente consegue perceber os insetos (ii) O número de insetos i, sendo que $i \leq n^2$.

Experimento	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
i	80	80	80	80	200	200	200	200	280	280	280	280
f	n	7	5	2	n	7	5	2	n	7	5	2

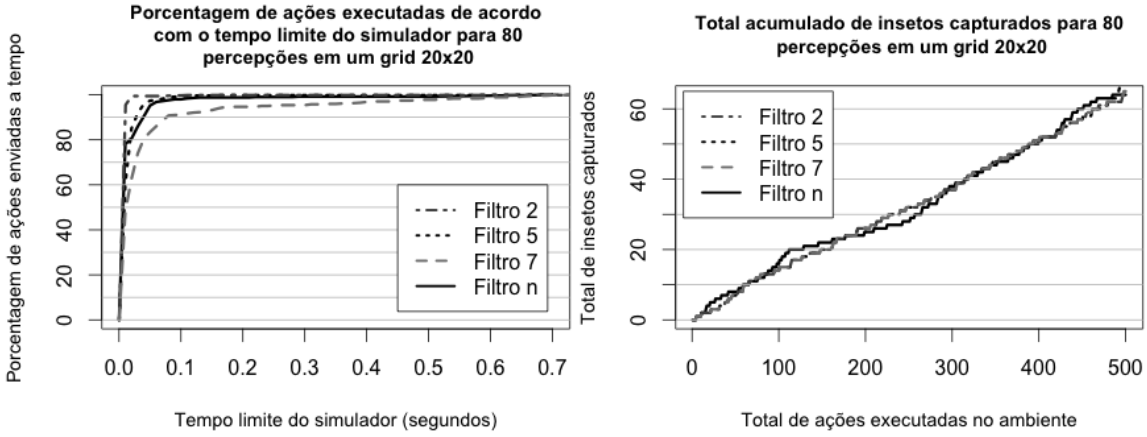
Tabela 1: Experimentos realizados

Foram realizados um total de 12 experimentos, onde se variou os valores desses parâmetros, conforme mostra a Tabela 1. Basicamente, variou-se o número de insetos num grid 20x20, ocupando 20% (80 insetos, experimentos 1 a 4), 50% (200 insetos, experimentos 5 a 8) e 70% (280 insetos, experimentos 9 a 12) do grid. Para cada um destes casos, simulou-se a ação dos agentes sem filtro de percepção (“n”) e com filtros correspondentes a distâncias de 2, 5 e 7 células.

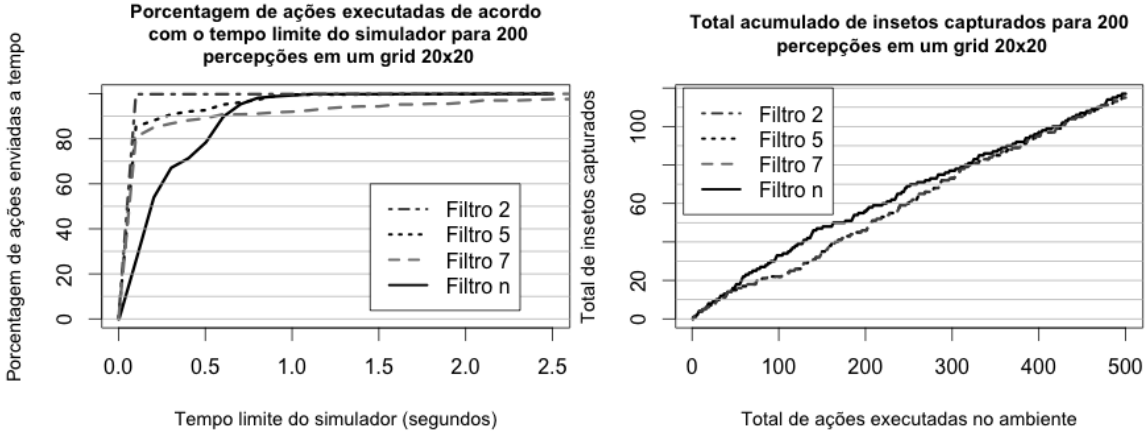
6. Resultados obtidos

Nos gráficos aqui apresentados, foi medida a capacidade do agente responder em um determinado tempo limite. O eixo horizontal marca o tempo limite do simulador em segundos. O eixo vertical mostra quantos por cento das ações foram executadas em tempo menor ou igual ao tempo limite.

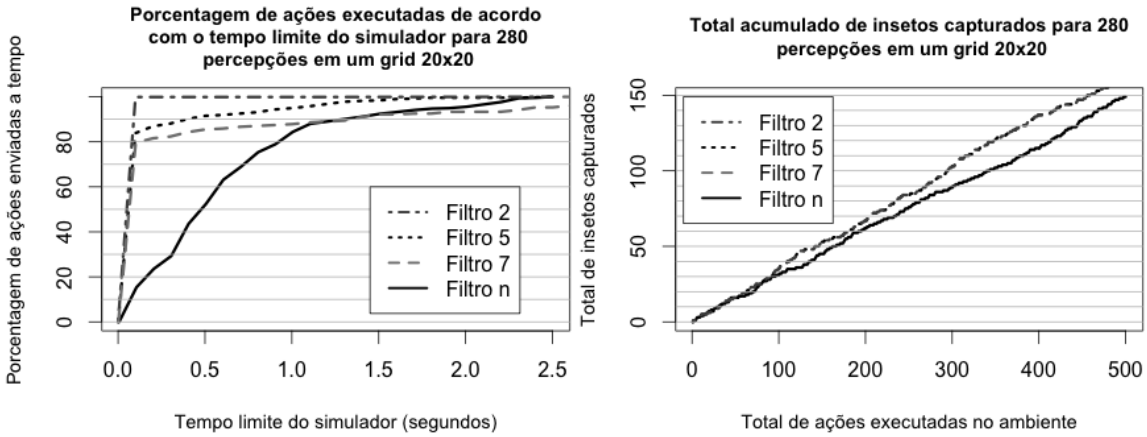
Na figura 4, são apresentados os resultados obtidos nos diversos experimentos. Nas figuras 4a e 4b, mostramos respectivamente o tempo de resposta para deliberar uma



(a) Tempo de resposta - 80 percepções (b) Métrica de desempenho - 80 percepções



(c) Tempo de resposta - 200 percepções (d) Métrica de desempenho - 200 percepções



(e) Tempo de resposta - 280 percepções (f) Métrica de desempenho - 280 percepções

Figura 4: Resultados dos experimentos. Os gráficos à esquerda mostram o desempenho dos agentes em função da quantidade de percepções, os da direita mostram o total de insetos capturados ao longo da simulação.

ação e a métrica de desempenho (número de insetos capturados) para um cenário com 80 insetos, reunindo os experimentos 1, 2, 3 e 4. Analogamente, as figuras 4c e 4d representam o cenário com 200 insetos com os experimentos 5, 6, 7 e 8 e as figuras 4e e 4f os cenários com 280 insetos reunindo os experimentos 9, 10, 11 e 12.

É possível verificar pelos gráficos que conforme se aumenta o número de percepções, a execução sem filtro é a que apresenta a maior perda de desempenho, necessitando de muito mais tempo que as outras para atingir porcentagens mais altas. Vale ressaltar que a escala horizontal do gráfico 4a se encontra reduzida pela metade, para que seja possível visualizar melhor a diferença entre os filtros.

Alguns outros fatos interessantes surgem ao analisar esses gráficos. Conforme esperado, o filtro mais restritivo ($f = 2$) é aquele que sempre necessita de menos tempo para execução e é também o primeiro que atinge a marca de 100%, ou seja, é garantido que todas as ações do agente serão enviadas ao simulador dentro do tempo limite especificado. Já o filtro com $f = 5$ é sempre o segundo a atingir a marca de 100%, mas talvez não seja sempre interessante seu uso, pois em um caso como no gráfico 4a, se o limite de tempo do simulador fosse de 0,01 segundos, o desempenho deste filtro é o mesmo daquele obtido sem filtro, e ainda levando ao agente menos informação. Mais interessante que os casos anteriores, observa-se que o filtro menos restritivo ($f = 7$) em muitos casos tem maior tempo de processamento do que o experimento sem filtro. Isso demonstra que não são todos os casos onde a utilização de um filtro diminui o tempo de processamento. Esse comportamento possivelmente se deve ao fato de que o tempo de processamento no ciclo de raciocínio do agente ganho com a exclusão destas percepções (possivelmente em número não suficiente) não compense o tempo de processamento para realizar esta filtragem.

Ao se analisar os três últimos gráficos, é possível notar que o desempenho do agente (quantidade de capturas) quando se utiliza os filtros é bastante semelhante a quando um filtro não é utilizado. O que demonstra que o agente ainda recebe informações suficientes para fazer boas decisões.

7. Conclusões e Trabalhos Futuros

Retomando as questões levantadas na seção 1, temos agora condições de respondê-las em vista dos experimentos realizados:

(i) Dado um certo limite de tempo de resposta do simulador, é possível para o agente processar todas as informações provenientes das percepções? A partir dos resultados obtidos em nossos experimentos, pode-se ver que dependendo do limite de tempo de resposta do simulador, é possível se prever aproximadamente se o agente terá ou não tempo para processar todas as percepções e executar suas ações. Para tal, basta entrar com o tempo nas abscissas dos gráficos da Figura 4 e verificar se a porcentagem de ações enviadas corresponde a 100%.

(ii) Ao utilizar um filtro de percepções, é possível melhorar o tempo de resposta? Os experimentos mostraram que ao utilizar os filtros de percepção, os agentes obtêm um controle maior sobre o tempo de resposta, reduzindo-o até onde for necessário.

(iii) Caso melhore, como se identificaria o filtro adequado? Os experimentos também tornaram possível identificar, para um dado tempo limite de resposta, quais dos

filtros testados garantem que a porcentagem de ações enviadas corresponda a 100%. Dentre estes, poderia-se escolher aquele menos restritivo, ou seja, que oferece maior informação ao agente.

Deste modo, nossos experimentos permitiram concluir que a quantidade de percepções afeta o tempo de resposta de um agente Jason. Além disto, mostramos que o uso de filtros de percepção contribui para a redução e o controle do tempo de raciocínio de agentes Jason.

Como próximos passos, pretendemos: (i) Medir com maior precisão em quais casos o custo extra de processamento usado para filtrar as percepções é justificado pela redução do tempo usado para a função de atualização de crenças; (ii) Realizar um teste estatístico de significância que permita atestar o efeito exercido sobre a quantidade de percepções no tempo de resposta de um agente Jason; (iii) Mapear o tempo relativo levado pela função de atualização de crenças e pelo filtro comparado com o restante das operações (seleção de mensagens, seleção de eventos, seleção de planos, etc.). (iv) Efetuar testes com outros métodos de filtragem, como por exemplo eliminar percepções de elementos que não gerem eventos para os quais hajam planos ou propriedades do ambiente que não estejam listadas em nenhum plano do agente; (v) Realizar testes em diferentes cenários.

Referências

Bordeux, C., Boulic, R., and Thalmann, D. (1999). An Efficient and Flexible Perception Pipeline for Autonomous Agents. *Computer Graphics Forum*, 18(3):23–30.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007a). *Programming Multi-Agent Systems in AgentSpeak using Jason*. JohnWiley & Sons Ltd.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. J. (2007b). *Programming multi-agent systems in {AgentSpeak} using {Jason}*. Wiley Series in Agent Technology. John Wiley & Sons Ltd, Chichester, UK.

Boulic, R., Bécheiraz, P., Emering, L., and Thalmann, D. (1997). Integration of Motion Control Techniques for Virtual Human and Avatar Real-Time Animation. *Proceedings of ACM VRST*, pages 111–118.

Braubach, L., Lamersdorf, W., and Pokahr, a. (2003). Jadex : Implementing a BDI-Infrastructure for JADE. *EXP in search of innovation*, 3(September):76–85.

Dastani, M., Mol, C., Tinnemeier, N. a. M., and Meyer, J. J. C. (2007). 2APL: A practical agent programming language. *Belgian/Netherlands Artificial Intelligence Conference*, pages 427–428.

Rao, A. S. (1996). {AgentSpeak(L)}: {BDI} agents speak out in a logical computable language. In de Velde, W. V. and Perram, J. W., editors, *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world (MAAMAW'96)*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 42–55, Secaucus, USA. Springer-Verlag.

van Oijen, J. and Dignum, F. (2011). Scalable Perception for BDI-Agents Embodied in Virtual Environments. *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, pages 46–53.

Concepção e análise de um modelo de agente BDI voltado para o planejamento de rota em um VANT

Fernando Rodrigues Santos[1], **Jomi Fred Hübner**[1], **Leandro Buss Becker**[1]

[1]Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brazil

fernando.rod.santos@gmail.com, {jomi.hubner, leandro.becker}@ufsc.br

Resumo. *O uso de agentes em aplicações com Veículos Aéreos Não-Tripulados (VANTs) tem sido explorado nos últimos anos, principalmente como alternativa para dotar o veículo de autonomia em suas missões. Este trabalho tem como objetivo desenvolver um modelo de comportamento autônomo para um VANT com o uso de um agente com arquitetura BDI, explorando sua capacidade de de reagir rapidamente a mudanças em seu ambiente e de ter objetivos de longo prazo. O artigo também discute a implementação do agente no sistema embarcado de um VANT real, além de apresentar uma comparação deste sistema com uma abordagem tradicional de programação.*

1. Introdução

Existem inúmeras possibilidades de aplicações usando Veículo Aéreo Não-Tripulado (VANT), como o monitoramento de regiões por times de VANTs, filmagens de shows, entre outras, como abordado em [Fahlstrom and Gleason 2012]. Em algumas situações, é necessário ou desejável, que o veículo tenha autonomia para realizar algumas tarefas. No contexto deste trabalho, um veículo é considerado autônomo quando ele não precisa da operação constante de um humano e ao qual possa ser atribuído objetivos. O processo de desenvolvimento de um VANT apresenta diversos desafios, como apresentados em [Gonçalves 2014], e a programação da autonomia deve levar em consideração as restrições de hardware, como desempenho, custo e consumo de energia do sistema embarcado do VANT, as quais limitam o uso de técnicas complexas que exijam grande capacidade de processamento da plataforma.

Na área de Sistemas Multi-Agentes (SMA), o tema autonomia é estudado há vários anos, produzindo teorias, arquiteturas de software e linguagens de programação voltadas especificamente para o desenvolvimento deste tipo de software, chamado de agente autônomo, segundo [Wooldridge 2002]. Como vantagens deste tipo de linguagem, vale destacar o seu alto nível de abstração, através de primitivas como objetivos, planos e ações que permitem ao desenvolvedor definir claramente o comportamento e a autonomia do agente. Nesta área, se destacam as linguagens baseadas na arquitetura BDI (Belief, Desire, Intention) que facilitam o desenvolvimento de agentes que apresentam a capacidade de reagir rapidamente a mudanças em seu ambiente e de possuir objetivos de longo prazo, conforme [Rao and Georgeff 1995] e [Bratman 1987].

A robótica móvel de modo geral, não só aplicações com VANTs, trabalha com planejamento de rotas, conforme [Romero et al. 2014], onde o planejador de rotas é o módulo responsável por determinar de forma autônoma o caminho a ser seguido pelo

robô, de acordo com a tarefa atualmente em execução. Dessa forma, busca-se uma solução com o desenvolvimento de um sistema de planejamento de rota para robôs móveis com o uso de agentes, visando explorar as potencialidades da arquitetura BDI nessas aplicações.

Assim, a proposta deste trabalho consiste em analisar a possibilidade do uso de um agente com arquitetura BDI no contexto de um sistema computacional de tempo real embarcado em um robô móvel, um VANT, para aplicações de tomadas de decisões e planejamento de rota. Além disso, é realizada uma comparação dessa abordagem proposta com uma abordagem tradicional de programação orientada a objetos.

No restante deste artigo expomos os trabalhos relacionados, apresentamos o modelo de comportamento autônomo para tomada de decisão de um robô móvel desenvolvido com um agente BDI, bem como uma visão geral do planejamento de rota em robótica móvel, além de um cenário de teste e a modelagem do problema. Também mostramos a sua implementação no sistema embarcado de um VANT e as análises desse sistema em comparação com uma abordagem tradicional de programação, por meio de testes de simulação.

2. Trabalhos Relacionados

O uso de SMA em veículos autônomos não é uma novidade, nota-se que o paradigma de agentes tem sido empregado em projetos com VANTs e que a maioria das abordagens apresentam soluções com agente e SMA. Alguns trabalhos modelam um VANT como um SMA, outros trazem resultados de coordenação em ambiente simulado, enquanto outro realiza testes com um VANT real acoplado a um simulador. Esses trabalhos exploram as potencialidades do SMA para tomada de decisão em grupo de VANTs, e em algumas aplicações utilizam outras técnicas computacionais ou específicas da área do problema.

O trabalho de [Hama 2012] aborda uma aplicação do paradigma da programação orientada a agentes para controle inteligente do comportamento de VANTs, com a concepção de um framework, através do uso da arquitetura, da teoria e de ferramentas orientadas a agentes, que visa prover uma abstração para a programação de comportamentos inteligentes em VANTs. Naquele trabalho é proposto o modelo UAVAS - Unmanned Aerial Vehicles AgentsSpeak, que é um framework de programação de comportamentos para VANTs, que é executado dentro do sistema operacional do hardware embarcado da aeronave. Aquele trabalho utiliza um sistema com múltiplos agentes (SMA) para controlar um único VANT, sendo que ainda enfatiza a comunicação entre os agentes e a cooperação do time. Tem-se o uso de agentes BDI, mas o foco é dado no modelo de abstração. Porém, o framework não foi implementado e testado em uma aplicação embarcada real, somente em simulação.

No trabalho de [Chaves 2013], foram estudados diferentes algoritmos de navegação e padrões de busca adequados, bem como uma visão geral sobre mecanismos de coordenação multi-agente para coordenação distribuída de VANTs, visando cooperação. O projeto propõe um modelo de VANTs cooperativos que combina mecanismos de coordenação multi-agente, algoritmos de navegação para varredura completa de uma região de busca e padrões estabelecidos pelos principais órgãos responsáveis pelas operações de busca e salvamento. Este trabalho explora a cooperação em grupo de VANTs com foco na coordenação do SMA e seus resultados são apenas em nível de simulações em PC, sem implementação em plataformas de VANTs reais, a qual dependerá da capaci-

dade do hardware utilizado para implementar e executar o modelo proposto.

Já o trabalho de [Selecký and Meiser 2012] aborda o uso de equipe com múltiplos VANTs autônomos. Este trabalho utiliza um simulador de VANTs e tráfego aéreo que permite trabalhar tanto com os VANTs virtuais quanto reais em um sistema de realidade mista, no qual VANTs reais e virtuais podem coexistir. O mesmo apresenta algumas soluções para os problemas no processo de integração de VANTs reais em um sistema de simulação multi-agente, que consiste em um ambiente de simulação onde os VANTs podem coexistir, coordenar o seu voo e cooperar em tarefas comuns. Esses VANTs reais são capazes de realizar planejamento on-board e raciocínio, e podem cooperar e coordenar seus movimentos uns com os outros, e também com os VANTs virtuais. Ele explora a parte de integração do VANT de hardware no ambiente simulado, mas não garante que o sistema possa ser utilizado sem modificações em uma aeronave real. Além disso, não investiga ou trata da possibilidade do uso de um agente em um VANT, utilizando uma estrutura do tipo HIL (Hardware in the Loop).

Logo, esses trabalhos não exploram o uso de um único agente BDI em um VANT e as potencialidades da arquitetura, tais como a sua capacidade de reagir rapidamente a mudanças no ambiente e de ter objetivos de longo prazo, buscando assim, dotar um VANT de autonomia em suas aplicações. Da mesma forma, não analisam a possibilidade da aplicação real da abordagem com agente BDI no sistema computacional embarcado de um VANT, sendo este o foco do presente trabalho.

3. Modelo de Planejamento de Rota

Esta seção apresenta uma visão geral do planejamento de rota em robótica móvel, além do modelo proposto com agente BDI. Também mostra o cenário de teste e a modelagem do problema, bem como a implementação e aplicação prática do modelo.

3.1. Visão Geral do Planejamento de Rota

Geralmente, a estrutura de controle de navegação de um Robô Móvel Autônomo (RMA) é organizada em cascata [Romero et al. 2014], com quatro níveis de controle. No nível 4 encontra-se o planejamento dinâmico e a geração das rotas do robô. No nível 3 são executados tipicamente os algoritmos de controle que são responsáveis pela condução do veículo, através do seguimento ou rastreamento de trajetórias, baseados em modelos cinemáticos ou em equações de movimento translacional, os quais garantem o movimento desejado para o robô. No nível 2 é executado o controle da dinâmica do robô, com o objetivo é manter as velocidades longitudinal e lateral do veículo, ou a rotação do robô, e suas derivadas, estabilizadas em torno de um ponto de equilíbrio, que mantém a estabilidade do movimento do veículo. No nível 1 são implementados os sistemas de controle dos sensores e atuadores, garantindo assim as percepções e atuações do robô no ambiente. A figura 1 ilustra o exemplo da estrutura de controle em cascata de um RMA.

O planejamento de rota consiste em dadas as configurações inicial e final do robô, descobrir uma sequência de movimentos a ser executada para que ele saia da primeira e atinja a segunda. Assim, a solução proposta neste artigo trabalha com o enfoque no nível 4 da estrutura de controle de navegação, para o desenvolvimento de um modelo de comportamento autônomo com um agente BDI para planejamento de rota e tomada de decisão em robôs móveis.

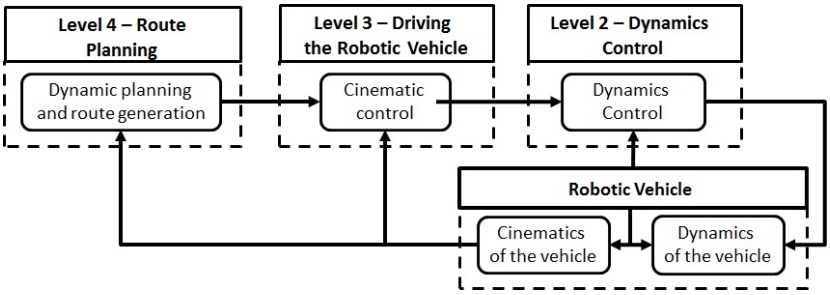


Figure 1. Estrutura de controle em cascata, adaptado de [Romero et al. 2014].

3.2. Modelo Proposto

O sistema proposto é um modelo de comportamento do VANT baseado no modelo de fluxo e conversão de dados para interação do agente com o ambiente proposto em [Hama 2012], porém com a utilização de um único agente BDI no VANT e não um SMA. A arquitetura do sistema é inspirada em [Chaves 2013], com dois níveis: um nível para os controles de estabilidade e de translação da aeronave; e outro nível para o planejamento de rota. Já a parte de testes, inspira-se em [Selecký and Meiser 2012] e seu modelo de integração, mas em vez da integração do VANT de hardware, fez-se uma integração da plataforma embarcada da aeronave com o modelo computacional do VANT.

A solução proposta com agente BDI trata do nível de **Planejamento** da estrutura do controle de navegação. Os controles da dinâmica e da condução do veículo foram desenvolvidos e implementados pelo projeto ProVant[1], sendo utilizados como parte do sistema proposto. Assim, os níveis 1, 2 e 3 foram encapsulados em um único nível, chamado de **Controle**, que executa as funcionalidades desses níveis, tendo uma nova estrutura como ilustrada na figura 2.

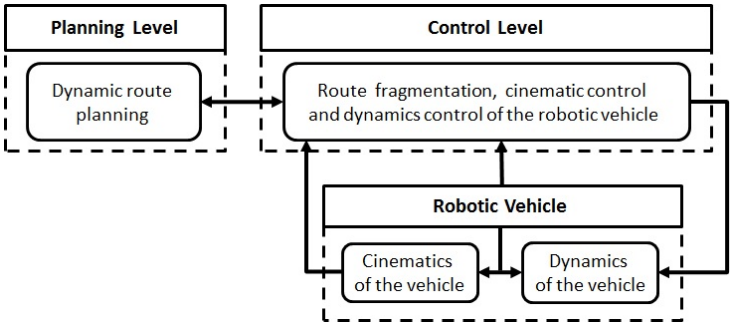


Figure 2. Estrutura proposta.

O fluxo de informação entre os níveis ocorre com os sinais dos sensores do veículo robótico indo do nível de **Controle** para o nível de **Planejamento**, fornecendo as informações das percepções do ambiente requeridas pelo agente, conforme a aplicação. Com base nesses dados, o nível de **Planejamento** realiza o seu processo de tomada de decisão, escolhendo um plano de ação com o ponto e a rota, além de definir as atuações a serem executadas no ambiente, enviando para o nível de **Controle** os pontos de destino (*waypoints*) da aeronave. Assim, o controle executa o algoritmo de translação e aciona os atuadores do sistema.

[1]Site: http://provant.das.ufsc.br

Uma interface de comunicação entre os dois níveis garante a troca de informações. O nível de **Controle** recebe as mensagens de atuação do nível de **Planejamento** com os valores de referência de velocidade e da posição de destino, e envia as percepções com as informações da posição atual (GPS), valores da carga e da taxa de consumo da bateria. A partir de uma nova mensagem do nível de **Planejamento**, ocorre a fragmentação da rota em pontos intermediários entre a posição atual do VANT e o *waypoint* de destino, através de um percurso retilíneo dividido em trechos menores com passo fixo. Também acontece a correção do ângulo de direção da aeronave em relação ao ponto a ser alcançado. Dessa forma, as referências de posição e ângulo são passadas para o nível de **Controle**, garantindo assim o seguimento da rota definida no nível de **Planejamento**.

No nível de **Controle** estão implementados os algoritmos de controle de estabilidade e de controle de translação do VANT, conforme detalhado em [Donadel et al. 2014]. Neste nível, são executadas as rotinas do controle translacional e do controle rotacional, com base nas velocidades em cada direção do eixo carteziano (x,y,z) e os respectivos valores de referências adotados. Além disso, também é realizada uma operação de transformação dos sinais de saída para as grandezas de medidas compatíveis com os atuadores do VANT, sendo estes, sinais de força e ângulo.

3.3. Cenário de Teste

O cenário escolhido para testar o sistema consiste em uma aplicação para visitar os nodos de uma rede de sensores sem fio com um VANT. Os sensores se encontram distribuídos numa região de monitoramento, a aeronave deve visitar todos os nodos da rede, a partir da estação-base, e retornar para a base finalizando a missão. A posição de cada sensor é conhecida pelo VANT desde o início da missão. A medida que a aeronave alcança um *waypoint*, ela tenta obter os dados daquele sensor e, em seguida, passa para o próximo nodo da rede.

A aeronave deve manter uma altitude de cruzeiro, levar em consideração a carga e a taxa de consumo de bateria do VANT, sendo que a taxa varia de acordo com a velocidade e direção do vento. A medida que a aeronave se desloca, mantendo uma velocidade constante de voo, a carga da bateria decrementa, conforme a direção do VANT e do vento. Se o vento está em sentido contrário à aeronave, a taxa de consumo de bateria é maior e com vento a favor o consumo é menor.

O VANT deve analisar também o "ponto de não retorno"[2], levando em consideração a carga e taxa de consumo da bateira, bem como a distância até a base. Quando o VANT perceber que não conseguirá visitar todos os nodos da rede devido a carga de bateria, ele deve abortar a missão de coleta de dados e retornar à estação-base para recarrega. Além disso, durante o retorno à base, caso o consumo de bateria seja alto e não seja possível chegar com segurança, o VANT deve realizar um pouso de emergência.

3.4. Modelagem

Com base no cenário de aplicação, foi realizada a modelagem do problema para ambas as abordagens, visando a comparação e análise do sistema.

[2]O ponto de não retorno é definido como aquele trecho da rota de um voo em que o combustível restante na aeronave não é suficiente para que ela retorne ao aeroporto de partida, em caso de emergência.

Para a abordagem com agentes, foi utilizado o método Prometheus AEOlus de [Uez 2013] para desenvolvimento da aplicação com um agente BDI, o qual tem como objetivo permitir o projeto e a análise de um sistema com agentes.

A modelagem do sistema foi feita com o Diagrama de Objetivos do Sistema, que permite a análise dos objetivos em forma de árvore de decomposição AND/OR, como ilustrado na figura 3. Esse diagrama descreve os objetivos do sistema durante a sua execução. Também foi feito o Diagrama de Visão Geral do Sistema, que permite a visualização da estrutura global do sistema e mostra além do agente, suas percepções e ações, bem como suas crenças iniciais, conforme a figura 4. Assim, tem-se o agente ***VANT*** com as percepções de posição, carga e taxa de consumo da bateria, bem como as atuções com a velocidade e a posição de destino da aeronave, além das crenças.

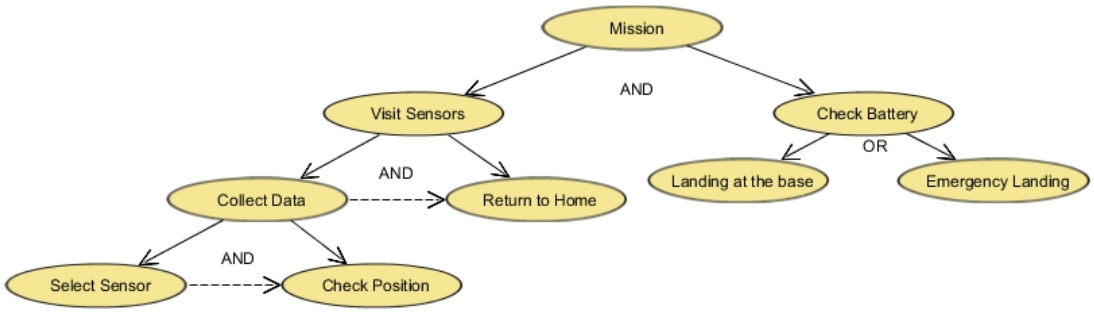


Figure 3. Diagrama de Objetivos do Sistema.

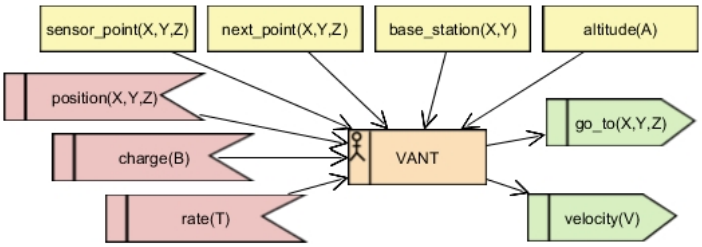


Figure 4. Diagrama Visão Geral do Sistema.

O agente possui como crenças a posição de todos os nodos sensores que precisa visitar (`sensor_point(X,Y,Z)`), a informação de qual será o próximo ponto a ser vistidado (`next_point(X,Y,Z)`), a localização da estação-base (`base_station(X,Y)`), além da altitude de cruzeiro (`altitude(A)`).

A modelagem do comportamento da aeronave para abordagem tradicional, foi realizada com a máquina de estados da figura 5, onde cada estado representa uma situação na qual se encontra o VANT em determinado momento da execução do sistema e as trasições correspondem aos eventos que acarretam em uma mudança de estado.

3.5. Implementação do Planejamento de Rota

O nível de **Planejamento** realiza o processo de planejamento de rota e de tomada de decisão do VANT. Neste módulo foi desenvolvido uma versão com o uso de um agente com arquitetura BDI e, para comparação, outra versão com programação orientada a objetos.

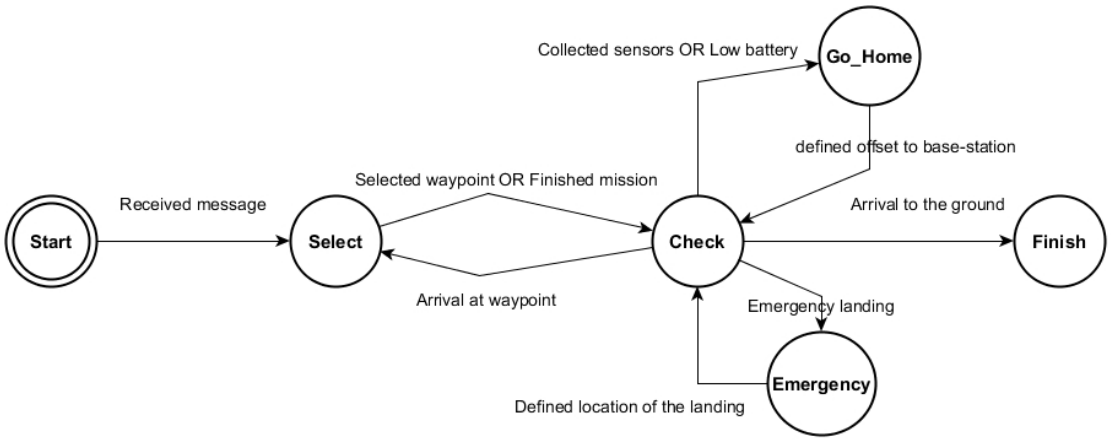


Figure 5. Máquina de Estados.

A versão com agente BDI foi implementada em linguagem Jason[3], tendo uma arquitetura customizada que permite implementar as percepções e as atuações no ambiente. Além disso, foi necessária uma infraestrutura de integração para que o código do agente possa acessar o código do sistema embarcado do robô móvel, viabilizando a integração entre o agente e seu ambiente. Enquanto o mecanismo recebe e mantém as informações mais recentes do ambiente, estas ficam disponíveis para o agente no momento em que ele decide realizar a percepção. Além disso, as informações de atuação do agente permanecem na interface até que uma nova decisão seja tomada para atuar no ambiente.

Um agente pode ser desenvolvido com dois estilos de programação, reativo e proativo. Nesta proposta, os planos em que o agente deve reagir rapidamente a mudanças no ambiente foram implementados de forma reativa, que ocorrem a cada nova percepção do ambiente ou alteração na sua base de crenças e intenções. Por exemplo, o plano de verificação da carga da bateria (programa 1) é executado quando o agente percebe a mudança no valor da carga da bateria (`+charge`), não possui energia suficiente para prosseguir a missão (`not energy`) e não possui o desejo de ir para base (`.desire(go_home)`). Assim, esse plano descarta todos os eventos e intenções do agente (`.drop_all_desires`) e cria a intenção (`!go_home`) de retorno para base.

Já os planos em que o agente tem objetivos de longo prazo e necessite de comprometimento com um plano até que ele seja finalizado foram implementados de forma proativa. Por exemplo, o plano de coletar dados dos sensores da rede (programa 2), baseado na especificação da figura 3 é executado quando o agente tiver essa intenção (`!collect_data`) e possuir sensores (`sensor_point(_,_,_)`) em sua base de crenças. Esse plano ativa outros planos através das intenções de seleção do próximo nodo sensor (`!select_sensor`) e da verificação da chegada ao local de destino (`!check_position`). Após atingir o *waypoint* alvo, o agente consulta o próximo ponto (`?next_point(X,Y,Z)`) que desejava alcançar e remove a informação do sensor (`-sensor_point(ID,X,Y)`) dessa posição. Então, retoma-se novamente a execução do plano com a ativação da intenção (`!collect_data`). Esse processo se repete até que todos os sensores da base de crenças do agente sejam visitados.

[3][Bordini et al. 2007] e Site: http://jason.sourceforge.net/wp/

Programa 1. Plano para verificação da carga da bateria (reativo).

```
+charge(_) : not energy & not .desire(go_home)
    <- .drop_all_desires;
       !!go_home.
```

Programa 2. Plano para coletar dados dos sensores da rede (proativo).

```
+!collect_data : sensor_point(_, _, _)
    <- !select_sensor;
       !check_position;
       ?next_point(X,Y,Z);
       -sensor_point(ID,X,Y);
       !collect_data.

+!collect_data.
```

Outra versão do nível de **Planejamento** foi desenvolvida utilizando uma abordagem tradicional, com programação orientada a objetos e implementada em linguagem C++. Esse sistema opera em um laço de repetição, com uma estrutura de seleção para execução de acordo com o estado do VANT. Neste laço, inicialmente se processa as mensagens com os valores de percepção, e então se realiza o processo de tomada de decisão e envio de mensagem com a atuação, segundo a especificação da figura 5. A tomada de decisão é feita por meio do cálculo das distâncias entre a aeronave e o ponto de destino.

4. Aplicação Prática

O modelo proposto foi implementado no sistema embarcado do VANT do projeto ProVant[4] para o cenário de aplicação descrito anteriormente. Esse veículo possui duas plataformas computacionais embarcadas que se comunicam pela porta serial, sendo uma de baixo nível (*discovery*) para acionamento dos sensores e atuadores, e outra de alto nível (*beaglebone*) para implementação do controle e planejamento da aeronave.

O sistema foi testado diretamente na plataforma de alto nível, a qual será embarcada na aeronave para execução dos níveis de **Planejamento** e **Controle**. Assim, para substituir o VANT real nos testes, foi utilizado um modelo computacional que possui o modelo matemático comportamental do VANT e representa a aeronave real da aplicação, fornecendo os dados dos sensores e sendo alimentado com as informações para os atuadores da aeronave, além das funções da plataforma de baixo nível. Este modelo computacional foi desenvolvido e implementado pelo projeto ProVant, em Matlab/Simulink, sendo este apenas utilizado como parte integrante do modelo final do sistema proposto.

Dessa forma, o sistema proposto foi implementado em um modelo de simulação Hardware In the Loop (HIL)[5] para a realização de testes e análises. A estrutura geral do modelo HIL é composta por uma plataforma embarcada com os níveis de **Planejamento** e de **Controle** que se comunicam via UDP, além de um computador com o modelo computacional da planta (VANT) e um mecanismo de comunicação entre as plataformas, por meio de um link TCP/IP. Tendo as respectivas estruturas para as abordagens com agente e tradicional, conforme ilustrado na figura 6.

[4]Site: http://provant.das.ufsc.br

[5]A simulação Hardware-in-the-loop (HIL) é descrita como a técnica aplicada ao desenvolvimento, teste e validação de sistemas de tempo real embarcados complexos. Este ambiente proporciona uma plataforma efetiva permitindo a adição de complexidade, com a segurança da plataforma de testes, conforme [Louall et al. 2011].

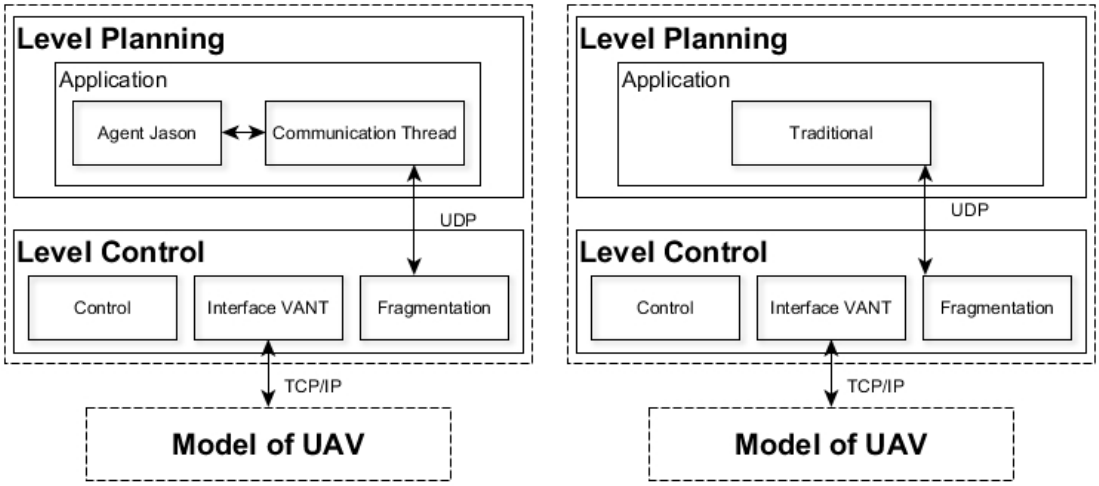


Figure 6. Estrutura HIL: abordagens com agente e tradicional.

Os ensaios com o modelo HIL realizados visaram testar o sistema proposto com agente BDI em uma plataforma embarcada. Dessa forma, não foram adicionados ruídos ao processo para testar estabilidade e robustez do sistema de controle.

5. Análises e Resultados

Como resultado deste trabalho, obteve-se duas versões do sistema de tomada de decisões e planejamento de rota de um robô autônomo. Esses sistemas foram implementados no sistema embarcado de um VANT e testados através de um modelo de simulação HIL que descreve o comportamento da aeronave.

A missão do VANT nas simulações foi de percorrer a área onde estão os nodos da rede de sensores sem fio, visitar cada um deles e retornar para estação-base, levando em consideração as variações do ambiente durante a sua execução.

5.1. Análise Qualitativa

A análise qualitativa busca fazer uma análise a nível de projeto e desenvolvimento do sistema com uso de uma linguagem de agentes em uma aplicação de sistemas embarcados, a qual geralmente se utiliza uma abordagem tradicional de programação e tem-se restrições de hardware e processamento.

5.1.1. Análises Iniciais

O desenvolvimento do agente BDI foi inicialmente baseado na implementação da abordagem tradicional, com a criação de flags de estado como crenças do agente. Porém, após a familiarização com a linguagem, foram realizadas melhorias e refinamentos no código, sendo este processo bastante custoso com relação ao tempo de projeto, até se chegar a uma implementação baseada em objetivos e não em estados.

Percebeu-se que o uso de uma linguagem de programação de agentes BDI facilita o desenvolvimento de agentes que necessitem de capacidade de reagir rapidamente a

mudanças no ambiente (como exemplifica o programa 1), bem como possuir objetivos de longo prazo em uma aplicação (como exemplifica o programa 2), conforme empregado na implementação dos planos do agente no sistema. Já uma abordagem tradicional apresenta dificuldade de se definir comportamentos proativos na modelagem em máquina de estados, além disso, o desenvolvimento desse tipo de comportamento requer a implementação de processos concorrentes, levando a uma programação mais complexa. Por outro lado, máquinas de estados são amplamente conhecidas e possuem várias ferramentas de análise, além de serem rapidamente e eficientemente implementadas.

Nota-se também que no uso de uma linguagem de programação de agentes, como Jason, muitas vezes, tem-se a necessidade do desenvolvimento de uma infraestrutura de integração para aplicações embarcadas. Visto que estas aplicações são desenvolvidas, geralmente, em linguagem C.

5.1.2. Diferença de programação

Analisamos a diferença de programação entre as abordagens para se inserir um novo comportamento ou funcionalidade no sistema. Na abordagem com agentes BDI, a inserção de um novo comportamento ou alteração de uma funcionalidade na aplicação é realizada através da criação de um novo plano, sem mexer no que já existe no código, como pode ser notado pela independência entre os códigos dos programas 1 e 2.

Já na abordagem tradicional de programação, desenvolvida com uma estrutura de seleção baseada nos estados do sistema, a inserção de um novo comportamento ou alteração de uma funcionalidade na aplicação é realizada através da criação de uma nova condição na execução do sistema como um todo. Além disso, deve-se verificar a lógica para que não haja conflito entre as demais condições para que não aconteçam simultaneamente. Logo, por serem paradigmas diferentes, apresentam características específicas que interferem no modo como programar o comportamento desejado para a aplicação.

5.1.3. Modelagem

Na análise da modelagem do problema, tem-se que a área de SMA não possui um método padrão para desenvolvimento de sistemas, nem um ferramental de desenvolvimento tão consolidado como para a abordagem tradicional. Além disso, cada abordagem modela o comportamento do sistema de uma forma particular, sendo que a Máquina de Estados especifica a sequência de estados pelos quais um processo passa durante o tempo de execução em resposta aos eventos, enquanto o Diagrama de Objetivos do Sistema descreve os objetivos do sistema durante a execução do processo.

Comportamentos sequenciais podem ser modelados tanto em uma como em outra abordagem, porém, os comportamentos proativos são mais difíceis de serem modelados em Máquinas de Estados. Enquanto comportamentos complexos e concorrentes podem ser modelados com o Diagrama de Objetivos do Prometheus AEOlus.

5.2. Análise Quantitativa

A análise quantitativa busca mensurar o desempenho das versões do sistema, visando comparar seus rendimentos em uma aplicação real. Duas formas de análise foram uti-

lizadas para avaliar os sistemas desenvolvidos: o tempo de utilização da CPU e o tamanho do código fonte gerado por cada abordagem. Para a análise do uso do processador, foram realizadas simulações do modelo HIL com as versões do sistema proposto.

5.2.1. Utilização da CPU

O tempo de utilização do processador pelo sistema de tomada de decisão no nível de **Planejamento** foi medido através do comando *time* do sistema operacional Linux. Esse comando informa o tempo de execução de um processo específico. Assim, dado o tempo total de execução da simulação, com o VANT percorrendo todos os sensores e retornando para estação-base, mediu-se qual a fatia de tempo que o processo de planejamento faz uso do processador para executar as suas tarefas. Com base no tempo total e no tempo do processo, obtem-se um percentual de uso da CPU pelo processo.

Os primeiros testes, foram realizados utilizando a comunicação com o envio da percepção para o nível de **Planejamento** a cada mudança na posição do VANT. Com isso, o agente sempre tinha uma nova informação para atualizar em sua base de crenças quando realizava o processo de percepção no seu ciclo de raciocínio. A abordagem com agente BDI teve um tempo de utilização da CPU de 61% e a abordagem tradicional teve um tempo 4%. Percebeu-se que o agente utilizava muito do seu tempo para fazer percepção, já que a frequência de atualização das informações é alta (a cada 5 ms).

Nos testes seguintes, foi utilizada a comunicação com o envio da percepção para o nível de **Planejamento** somente após uma mudança significativa na posição do VANT, sendo essa maior que 0,1m. Com isso, obteve-se uma redução do tempo de uso da CPU, tanto com o agente como o tradicional. Foram realizados experimentos com 10 amostras, os resultados do agente Jason obteve Média = 15,158 % e Desvio Padrão = 1,126. Já a abordagem tradicional obteve Média = 0,833 % e Desvio Padrão = 0,230. Assim, a solução com agente obteve um desempenho satisfatório, o que possibilita a sua execução numa aplicação em plataforma embarcada sem comprometer o desempenho do hardware.

5.2.2. Tamanho do Código

Outro ponto analisado foi o tamanho do código fonte, considerando somente a parte do código referente ao nível de planejamento do sistema, para cada uma das abordagens utilizadas.

Como a quantidade de linhas no código fonte de um programa depende do estilo de programação do desenvolvedor, optou-se por utilizar o comando *gzip* para compactar o arquivo fonte (sem comentários) dos programas e verificar o tamanho do mesmo em bytes. Além disso, também realizou-se a contagem do número de identificadores utilizados no código fonte de cada programa.

O código do agente Jason possui 0,700 Kbytes e 81 identificadores, enquanto o código da abordagem tradicional tem 2,164 Kbytes e 165 identificadores. Por ter um nível de abstração maior e utilizar a programação lógica, o código do agente é menor que o da versão tradicional.

6. Considerações Finais

Conclui-se que é possível a execução de um agente BDI embarcado em um VANT, tendo em vista a redução do percentual de uso da CPU apresentado nas análises e testes através de um modelo de simulação HIL. Embora necessite de uma infraestrutura para integração com o sistema embarcado do robô móvel, a solução com agente é viável para aplicações de planejamento de rota e pode ser embarcada em um robô autônomo.

O trabalho trouxe como contribuição um modelo de comportamento para planejamento de rota de um VANT autônomo com uma abordagem de agente BDI, além da integração do sistema na plataforma computacional embarcada da aeronave. Também apresentou algumas potencialidades da arquitetura BDI em VANTs, analisando algumas vantagens e desvantagens do uso desse tipo de agente nessas aplicações, podendo essa solução ser utilizada e explorada em outros cenários da robótica móvel, tais como veículos terrestres, aquáticos, entre outros.

Como trabalhos futuros, pretende-se extender a aplicação para o cenário mais completo de coleta de dados em uma rede de sensores sem fio, além da realização de testes do modelo proposto com o VANT real.

References

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. Wiley-Interscience.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Harvard University Press.

Chaves, q. N. (2013). Proposta de modelo de veículos aéreos não tripulados (vants) cooperativos aplicados a operações de busca. Master's thesis, USP - Universidade de São Paulo.

Donadel, R., Raffo, G. V., and Becker, L. B. (2014). Modeling and control of a tiltrotor uav for path tracking. *19th World Congress The International Federation of Automatic Control*.

Fahlstrom, P. G. and Gleason, T. J. (2012). *Introduction to UAV Systems*. Wiley.

Gonçalves, F. S. (2014). Projeto da arquitetura de software embarcado de um veículo aéreo não tripulado. Master's thesis, UFSC - Universidade Federal de Santa Catarina.

Hama, M. T. (2012). Uma plataforma orientada a agentes para o desenvolvimento de software em veículos aéreos não-tripulados. Master's thesis, UFRGS - Universidade Federal do Rio Grande do Sul.

Louall, R., Belloula, A., Djouadi, M., and Bouaziz, S. (2011). Real-time characterization of microsoft flight simulator 2004 for integration into hardware in the loop architecture. *Control Automation (MED) - 19th Mediterranean Conference*.

Rao, A. S. and Georgeff, M. P. (1995). Bdi agents: from theory to practice. *Proceedings of the First International Conference on MultiAgent Systems*.

Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D. (2014). *Robótica Móvel*. LTC.

Selecký, M. and Meiser, T. (2012). Integration of autonomous uavs into multi-agent simulation. *Acta Polytechnica*, 52(5).

Uez, D. M. (2013). Método para o desenvolvimento de software orientado a agentes considerando o ambiente e a organização. Master's thesis, UFSC - Universidade Federal de Santa Catarina.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. Chichester: John Wiley and Sons Ltd.

Detecting Normative Indirect Conflicts: dealing with actions and states

Jean de Oliveira Zahn and Viviane Torres da Silva

Computer Science Department – Universidade Federal Fluminense (IC/UFF)
24.210-240 – Niterói – RJ – Brazil

{jzahn, viviane.silva}@ic.uff.br

Abstract. *Norms have been used in Multi-agent Systems to describe what agents can do, must do and cannot do. It intends to regulate the behavior of the autonomous and heterogeneous entities. One of the main issues on the specification of norms is the detection of normative conflicts. Two norms are in conflict when the fulfillment of one norm violates the other and vice-versa. Although several works have been proposed to deal with normative conflicts, the majority focuses on direct normative conflicts that occur when the norms apply to the same entity and govern the same behavior. We propose a mechanism to check indirect conflicts between norms that do not regulate exactly the same behavior but related ones. In order to do so, it is important to analyze the characteristics of the multi-agent system domain while checking for the normative conflicts. In this paper, we extend our preliminary work by identifying the relationships between actions and states, presenting the relationships that can be used to link the states of the multi-agent system and discussing the checking of indirect normative conflicts that can only be found when considering these relationships. The conflict checker is able to find out conflicts between norms that do not regulate the achievement of the same states and conflicts between norms when one regulates the achievement of a state and the other the execution of an action.*

1. Introduction

In open multi-agent systems, norms are being used to regulate the behavior of the involved entities. Norms state what can be performed (permission), cannot be performed (forbidden) and must be performed (obligation) in the system. One of the main challenges on developing normative systems is that norms may conflict with each other. The conflict between two norms arises when the compliance with one norm results on the violation of the other, and vice-versa.

Several approaches propose methods for detecting direct and indirect conflicts. Direct conflicts occur between norms that govern the same behavior executed by the same entity and indirect conflicts are identified when we consider the relationships between the behaviors being regulated and the relationships between the entities. Thus, the detection of indirect conflicts is only possible when we analyze the domain-dependent relationships between the states and actions being regulated and the relationships between the involved entities.

In our preliminary work [Zahn and Silva 2014] we describe the relationships used relate the entities and to relate the actions of a multi-agent system. The conflict

checker algorithm stated in [Zahn and Silva 2014] focus on the identification of indirect conflicts by considering these relationships. It is able to identify conflicts between norms that are not applied to the same entity and between norms that regulate the execution of different actions.

In our current work, we extend such approach by presenting three relationships that can be used to relate states (dependency, composition and orthogonality). We also present the normative conflict checker that considers such relationships and the definition of the actions of the domain when checking for conflicts. The conflict checker is able to check for conflicts between two norms that regulate the achievement of different (but related) states and also conflicts between two norms when one norm regulates the execution of an action and the other the achievement of a state. The normative structure used in our current work extends the structure represented in our previous work [Zahn and Silva 2014]. However, besides to actions, this paper considers states as a behavior regulated by the norm.

The remainder of this paper is divided in 5 sections. Section 2 presents the definition of action and state being used in this paper. Section 3 describes the three relationships between states being considered in this paper. This Section also discusses about the rules to be followed by the conflict checker algorithm that faces two norms that regulate different but related states or that faces two norms when one regulates the execution of an action and the other the achievement of a state. Section 4 presents the algorithm to detect the indirect normative conflicts. Section 5 states some related works and, finally, Section 6 concludes and presents future work.

2. State and Action Definition

In this paper, we follow the definitions presented in [Russell and Norvig 2010] for states and actions. Each state is represented as a conjunction of *fluents* that are ground, functionless atoms. An action is described by defining the precondition for its execution and the effect (or post-condition) of its execution. The precondition of an action represents the state that must be true before the execution of the action. The precondition of an action must hold on the current state in order to be executed. The effect of an action represents the state that will be true if the action is executed. Since most actions leave most things unchanged [Russell and Norvig 2010], the effect of an action defines only what is changed after its execution. Therefore, an action is defined by a state representing its precondition, its name and its effect, as follows:

```
Action Scheme:
        (preconditionState; actionName; effect)
```

Let´s consider for instance an action to load a plane P with a cargo C. The precondition of this action states that the plane must be free and the effect of this action states that the cargo is in the plane.

```
Example:
        (free(p); load(c,p); in(c,p))
```

3. Relationships Between States

The aim of this section is to demonstrate that two norms apparently not in conflict are in fact in conflict when we figure out that the behaviors regulated by these norms are related. In order to give an example, please consider the following norms below. Norm 1 obliges *agent 1* to execute *action A* and norm 2 forbids the same agent to achieve state

S1. In this paper we are assuming that entities whose behaviors are being regulated are the same (or are related entities [Zahn and Silva 2014]), the context where the norms apply are the same and the period during while they are active intersect. Therefore, we are using a very simple template to describe our norms that does only mention the deontic concept (permission, obligation or prohibition) and the behavior being regulated.

```
Norm Template : {deonticConcept, actionState}
        Norm1 = {Obligation, execute A}
        Norm2 = {Prohibition, achieve S1}
```

Since these norms are not regulating exactly the same behavior, one could conclude that they are not in conflict. But, let's now suppose that *action A* is defined as follows:

`(S1, A, S2)` where S1 is the **precondition** for the execution of *A* and S2 is its **effect**.

If *agent1* decides to fulfill the prohibition of *norm2* stating that it cannot achieve state S1, it will never be able to execute *A*. Consequently, agent1 will violate norm1 that obliges it to execute *A*. Since the fulfillment of one norm implies on the violation of the other, we must conclude that these norms are in fact in conflict. Note that this conclusion is only possible because we have information about the domain where the norms are immersed, i.e., we know the definition of *action A* and the dependency between *action A* and state S1. Thus, if these norms are used in another domain that changes the definition of *action A*, it is not possible to affirm before hand that there will be a conflict between the norms.

The next subsections present three relationships between states and between states (or actions) that our approach is able to analyze when checking for conflicts between norms.

3.1. Dependency

The dependency relationship between states and between states and actions is originated from the definition of the actions themselves. As states in Section 2, an action is defined by its precondition and its effect. Since an action can only be executed if its precondition is achieved, there is a dependency between the action and the precondition. Moreover, the effect is only achieved it the action is executed. Therefore, there is a dependency between the effect and the action. In addition, the effect is only achieved if the precondition is achieved (and the action is executed). The definition of an action implicitly states the dependencies between precondition, action and effect that can be explicitly represented as follows: `(client, dependent, dependency)`

```
(precondition, action, dependency)
(action, effect, dependency)
(precondition, effect, dependency)
```

In order to explain the dependency relationship, let's consider the example of loading a plan presented in Section 2. The action *load(c,p)* implicitly defines the dependencies below. The dependence relationships defined implicitly should be used when checking for normative conflicts, as described in the next sections.

```
(free(p), load(c,p), dependency)
(load(c,p), in(c,p), dependency)
(free(p), in(c,p), dependency)
```

3.1.1. Obligation or Permission to Achieve the Dependent State

When a norm obliges (or permits) the achievement of a state, it means that the entity which behavior is being regulated must (or can) execute an action which **effect** is such state. For instance, in order to achieve the state *in(c,p)* it is necessary to execute the action *load(c,p)*. The state *in(c,p)* depends on the action *load(c,p)*.

If there is only one action whose **effect** is the state that must (or can) be achieved, there must not be a norm prohibiting the execution of such action. If the execution of the action is prohibited, this norm will be in conflict with the one that obliges (or permits) the achievement of the state. It is impossible to achieve the state without executing the action.

3.1.2. Obligation or Permission to Execute the Dependent Action

When a norm obliges (or permits) the execution of an action, it means that the **precondition** of such action must be achieved in order to the entity be able to execute the action. For instance, in order to execute the action *load(c,p)* it is necessary to achieve the state *free(p)*. The action *load(c,p)* depends on the state *free(p)*. Thus, if there is a norm prohibiting the achievement of the **precondition**, this norm will be in conflict with the one obliging (or permitting) the execution of the action. In any other case, there will not be a conflict.

3.1.3. Prohibition to Achieve the Dependent State

When a norm prohibits the achievement of a state, it means that the entity whose behavior is being regulated should not execute actions whose **effect** is such state. For instance, if the state *in(c,p)* is prohibited, the entity cannot execute the action *load(c,p)*.

If there is an action, which **effect** is the state being prohibited, there must not be a norm permitting or obligating the execution of such action. Note that such verification must be done for every norm applied to the actions, which **effect** is the state being prohibited. For instance, if there is a norm prohibiting an entity to achieve *in(c,p)*, the conflict arises when another norm permits (or obliges) the execution of an action whose effect is *in(c,p)*.

3.1.4. Prohibition to Execute the Dependent Action

When a norm prohibits the execution of an action, it means that the **effect** of such action will not be achieved if the agent fulfills the norm. If there is a permission or an obligation applied to such **effect** and there is not any action with the same **effect**, the norms are in conflict. If is not possible to achieve the state without violating the prohibition of executing the action.

3.2. Composition

We define this relationship since it is particular important when there is a need to abstract the details of simple behaviors and to group all behavior in a more generic one. This generic behavior aggregates the others that are implicitly represented. For instance, let's suppose that a plane can only fly from a place to another if it is loaded and fueled. We can define the state *ready(p)* being a composition of the states *in(c,p)* and *fueled(p)*.

```
(in(c,p), ready(p), composition)
```

(fueled(c,p), ready(p), composition)

So, the composition between behaviors indicates that there is one behavior that represents the whole (in our example *ready(p)*) and others that represent parts of the whole (in our example *in(c,p)* and *fueled(p)*). In [Silva 2013] this relationship is applied in the scope of actions. In the current paper we apply in the context of states and define the *wholeState* composed of *partStates*. The *wholeState* is achieved when all *partStates* are achieved. In this Section we detail the checking for conflicts between norms applied to different states/actions related by the composition relationship.

3.2.1. Obligation or Permission to Achieve the WholeState

State-State: When a norm obliges (or permits) the achievement of a *wholeState*, it means that all its *partStates* must be achieved in order to fulfill the norm. Therefore, there must not be another norm that prohibits the achievement of any *partState*. If there is a norm prohibiting the achievement of at least one *partState*, this norm will be in conflict with the other that obliges (or permits) the achievement of the *wholeState*. The fulfillment of the prohibition will violate the obligation (or permission) and vice-versa.

State-Action: Let's now consider the problems related to the obligation (or permission) applied to *wholeState* and norms applied to actions. If there is a norm that prohibits the execution of an action which effect is the *wholeState* (or a *partState*), such norm will conflict with the obligation (or permission) of achieving the *wholeState*/*partState* if this action is the only action with effect equal to the *wholeState*/*partState*. If there are other actions, there is a need to check if all other actions are also being prohibited. If the action is not executed by fulfilling the prohibition, the agent will not achieve the *wholeState*/*partState* and the obligation (or permission) of achieving the *wholeState* will be violated. However, if there are other actions that can be executed and that will achieve the *wholeState*/*partState*, there will not be a conflict between these norms.

3.2.2. Prohibition to Achieve WholeState

State-State: When a norm prohibits the achievement of a *wholeState*, it does not prevent the achievement of the *partStates* separately. The violation of the prohibition will only occur if all *partStates* are achieved in the same period of the prohibition. Thus, a conflict between the prohibition applied to the *wholeState* and obligations (or permissions) applied to *partStates* will only occur if all *partStates* are being obliged (or permitted) to be achieved in the same period of the prohibition applied to the *wholeState*. If there is at least one *partState* that is not being obliged to be achieved at the same period of the prohibition, the conflict is not characterized.

State-Action: If there are norms prohibiting the execution of actions, which precondition or effect are *partStates* (or wholeState), these norms will never conflict with the prohibition applied to the *wholeState*. However, on the other hand, if the obligation/permission is applied to an action which precondition is a *partState*, the conflict between such obligation/permission and the prohibition applied to the *wholeState* does not occurs. The entity is prohibited to achieve the *wholeState* but is not prohibited to achieve the *partStates* insolated. The entity will be able to fulfill the prohibition of achieving the *wholeState* and the obligation/permission of execution the action. For instance, if there is a norm permitting/obliging the execution of *unload(c,p)* and another prohibiting the achievement of *ready(p)*, these norms are not in conflict. In

order to execute *unload(c,p)* the precondition *in(c,p)* must be achieved but it does not mean that *read(p)* is achieved *ready(p)* is achieved only if *in(c,p)* and *fueled(p)* are achieved.

3.3. Orthogonality

There are states that can be achieved at the same time by the same entity. For instance, *in(c,p)* and *fueled(p)* can be achieved at the same time since the actions to *load(c,p)* and *fuel(p)* can be executed at the same time. However, there are states that cannot be achieved at the same time by the same entity because they are orthogonal. For instance, *flying(p)* and *taxing(p)* cannot be achieved at the same time. The plane cannot be flying and taxing at the same time. In this Section, we focus on the checking of conflicts between norms applied to different states/actions related by the orthogonal relationships.

```
       flying(p), taxing(p), orthogonal)
Actions definitions:
       (flying(p), land(p), taxing(p))
       (taxing(p), takingOff(p), flying(p))
```

3.3.1. Obligation or Permission to Achieve State

State-State: If there is a norm obligating the achievement of a state and another obligating (or permitting) the achievement of another state that is orthogonal to the previous one, these norms are in conflict. Both states cannot be achieved by the same entity at the same time. For example, if there are obligations to achieve *flying(p)* and to achieve *taxing(p)*, these norms are in conflict.

State-Action: If there is a norm obligating the achievement of a state and another obligating (or permitting) the execution of an action which **effect** is a state orthogonal to the previous one, these norms are in conflict. Since both states cannot be achieved by the same entity at the same time, the entity cannot fulfill the obligation to achieve the state and to execute the action. For instance, if there is a norm obligating the execution of *land(p)* which **effect** is *taxing(p)* and another norm obligating the achievement of *flying(p)*, there norms are in conflict.

3.3.2. Prohibition to Achieve State

State-State: If there is a prohibition to an entity to achieve a state and a permission (or an obligation) to this entity to achieve another state that is orthogonal to the first one, there is not a conflict. The entity can achieve the state being permitted (or obliged) without violating the prohibition since they are orthogonal. On the other hand, if there is a prohibition to achieve a state and another prohibition to achieve another state that is orthogonal to the first one, there is a potential inconsistence if the only two states available to the entity are the ones being prohibited. For instance, let's suppose that a plane is flying or taxing. Therefore, if there is a norm prohibiting a plane to achieve *flying(p)* and another prohibiting the plane to achieve *taxing(p)*, these two norms are in conflict. Note that we are not dealing with such restriction since it would be necessary to know all possible states of the system.

State-Action: If there is a norm prohibition the achievement of a state and another obligating (or permitting) the execution of an action which **effect** is a state orthogonal to the previous one, these is not a conflict. However, if there is a norm prohibiting the

achievement of a state and another prohibiting the execution of an action which effect is a state orthogonal to the first one, there is a potential conflict since these are the only states that can be achieve. A similar case was discussed previously.

4. Conflict Checker

The conflict checker algorithm checks for conflicts by considering normative pairs. The algorithm that is based on the relationships between states described in Section 3 is divided in three small algorithms, each one associated with one relationship kind.

Algorithm 1 Verifying Dependency Relationship

```
Require: n1 and n2 as parameter
Function: dependencyRelationship(n1, n2)
if ((checkStateAndActionRelationship(n1.as, n2.as) = dependency)) then
  if (( (n1.deoC = O or n1.deoC = P) and n2.deoC = F) and
     (n1.as is 'state' and n2.as is 'action')) then
       if (selectActionsHaveEffect(n1.as)=n1.as) then
          return true
       else if (checkAllBehavior(selectActionsHaveEffect(n1.as), F)) then
              return true
            endif
       endif
  endif
  if (( (n1.deoC = O or n1.deoC = P) and n2.deoC = F) and
     (n1.as is 'action' and n2.as is 'state') and
     (n1.as ∈ selectActionsHavePrecondition(n2.as)) then
       return true
  endif
  if ((n1.deoC = F and n1.as is 'state') and
      checkAnyBehavior(selectActionsHaveEffect(n1.as), F)) then
       return true
  endif
  if ((n1.deoC = F and (n2.deoC = P or n2.deoC = O) and
     (n1.as is 'action' and n2.as is 'state')) then
     if (selectActionsHaveEffect(n2.as) = n1.as) then
        return true
     else if (checkAllBehavior(selectActionsHaveEffect(n1.as), F)) then
            return true
          endif
     endif
  endif
endif
return false
end function
```

Figure 1. Verifying dependency relationship.

Algorithm 1 is responsible to check if the behaviors being regulated by the two norms are related by the dependency relationship (by executing *checkStateAndActionRelationship* function) and, if it is the case, if the norms are in conflict or not. The algorithm follows the strategies described in Section 3.1:

- (Sections 3.1.1) If one norm is obliging (or permitting) the achievement of a state and the other the execution of an action, there is a potential conflict. If the state is the effect of the action and there is not any other action with such **effect**, there is a conflict. In addition, if there is other actions with the same **effect** but all of them are being prohibited, there is also a conflict.
- (Sections 3.1.2) If one norm is obligating (or permitting) the execution of an action and the other is prohibiting the achievement of a state that is the **precondition** of such action, the norms will be in conflict.

- (Section 3.1.3) If one norm is prohibiting the achievement of a state, it is necessary to check if any action which **effect** is the state is being permitted or obligated. If it is the case, the norms will be in conflict.
- (Section 3.1.4) If one norm is prohibiting the execution of an action and the other is permitting or obligating the achievement of a state, it is important to check if there are other actions with the same **effect**. If not, the norms are in conflict. If there are other actions, we must check if they are also being prohibited. In this case, there is a conflict.

Algorithm 2 Verifying Composition Relationship

```
Require: n1 and n2 as parameter
Function: compositionRelationship(n1, n2)
if (checkStateRelationship(n1.as, n2.as) = composition) then
  if ((n1.deoC = P or n1.deoC = O) and (n2.deoC = F)) and
     n1.as is 'wholeState' ) then
        return true
  endif
endif
if (checkStateRelationship(n1.as, n2.as.effect) = composition) then
  if ((n1.deoC = P or n1.deoC = O) and (n2.deoC = F)) and
     n1.as is 'wholeState' ) then
     if (checkAllBehavior(selectActionsHaveEffect(n1.as, F)) then
        return true
     endif
  endif
endif
if (checkStateRelationship(n1.as, n2.as) = composition) then
  if ((n1.deoC = F) and n1.as is 'wholeState' ) then
    if (checkAllPartStates(n1.as, P) or checkAllPartStates(n1.as, O)) then
       return true
    endif
  endif
endif
if (checkStateRelationship(n1.as, n2.as.effect) = composition) then
  if ( ((n1.deoC = F) and n1.as is 'wholeState') and
     ((n2.deoC = O) or (n2.deoC = P)) then
      if (checkAllBehavior(selectAllPartStates(n1.as, O)) or
         checkAllBehavior(selectAllPartStates(n1.as, P)) ) then
            return true
      endif
  endif
endif
return false
end function
```

Figure 2. Verifying composition relationship.

Algorithm 2 checks if the states being regulated by the norms are related by the composition relationship. If it is the case, it follows the strategies presented in Section 3.2:

- (Sections3.2.1) If a norm obliges (or permits) the achievement of the *wholeState* and the other prohibits the achievement of a *partState*, these norms are in conflict. In addition, if a norm obliges (or permits) the achievement of the *wholeState* and the other prohibits the execution of an action which effect is the *wholeState*/*partState*, there is a potential conflict. It is important to check if it is the only action that can achieve the *wholeState*/*partState*. If it is, there is a conflict. If not, there will be a conflict if all actions are being prohibited.
- (Section 3.2.2) If a norm prohibits the achievement of a *wholeState*, there will be a conflict if all *partStates* are being permitted or obliged. In addition, if a norm prohibits the achievement of a *wholeState* and the other obligates or permits the execution of an action which effects is a *partState* and, there will be a potential

conflict. If there are obligations or permissions applied to all other *partStates*, there will conflicts.

Algorithm 3 is responsible to check for conflicts between behaviors that are orthogonal, following the strategies described in Section 3.3:

Algorithm 3 Verifying Orthogonal Relationship

```
Require: n1 and n2 as parameter
Function: orthogonalRelationship(n1, n2)
if ((checkStateRelationship(n1.as, n2.as) = orthogonal) and
   (n1.deoC = O or n1.deoC = P) and (n2.deoC = O or n1.deoC = P)) then
     return true
endif
if ((checkStateRelationship(n1.as, n2.as.effect) = orthogonal) and
   (n1.deoC = O or n1.deoC = P) and (n2.deoC = O or n1.deoC = P)) then
     return true
endif
if ((checkStateRelationship(n1.as, n2.as.precondition) = orthogonal) and
   (n1.deoC = O or n1.deoC = P) and (n2.deoC = F) then
   if (checkAllBehavior(selectAllActionHavePrecondition(n1.as, F))) then
      return true
   endif
endif
if ((checkStateRelationship(n1.as, n2.as.precondition) = orthogonal) and
   (n1.deoC = F) and (n2.deoC = O or n2.deoC = P)  then
   if (n1.as = n2.as.effect) then
      return true
   endif
endif
return false
end function
```

Figure 3. Verifying orthogonal relationship.

- (Sections 3.3.1) If a norm obligates (or permits) the achievement of a state and the other obligates (or permits) the achievement of another state orthogonal to the first one, they are in conflict. In addition, if there is a norm obligating (or permitting) the achievement of a state and the other obligating (or permitting) the execution of an action which **effect** is orthogonal to such state, there is a conflict. Moreover, if a norm obligates (or permits) the achievement of a state and the other prohibits the execution of an action which precondition is orthogonal to such state, there is a potential conflict. It is important to check if all actions which precondition is such state are being prohibited. If it is the case, there are conflicts.

Algorithm 4 Verifying Relationships – Main

```
Require: n1 and n2 from the norm base
if (n1.as = n2.as and
   ((n1.deoC = O and n2.deoC = F) or (n1.deoC = P and n2.deoC = F))) then
       return norms are in conflict!
else
   if (dependencyRelationship(n1, n2) or
      dependencyRelationship(n2, n1) or
      compositionRelationship(n1, n2) or
      compositionRelationship(n2, n1) or
      orthogonalRelationship(n1, n2) or
      orthogonalRelationship(n2, n1)) then
         return norms are in conflict!
   endif
endif
return Norms are not in conflict!
end function
```

Figure 4. Verifying Relationships – Main.

- (Sections 3.3.2) If there is a prohibition to achieve a state and an obligating (or permitting) to execute an action which precondition is orthogonal to the state, there is a potential conflict. It is important to check if the effect of such action is the one being prohibited.

Algorithm 4 is responsible to call the others algorithms. It is the main algorithm and what returns the finally result about the existence of conflict (or not) among the norms. In order to exemplify our approach, let's consider the compositions relationships described in Section 3.2 and the norms N1 and N2 below. N1 prohibits the execution of the action to load the plan and N2 obligates the plan to be ready. We assume that the contexts are the same, the entities are the same (or are related) and the norms are applied in period of time that intersects.

```
N1 = {Obligation, ready(p)}
N2 = {Prohibition, load(c,p)}
```

By analyzing the domain, we know that the state *ready(p)* is a *wholeState* and *in(c,p)* and *fueled(c,p)* are *partStates*. *Ready(p)* can only be achieved if *in(c,p)* and *fueled(c,p)* are achieved. Besides, in order to achieve *in(c,p)* it is necessary to execute *load(c,p)*. **Algorithm 2** identifies that *n1.as* (i.e., *ready(p)*) is composed of *n2.as*.effect (i.e., of *in(c,p)*) and that n1 is an obligation and n2 a prohibition. Since *loaf(c,p)* is the only action that can achieve *in(c,p)* and this action is being prohibited, the algorithm concludes that N1 and N2 are in conflict.

5. Related Work

Normative conflicts in Multi-Agentes Systems have been a motivator to several works and researchers. We can divide the checking of normative conflicts in two groups: checking of direct normative conflicts and checking of indirect normative conflicts.

The majority of works is concentrated in direct conflicts and it not is able to check indirect conflicts (that is domain-dependent). The authors in [Kollingbaum et al. 2008], [Vasconcelos et al. 2007], [Vasconcelos et al. 2009] and [Vasconcelos and Norman 2009] presents approaches for the checking of norms applied to the same action. This works are not able to detect indirect conflicts.

Some papers focus on the checking of indirect conflicts, such as [Dung and Sartor 2011], [Figueiredo et al. 2011], [Garcia-Camino et al. 2006], [Kollingbaum et al. 2008] and [Vasconcelos et al. 2007]. The approaches in [Gaertner et al. 2007] and [Garcia-Camino et al. 2006] take into account the normative position, which describes activities that are propagated to other activities. In [Gaertner et al. 2007] the approach considers that multiple, concurrent and related activities are executed by agents and present a conflict checker that considers those. The authors in [Garcia-Camino et al. 2006] consider the composition relationship between activities. In addition, they state that the conflict-free normative positions of an activity propagate to its sub-activities. A normative conflict occurs when the normative positions coming from the super-activity contradicts the normative position of a sub-activity.

In [Kollingbaum et al. 2008] and [Vasconcelos et al. 2007] the normative conflict checker considers indirect conflicts by taking into account the domain specific relationships among actions. The composition and delegation relationship are defined between actions and they use unification to find out the norms that overlap. This

approach is incomplete, due not consider the entities relationships. In [Aphale et al. 2012] the authors present a model of conflict identification and resolution that focuses attention on conflicts that are most critical to the goals of the organization.

The work presented in [Dung and Sartor 2011] focuses on conflicts between norms defined in different contexts. According to the authors, a particular situation can be judged by different legal systems and the norms of those systems can conflict. Similar to such approach, the works presented in [Li et al. 2013a] and [Li et al. 2013b] are able to detect conflict among different laws defined in different jurisdictions. In this paper, we focus on conflicts between norms defined in the same legal systems.

In [Silva 2013] the author presented an algorithm to detect conflicts between two norms. The algorithm focuses on detecting conflicts between a prohibition and an obligation that do not govern the behavior of the same entity, but entities that are somehow related. In addition, this first version of the conflict checker algorithm can also identify conflicts between a prohibition and an obligation that are applied to different actions related by the refinement and composition relationships. In this current paper, we focus on describing the relationships between states and on checking for conflicts between norms that regulate different states and between norms that regulate an action and a state

6. Conclusion and Future Work

This paper presents an extension of our preliminary work [Zahn and Silva 2014] on the detection of indirect normative conflicts. The first version of the conflict checker [Zahn and Silva 2014] is able to check for conflicts between norms addressed to different but related entities and that regulate the execution of different but related actions. This conflict checker algorithm was extended in this current paper to be able to check for conflicts between norms that regulate the achievement of different but related states and between norms when one regulates the achievement of a state and the other the execution of an action. In other to do so, the algorithm verifies the domain-dependent relationships between the states of the multi-agent systems and also the definitions of the actions. The paper defines (only) three different relationships that can be used to relate states but note that others can be created. The conflict checker algorithm as implemented in Jess < *http://goo.gl/CAxIay*> and is available at <*http://goo.gl/h4hwoz*>.

We are in the process of defining other relationships that can be used to relate actions (dependency and orthogonality) that were not described in [Zahn and Silva 2014] and extending the conflict checker to follow these relationships. In addition, we have noticed that there are some conflicts between norms that can only be detected when we analyze more than two norms. We are defining a strategy to check for conflicts that can analyze multiple norms at the same time without configure an NP problem.

References

Aphale, M. S., Norman, T. J., Sensoy, S., “Goal Directed Conflict Resolution and Policy Refinement,” in International Workshop on Coordination, Organisations, Institutions and Norms, 2012, pp. 87-104.

Dung, P., Sartor, G., “The modular logic of private international law,” in Artificil Intelligence and Law. Springer, 19(2-3), 2011, pp. 233-261.

Figueiredo, K., Silva, S., Braga, C., “Modeling Norms in Multi-agent Systems with Norm-ML,” in International Workshop on Coordination, Organisations, Institutions and Norms VI. LNAI 6541, Springer, 2011, pp. 39-57.

Gaertner, D., Garcia-Camino, A., Noriega, P., Vasconcelos, W., “Distributed Norm Management in Regulated Multi-agent Systems,” in International Conference on Autonomous Agents and Multiagent Systems. ACM, 2007, pp. 624-631.

Garcia-Camino, A., Noriega, P., Rodrigues-Aguilar, J., “An Algorithm for Conflict Resolution in Regulated Compound Activities,” in Engineering Societies in the Agents World VII, LNCS 4457, Springer, 2006, pp. 193-208.

Kollingbaum, M., Vasconcelos, W., Garcia-Camino, A., Norman, T., “Managing Conflict Resolution in Norm-Regulated Environments,” in Engineering Societies in the Agents World VIII, LNCS 4995, Springer, 2008, pp. 55-71.

Kollingbaum, M., Vasconcelos, W., Garcia-Camino, A., Norman, T., “Conflict Resolution in Norm regulated Environments via Unification and Constraints,” in Declarative Agent Languages and Technologies V, LNCS 4897, Springer, 2008, pp. 158-174.

Li, T., Balke, T., De Vos, M., Satoh, K., Padget, J., ”Detecting Conflicts in Legal Systems. In New Frontiers in Artificial Intelligence,” LNCS 7856, Springer, pp. 174-189, 2013a.

Li, T., Balke, T., De Vos, M., Padget, J., Satoh, K., “Legal Conflict Detection in Interacting Legal Systems.” in The 26th International Conference on Legal Knowledge and Information Systems (JURIX), 2013.

Russell, Stuart J., Norvig, Peter., Artificial Intelligence: a modern approach. 3rd ed., Pearson, Upper Saddle River, New Jersey, 2010.

Silva, V., “Normative Conflicts that Depend on the Application Domain,” in International Workshop on Coordination, Organisations, Institutions and Norms, 2013, pp. 119-130.

Vasconcelos, W., Kollingbaum, M., Norman, T., “Resolving conflict and inconsistency in norm regulated virtual organizations,” in International Conference on Autonomous Agents and Multiagent Systems. ACM, 2007, pp. 632-639.

Vasconcelos, W., Kollingbaum, M., Norman, T., “Normative conflict resolution in multi-agent systems,” in Journal of Autonomous Agents and Multi-Agent Systems. ACM, 19(2), 2009, pp. 124-152.

Vasconcelos, W., Norman, T., “Contract Formation through Preemptive Normative Conflict Resolution,” in International Conference of the Catalan Association for Artificial Intelligence. ACM, 2009, pp. 179-188.

Zahn, J. O., Silva, V. T., “On the Checking of Indirect Normative Conflicts,” in: Workshop-Escola de Sistemas de Agentes, seus Ambientes e aplicações, 2014, Porto Alegre. Anais do Workshop-Escola de Sistemas de Agentes, seus Ambientes e aplicações, 2014. p. 13-24.

Verificação de conflitos normativos em sistemas multiagentes: uma abordagem visual

Daniela Godinho Yabe[1], Eduardo Augusto Silvestre[2] e Viviane Torres da Silva[3]

[1]Instituto Federal do Triangulo Mineiro – IFTM

[2]Universidade Federal Fluminense – UFF

[3]IBM Research, Brazil

danielagodinho1@gmail.com, eduardosilvestre@iftm.edu.br, vivianet@br.ibm.com

Abstract. *In multi-agent systems, a norm is simply an established, expected pattern of behavior. Norms describe the behavior that can be performed, that must be performed, and that cannot be performed in the system. One of the main challenges on developing normative systems is that norms may conflict with each other. Norms are in conflict when the fulfillment of one norm violates the other and vice-versa. Several authors have investigated normative conflicts in the literature, but the approaches are based on a strong logical-mathematical notation and was no found a simple implementation in high level language. This paper presents algorithms that check for conflicts between pairs of norms. It also presents a visual application for creating, importing and verification of conflicts in multi-agent systems.*

Resumo. *Em sistemas multiagentes, uma norma é um padrão de comportamento estabelecido e esperado. Normas descrevem o comportamento que pode ser executado, que deve ser executado e que não pode ser executado. Um dos principais desafios em sistemas multiagentes é que as normas podem entrar em conflito. Normas estão em conflito quando o cumprimento de uma norma viola uma outra norma e vice-versa. Vários autores pesquisaram conflitos normativos na literatura, mas as abordagens são baseadas em uma forte notação lógica-matemática e não foi encontrado uma implementação em linguagem de alto nível. Este trabalho apresenta algoritmos que verificam conflitos entre pares de normas. Apresenta também uma aplicação visual para criação, importação e verificação de conflitos em sistemas multiagentes.*

1. Introdução

Atualmente, os sistemas que usam agentes vem ganhando importância na pesquisa e na prática para o desenvolvimento de aplicações diversas. Segundo Russell e Norvig (2009), um agente de software é uma entidade capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores. Os agentes inteligentes podem ser classificados de acordo com a maneira que eles coletam informações e agem no ambiente. No caso de vários agentes cooperando ou disputando entre si, inseridos em um mesmo ambiente e trocando informações, chamamos esse

sistema de multiagente (SMA) (RUSSELL E NORVIG, 2009). SMAs são sociedades autônomas, heterogêneas e podem trabalhar a fim de alcançar objetivos comuns ou diferentes (WOOLRIDGE, 2009).

Um agente é um sistema de computador que está situado em algum ambiente e é capaz de realizar comportamentos autônomos neste ambiente a fim de atingir seus objetivos (WOOLRIDGE, 2009). Autonomia é considerada uma propriedade que possibilita agentes tomarem decisões e, para isto, eles consideram os seus interesses em realizar determinados comportamentos, ou seja, consideram a sua motivação para atingir determinados objetivos ou para executar determinadas ações (LÓPEZ, 2003).

A fim de lidar com a heterogeneidade, autonomia e diversidade de interesses entre os agentes da sociedade, projetistas desses sistemas estabelecem um conjunto de normas que é usado como um mecanismo de controle social que visa possibilitar que os agentes possam trabalhar em conjunto (SILVA, 2008).

Essas normas regulam o comportamento dos agentes, com definições de obrigações, proibições e permissões. Como os agentes são autônomos, podem existir situações onde um agente prefere violar uma norma a fim de realizar um determinado comportamento que é mais importante do que o cumprimento da norma. A introdução de normas em sistemas multiagentes tem sido considerada como um fator importante para garantir a eficácia dos agentes (LÓPEZ, 2003).

Neste contexto, vale ressaltar a possibilidade de existir conflitos entre normas. Os conflitos acontecem quando duas normas regulando o mesmo comportamento estão ativadas e tem restrições inconsistentes (VASCONCELOS ET AL., 2009). A verificação e a resolução de conflitos entre normas em sistemas multiagentes são importantes para manter a consistência do comportamento autônomo dos agentes.

Após uma extensa pesquisa bibliográfica, vários artigos foram encontrados (Cholvy and Cuppens (1995), Elhag et al (2000), Kollingbaum et al (2007), Vasconcelos et al (2009)). A partir da análise desses artigos constatou-se que a maioria deles utiliza uma notação lógica matemática de forma extensiva e não existe implementação dos algoritmos de verificação de conflitos em uma linguagem de alto nível. Além disso, não foi encontrada nenhuma ferramenta capaz de criar normas e realizar a verificação de pares de conflitos normativos. Os autores desenvolveram uma ferramenta onde é possível criar normas, importar normas e também verificar pares de conflitos normativos.

Este artigo apresenta uma ferramenta visual para verificação de conflitos entre pares de normas. As principais contribuições apresentadas neste artigo são: (i) algoritmos para verificação de conflitos entre pares de normas; (ii) aplicação em linguagem de alto nível que verifica conflitos entre pares de normas e (iii) uma interface visual para verificação de conflitos e possível inserção de novas funcionalidades.

O restante do artigo é organizado da seguinte forma: a Seção 2 define normas e conflitos normativos, a Seção 3 define a aplicação desenvolvida, os algoritmos e exemplos de entradas de dados no sistema. Por fim, a seção 4 apresenta as considerações finais.

2. Normas e conflitos normativos

Neste artigo, a definição de uma norma é a mesma usada por Figueiredo et. al. (2011). A definição é ampla e cobre definições apresentadas em artigos anteriores.

Várias especificações, linguagens e metodologias definem uma norma de forma semelhante. A norma é associada a um conceito deôntico, um contexto, uma entidade e uma ação (ou estado) a ser regulada.

Definição (norma): Uma norma n é uma tupla da forma {deoC, c, e, a, ac, dc} onde deoC é o conceito deôntico do conjunto {obrigação, proibição e permissão}, c é contexto onde a norma é definida, e é a entidade que está sendo regulada, a é a ação a ser regulada, ac é a condição de ativação e dc e a condição de desativação (FIGUEIREDO ET AL., 2011).

O contexto de uma norma indica o âmbito onde a norma está definida. A norma deve ser cumprida apenas quando a entidade está em execução em tal contexto. Fora do seu contexto, a norma não é válida. Quase todas as abordagens consideram que uma norma está definida no contexto de uma organização. Observa-se que algumas abordagens consideram que o contexto pode ser uma interação ou uma cena, por exemplo. Neste trabalho, consideramos que uma norma pode ser definida no contexto de uma organização ou de um ambiente que é o habitat das entidades.

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

Duas normas estão em conflito se tem conceitos deônticos opostos, se estão no mesmo contexto, se são referentes a mesma entidade e ação, tem períodos de validade que se interceptam e não foram cumpridas e nem violadas.

3. Aplicação desenvolvida

A literatura que analisa conflitos normativos carece de aplicações que facilitem a visualização e o estudo de conflitos normativos. Neste trabalho, é desenvolvida uma aplicação capaz de verificar todos os conflitos entre pares de normas existentes em um sistema multiagente. Os autores desenvolveram algoritmos que oferecem ao usuário condições de inserção ou importação de normas que serão analisadas e mostradas, caso haja conflitos entre elas.

O sistema apresenta algoritmos para verificação de cenários de conflito, classes para apresentar uma norma e uma arquitetura que permite a implantação de novas funcionalidades.

A Figura 1 apresenta o diagrama de classes UML do programa implementado.

A linguagem Java foi utilizada para o desenvolvimento do programa. A IDE Eclipse foi utilizada como ambiente de desenvolvimento. As principais abstrações, bibliotecas e suas funcionalidades básicas são:

- Bahavior: classe abstrata que representa o comportamento de cada norma;
- BehaviorAtomicAction: classe concreta que herda de Behavior e instancia uma ação;
- ConflictChecker: classe concreta que possui as funções utilizadas para verificar se há conflito entre pares de normas;
- Constraint: classe abstrata que representa as condições de ativação ou desativação de cada norma, bem como o tipo e a data;
- ConstraintDate: classe concreta que herda de Constraint e armazena a data de ativação e desativação de cada norma;

- ConstraintType: define constantes para representar o tipo da restrição (data ou ação);
- Context: classe abstrata que representa o contexto da norma;
- ContextType: define constantes para representar o tipo do contexto (organização ou ambiente);
- DeonticConcept: define constantes para representar o tipo de conceito deôntico (obrigação, permissão, proibição);
- Entity: classe abstrata que representa a entidade da norma;
- EntityType: define constantes para representar o tipo da entidade (agente, papel, organização ou todos);
- Norm: classe concreta que representa uma norma;
- GUI: conjunto de classes que fornecem acesso à interface visual;
- Biblioteca *joda-time*: biblioteca utilizada para restrições temporais.

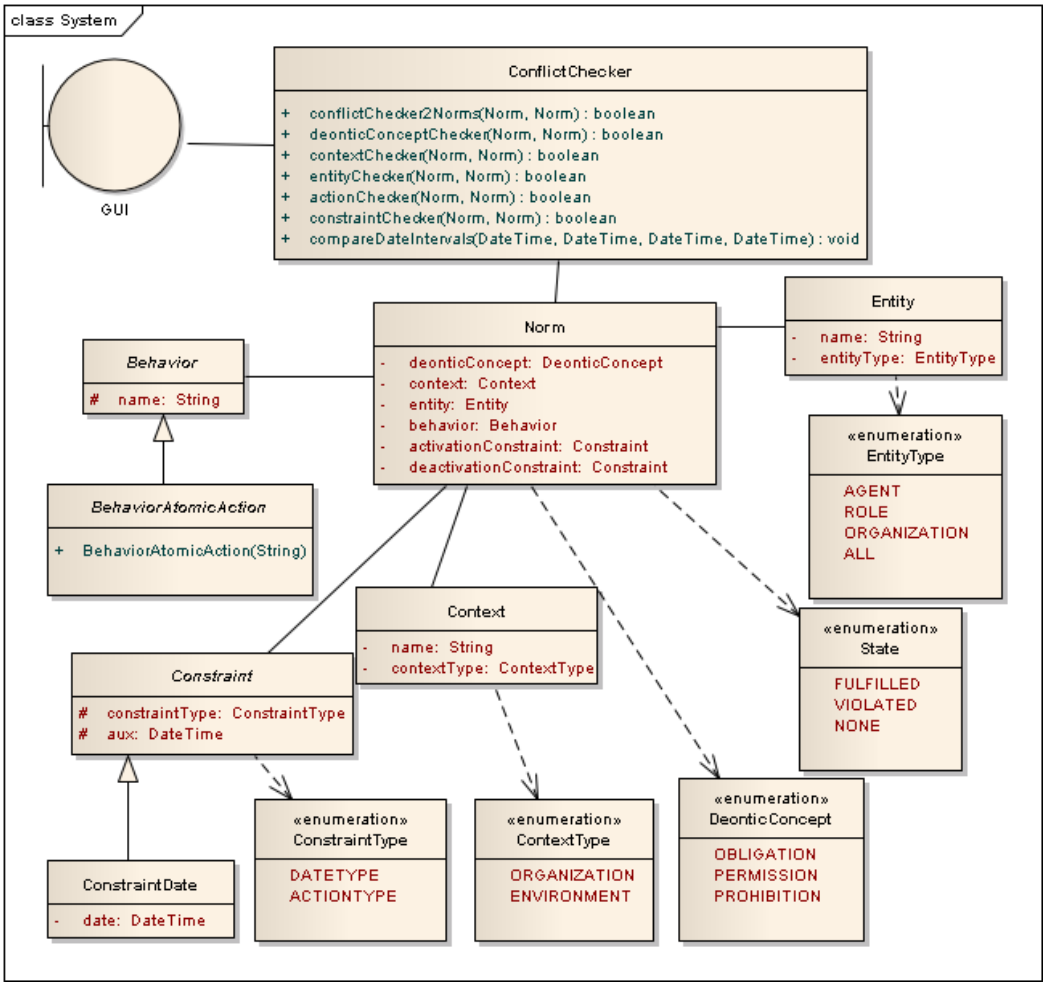


Figura 1 - Diagrama de classes UML.

A tela principal do software (Figura 2) apresenta um menu contendo as opções "Import Norm", "Conflict between 2 norms" e um formulário que permite ao usuário cadastrar normas manualmente de acordo com as características do seu sistema multiagente. Após preencher o formulário o usuário poderá carregar a norma cadastrada

para a tabela de verificação de conflitos através do botão “Save”. Ao escolher a opção “Import Norm” o usuário será direcionado para tela de importação (Figura 3) de normas pré-cadastradas no software. Estas normas pré-cadastradas permitem realizar uma análise prévia e ao mesmo tempo demonstrar a utilização da aplicação. Após o usuário inserir ou importar as normas, o usuário pode clicar no botão “Conflict between 2 norms”, o sistema irá analisar as normas selecionadas em pares e verificar se há conflito entre elas.

Nas próximas seções serão apresentados os principais algoritmos utilizados nesta abordagem. Além disso, o código implementado está disponível para download em: github.com/eduardoasilvestre/SimpleConflictsShowAllVisual.

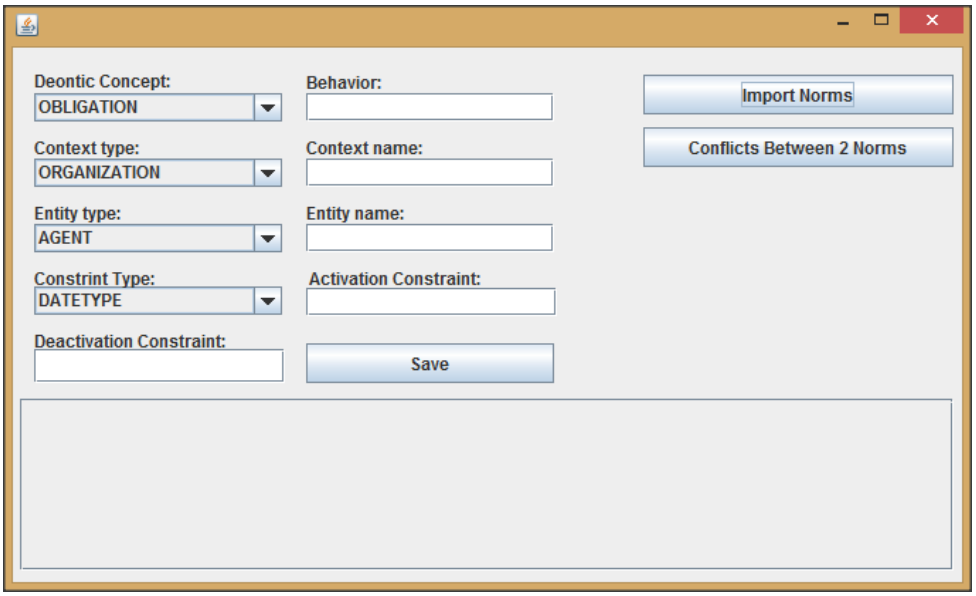


Figura 2 - Tela inicial do sistema.

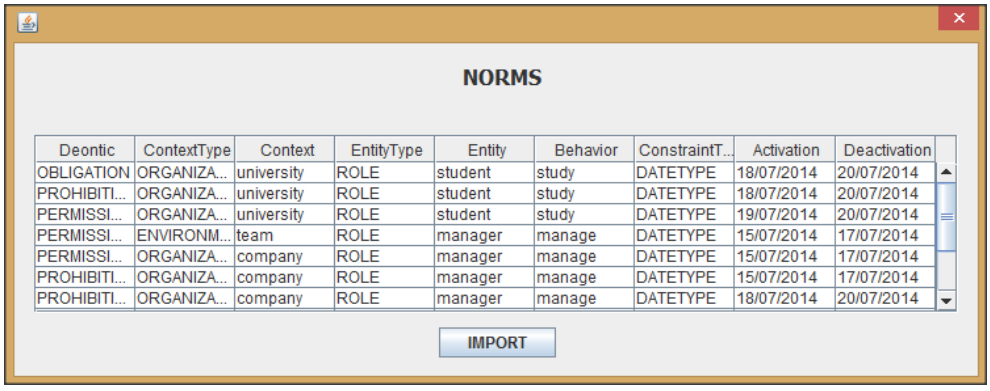


Figura 3 - Tela de importação de normas.

3.1. Algoritmos

Nesta seção, apresentaremos o algoritmo para verificação de conflitos entre pares de normas que foi implementado. O Algoritmo 1 recebe uma lista de normas que são cadastradas ou importadas pelo usuário. Desta forma, o algoritmo verifica o conflito direto entre duas normas utilizando funções auxiliares que analisam os conceitos

deônticos, ações, entidades, contextos e a condição de ativação e desativação de cada norma.

Algoritmo 1 Main

```
normSet ← receives the norms of the GUI
for i ← 1 until size(normSet) do
  for j ← i + 1 until size(normSet) do
    norm1 ← normSet[i]
    norm2 ← normSet[j]
    conclflictChecker ← conclflictChecker2Norms(norm1,norm2)
    if (conclflictChecker = true) then
      print "The following norms are in conflict"
      print norm1
      print norm2
    endif
  endfor
endfor
```

Algoritmo 1 - Algoritmo principal (main).

O Algoritmo 2 recebe as normas de entrada e apenas envia para funções específicas para análise.

Algoritmo 2. Função: Chama as funções de verificação de conflito

```
Require: norm1 and norm2 as parameter
function conclflictChecker2Norms(norm1, norm2)
      if (deonticConceptChecker(norm1, norm2) = false) then
        return false
      endif
      if (stateChecker(norm1, norm2) = false) then
        return false
      endif
      if (contextChecker(norm1, norm2) = false) then
        return false
      endif
      if (entityChecke(norm1, norm2) = false) then
        return false
      endif
      if (constraintChecke(norm1, norm2) = false) then
        return false
      endif
      if (actionChecke(norm1, norm2) = false) then
        return false
      endif
      return true
endfunction
```

Algoritmo 2 - Chama as funções de verificação de conflito.

O Algoritmo 3 analisa se os conceitos deônticos das duas normas são opostos, verificando se há conflito entre uma norma que obriga e outra que proíbe, uma norma que proíbe e outra que permite e uma norma que obriga e outra que permite.

Algoritmo 3. Função: Verifica conceito deôntico

```
Require: norm1 and norm2 as parameter
function deonticConceptChecker(norm1, norm2)
      if ((norm1.deonticConcept = PERMISSION) and (norm2.deonticConcept  = OBLIGATION) or
```

```
(norm1.deonticConcept = PROHIBITION)  and (norm2.deonticConcept =PERMISSION) ) then
        return true
    endif
    if ((norm2.deonticConcept = PROHIBITION) and (norm1.deonticConcept  = OBLIGATION) or
(norm2.deonticConcept =  PROHIBITION)  and (norm1.deonticConcept =PERMISSION) ) then
        return true
    endif
    return false
endfunction
```

Algoritmo 3 - Verifica o conceito deôntico entre pares de normas.

O Algoritmo 4 analisa se as duas normas tem as mesmas ações.

Algoritmo 4. Função: Verifica ação

```
Require: norm1 and norm2 as parameter
function actionChecker(norm1, norm2)
    if (b1.behavior = null or b2.behavior = null)  then
        return false
    endif
    return (b1.behavior = b2.behavior)
endfunction
```

Algoritmo 4 - Analisa a ação entre pares de normas.

O Algoritmo 5 verifica se as entidades de cada norma são idênticas.

Algoritmo 5. Função: Verifica entidade

```
Require: norm1 and norm2 as parameter
function entityChecker(norm1, norm2)
    return (norm1.entity = norm2.entity)
endfunction
```

Algoritmo 5 – Analisa a entidade entre as normas.

O Algoritmo 6 analisa se o contexto de ambas as normas são iguais.

Algoritmo 6. Função: Verifica contexto

```
Require: norm1 and norm2 as parameter
function contextChecker(norm1, norm2)
    return (norm1.context = norm2.context)
endfunction
```

Algoritmo 6 - Analisa o contexto entre pares de normas.

Enquanto o Algoritmo 7 verifica se a condição de ativação e desativação de cada norma são iguais ou se interceptam.

Algoritmo 7. Função: Verifica condição de ativação e desativação

```
Require: norm1 and norm2 as parameter
function constraintChecker(norm1, norm2)
    if ((norm1.dConstraint > norm2.aConstraint) or (norm2.dConstraint > norm1.aConstraint)) then
        return true
    endif
    return false
endfunction
```

Algoritmo 7 - Verifica a condição de ativação e desativação das normas.

3.2. Exemplo

Para facilitar a compreensão dos algoritmos juntamente com a aplicação visual, esta seção irá fornecer um exemplo de uso do programa. Suponha que o Agente A tenha que cumprir as normas N1, N2 e N3:

N1 = O agente estudante é obrigado a estudar na universidade entre x e y.
N2 = O agente estudante é proibido de estudar na universidade entre x e y.
N3 = O agente estudante é permitido de estudar na universidade entre x e y.

A partir da definição de uma norma apresentada na Seção 2 pode-se obter:

N1 = {deoC=OBRIGADO, c=universidade, e=estudante, a=estudar, ac=x, dc=y}.
N2 = {deoC=PROIBIDO, c=universidade, e=estudante, a=estudar, ac=x, dc=y}.
N3 = {deoC=PERMITIDO, c=universidade, e=estudante, a=estudar, ac=x, dc=y }.

Onde, deoC é o conceito deôntico, c é o contexto onde a norma é definida, e representa a entidade que tem de cumprir a norma, a ação a ser executada, ac o momento que a norma começa a ser válida e dc o momento em que a norma não é mais válida.

A Figura 4 mostra as normas instanciadas e prontas para serem verificadas. A aplicação permite que o usuário crie normas manualmente de acordo com as especificidades do seu sistema multiagente. As normas criadas são carregadas para uma tabela na tela. Além disso, é permitido que o usuário selecione quais normas ele deseja verificar o conflito.

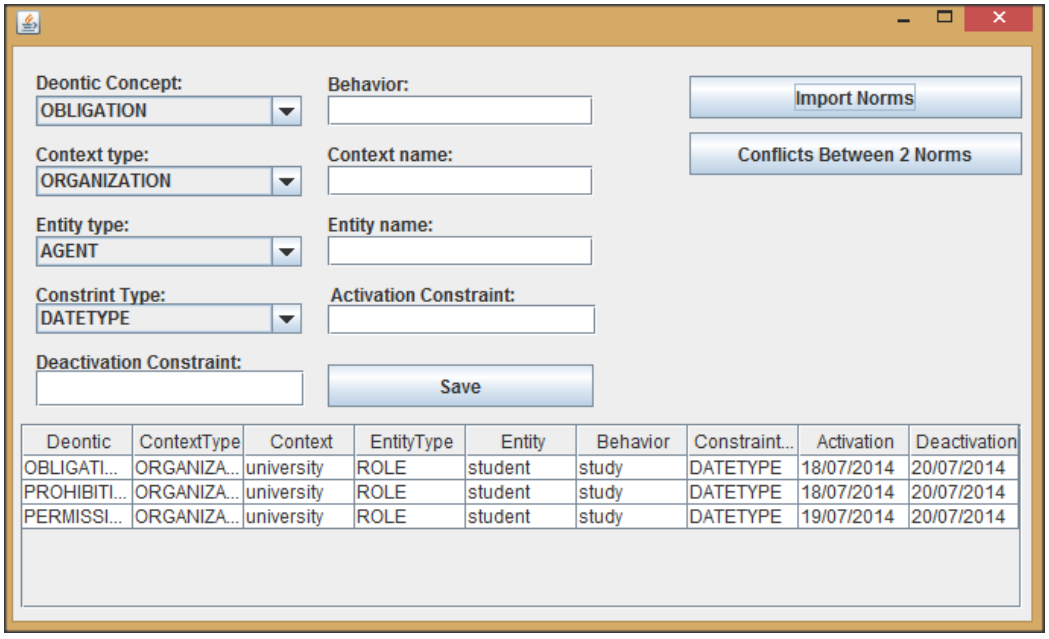


Figura 4 - Tela inicial com normas preenchidas.

Após a execução do algoritmo a aplicação retornará que existe conflito entre as normas N1 e N2 e entre as normas N2 e N3 (Figura 5). O agente A não é capaz de

cumprir porque, no primeiro caso, uma norma proíbe e outra obriga, já no segundo caso, uma proíbe e a outra permite

Figura 5 - Mensagem mostrando as normas que estão em conflito.

4. Considerações finais

Os sistemas multiagentes vem ganhado maior importância nos últimos anos, tanto nas universidades quanto nas empresas. Devido a essa amplitude existem várias problemas em aberto. Um dos principais problemas é garantir a existência de conjunto de normas sem conflito. E, a partir deste problema vários estudos vêm sendo realizados.

Este trabalho identificou uma carência na verificação de conflitos normativos. A maioria dos trabalhos na literatura é baseado em forte notação matemática, mas não apresenta uma implementação algoritmos/códigos que facilitem a solução do problema.

A principal contribuição deste trabalho é a criação de algoritmos e a implementação em linguagem de alto nível de soluções que verificam conflitos diretos entre pares de normas em sistemas multiagentes. Além disso foi desenvolvida uma aplicação visual para criação e análise de conflitos entre pares de normas.

Como a arquitetura suporta inserção de novas funcionalidades, para trabalhos futuros pretende-se implementar novos algoritmos que possam resolver problemas de conflitos normativos recorrentes.

Referências

Cholvy, L., & Cuppens, F. (1995, May). Solving normative conflicts by merging roles. In Proceedings of the 5th international conference on Artificial intelligence and law (pp. 201-209). ACM.

Elhag, A. A. O., Breuker, J. A. P. J., & Brouwer, P. W. (2000). On the formal analysis of normative conflicts. Information and Communication Technology Law, 9(3), 207-217.

Figueiredo, K., Silva, S., Braba, C., (2011). Modeling Norms in Multi-agent Systems with Norm-ML. In International Workshop on Coordination, Organisations, Institutions and Norms VI. LNAI 6541, Springer, pp. 39-57.

López, F. (2003). Social Power and Norms: Impact on agent behaviour. Phd thesis,University of Southampton.

Kollingbaum, M. J., Norman, T. J., Preece, A., & Sleeman, D. (2007). Norm conflicts and inconsistencies in virtual organisations. In Coordination, Organizations,

Institutions, and Norms in Agent Systems II (pp. 245-258). Springer Berlin Heidelberg.

Russel, S., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach, 3rd edition. Pearson Education.

Silva, V. T. (2008). From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. Autonomous Agents and Multi-Agent Systems, 17(1), 113-155.

Vasconcelos, W. W., Kollingbaum, M. J., & Norman, T. J. (2009). Normative conflict resolution in multi-agent systems. Autonomous Agents and Multi-Agent Systems, 19(2), 124-152.

Wooldrige, M. (2009). An introduction to multiagent systems. John Wiley & Sons.

Modelo Baseado em Equações *versus* Modelo Baseado em Agentes: uma abordagem usando sistema predador-presa

Diego A. Porcellis[1], Carlos Bertin[1], Marcilene Moraes[1]

[1]Programa de Pós-Graduação em Modelagem Computacional, PPGMC – FURG
Universidade Federal do Rio Grande

{diegoporcellis, marcilenemorais}@furg.br, carlosbertin@gmail.com

Abstract. *This paper presents a comparative study between a equation-based model and the agent-based model, where we simulate the predator-prey system of Lotka-Volterra and the other using often techniques used in social modeling, to represent the behavior of agents in a given environment. In both approaches highlights the advantages and disadvantages of each, and finally we present a comparison between the results.*

Resumo. *Este artigo realiza um estudo comparativo entre um modelo baseado em equações e o modelo baseado em agentes, no qual um simula o sistema predador-presa de Lotka-Volterra e o outro utiliza técnicas frequentemente adotadas na modelagem social, para representar o comportamento dos agentes em um determinado ambiente. Em ambas as abordagens, evidencia-se as vantagens e desvantagens de cada um, e por fim é apresentado um comparativo entre os resultados encontrados.*

1. Introdução

Métodos de simulação tem se apresentado eficientes no auxílio da tomada de decisões, no planejamento de médio a longo prazo e em situações que envolvem custos e riscos elevados. O uso do computador em simulações é de grande valia, pois possibilita visualizar o comportamento de sistemas sem a necessidade de realização efetiva das modificações ou a construção, por vezes, de protótipos caros [REBONATTO, 2000].

Neste sentido, surge o conceito de Sistemas Multiagentes (SMA), sub-área da Inteligência Artificial Distribuída (IAD), no qual permite estudar "o comportamento de um conjunto de agentes autônomos, eventualmente com características diferentes, evoluindo em um ambiente comum. Estes agentes podem interagir uns com os outros, com o objetivo de realizar suas tarefas de modo cooperativo, compartilhando informações, evitando conflitos e coordenando a execução de atividades" [Alvares e Sichman *apud* ADAMATTI 2011]. A integração entre tecnologias de agentes e métodos de simulação, permite analisar o cenário em estudo de uma forma mais realista.

O modelo baseado em agentes, que simula um sistema predador-presa tem por objetivo demonstrar a cooperação em sistemas multiagentes (entre os predadores) e, por outro lado, a atuação independente da presa. Assim, é possível explorar o comportamento social dos agentes no meio em que estão inseridos. Os agentes predadores devem identificar a presa, para então iniciar a sua perseguição e, consequentemente, a sua captura [MOMM 2012].

Para representar a interação entre presa e predadores, os estudiosos Lotka e Volterra elaboraram independentemente, o sistema predador-presa baseado em equações, chamado de modelo Lotka-Volterra. Trata-se de um par de equações diferenciais, não lineares e de primeira ordem, utilizadas para descrever dinâmicas de sistemas biológicos, mais especificamente quando duas espécies interagem: uma como presa e outra como predadora. A lógica e a teoria matemática sugerem que quando as presas são numerosas, seus predadores aumentam em número, reduzindo assim a população de presas, que por sua vez faz com que o número de predadores decline. A população de presas eventualmente se recupera, iniciando um novo ciclo [NADELHOFFER, 2005].

Por meio de um efetivo estudo do modelo de Lotka-Volterra, um modelo baseado em equações (MBE), pode-se extrair características comportamentais do sistema predador-presa. Com o enfoque do trabalho voltado para o comportamento social do modelo, criou-se uma modelo baseado em agentes, adotando técnicas comumente utilizadas na modelagem social. No MBE a simulação é implementada através de um método matemático, utilizando equações diferenciais inter-relacionadas, tornando o modelo mais abstrado, com agentes homogêneos, apresentando resultado global, a longo prazo. Já no modelo baseado em agentes (MBA), os agentes são heterogêneos, com comportamento individualizado, permitindo assim retratar melhor a vida real.

Com intuito de simular a dinâmica interativa dos agentes, utilizou-se o ambiente de programação Netlogo. Uma de suas principais características é que ele realiza simulações voltados para fenômenos naturais e sociais.

O objetivo deste estudo é discutir as vantagens e desvantagens dos modelos MBE e MBA, utilizando o ambiente de programação Netlogo e tendo como tema o modelo de simulação predador-presa. O artigo está dividido em seções a seção 2 conceitua e distingue o MBE do MBA, a seção 3 define o modelo de Lotka-Volterra, a seção 4 demonstra a ferramenta NetLogo, a seção 5 aborda sobre a implementação realizada, na seção 6 são feitas as discussões sobre os resultados encontrados e o trabalho é concluído na seção7.

2. MBE e MBA

O MBE e o MBA são duas técnicas de modelagem de sistemas sociais difundidas. A MBA opõe-se ás abordagens tradicionais de simulação que geralmente são realizadas através das MBE, onde são construídas a partir de conjuntos de equações diferenciais inter-relacionadas [TANG, PARSONS e SKLAR 2006].

O MBA como técnica de modelagem de sistemas sociais está sendo estimulada pelo crescimento dos sistemas computacionais e é cada vez mais aplicada a sistemas anteriormente modelados com equações diferenciais [RAHMANDAD e STERMAN 2004]. Neste modelo, pode-se inserir um comportamento difuso para cada agente no sistema [TANG, PARSONS e SKLAR 2006]. Já os MBE são mais abstratos que MBA, pois eles representam o sistema como um todo através de uma média resumida da possibilidade de ação de determinado agente [COSTA, JEANNES e CAVA 2008].

No MBE, existem leis globais que se aplicam a todos os membros do estudo tornando os agentes homogêneos, tratados como iguais, assim eles podem não ser tão realistas devido a essa homogeneidade, no entanto, ele tem como vantagem o fato de

poder ser manuseado analiticamente. Entretanto nos últimos anos, os modelos vêm se tornando "mais realistas" e complexos tornando-se difícil de manusear analiticamente [NGUYEN et al 2012].

O MBE retrata bem os resultados a longo prazo sem a necessidade de realizar experimentos simulados [NGUYEN et al 2012]. Porém, fica difícil de visualizar individualmente o comportamento dos agentes e também o comportamento destes dentro de um ambiente. Com o MBA consegue-se visualizar os agentes individualmente, as interações entre eles e também simular interações entre agentes e o ambiente onde estes estão inseridos. Devido a isso o MBA é considerado mais flexível que o MBE [TANG, PARSONS e SKLAR 2006].

Então pode-se dizer que a escolha pelo uso do MBA ocorre por quatro principais motivos [TANG, PARSONS e SKLAR 2006]:

(I) É uma maneira natural de descrever sistemas compostos e interação entre entidades;

(II) São flexíveis;

(III) Facilidade de capturar fenômenos emergentes;

(IV) Fornecem maior nível de detalhes úteis em relação a simulação.

Outra vantagem que pode-se citar no uso do MBA é sua simplicidade na criação, facilidade de entendimento e, portanto, é mais acessível a modificações e ampliações tornando assim o modelo cada vez mais realista e detalhista [TANG, PARSONS e SKLAR 2006]. Alguns autores argumentam que o MBE e MBA são mais produtivos quando juntos em vez de serem consideradas como incompatíveis [COSTA, JEANNES e CAVA 2008].

3. Modelo de Lotka-Volterra

O estudo de dinâmicas populacionais é essencial para o entendimento do que ocorre nos ecossistemas. Populações de duas espécies diferentes podem se relacionar de diversas maneiras. Essas relações podem ser consideradas harmônicas, quando há benefício por pelo menos uma das espécies, ou podem ser desarmônicas, quando há prejuízo para uma ou ambas as espécies [ODUM, 1988]. A predação é um exemplo de relação que resulta em efeitos negativos no crescimento e sobrevivência de uma população e em um efeito positivo na outra.

Uma das ferramentas mais importantes na compreensão destas relações é a modelagem matemática. Para fazê-la, é necessário identificar características fundamentais do sistema a ser estudado, de maneira a obter um conjunto de regras matemáticas simples o suficiente para que se possa extrair informações úteis. Dependendo da natureza do problema, estas regras podem ser instituídas de diversas formas, através de equações diferenciais, equações algébricas, equações de diferenças, etc.

Alfred Lotka (1925) e Vito Volterra (1926) foram pioneiros em estudar interações entre presas e predadores. O modelo de Lotka-Volterra, mesmo com sua simplicidade, é muito representativo. Através de duas equações diferenciais ele consegue descrever o comportamento de dois tipos diferentes de populações, a presa e o predador.

Como citado anteriormente, o modelo Lotka-Volterra consiste em um par de equações diferenciais, e que frequentemente são utilizadas para representar a dinâmica populacional de sistemas biológicos. Segundo Malaquias e Mizukoski apud SOUZA (2007), o modelo de Lotka-Volterra é dado pelas equações (1) e (2):

$$\begin{cases} \frac{dx}{dt} = ax - cxy \quad (1) \quad (Corresponde\ a\ dinâmica\ da\ presa) \\ \frac{dy}{dt} = -bx + dxy \quad (2)\ (Corresponde\ a\ dinâmica\ do\ predador) \end{cases}$$

onde:

(I) *x* é a densidade de presas;

(II) *y* é a densidade de predadores;

(III) *a* é a taxa de nascimento das presas;

(IV) *b* é a taxa mortalidade dos predadores;

(V) *c* é a taxa de mortalidade das presas devido a relação com os predadores, esta também é denominada de coeficiente de predação;

(VI) *d* é a taxa de reprodução dos predadores por unidade de presas consumidas;

(VII) *t* representa o crescimento das duas populações ao longo do tempo.

Analisando as equações (1) e (2), é notável que cada encontro entre presas e predadores tende a inibir o crescimento da população de presas, pois representa sua captura, o que sugere o aumento de sua taxa de mortalidade, e ainda resulta no aumento do número de predadores, já que sua reprodução esta relacionada ao consumo.

4. NetLogo

A ferramenta escolhida para simular MBA foi o NetLogo (Figura 1), que é uma linguagem de programação agregada a um ambiente de desenvolvimento multiagente utilizada para simulações de fenômenos naturais e sociais. O NetLogo tem como vantagem os códigos que são simples e legíveis além de proporcionar uma interface intuitiva onde, pode-se inserir *sliders*, *ploters*, botões de ação e de escolha para controlar variáveis e parâmetros. Ainda pode-se visualizar graficamente os agentes interagindo entre si e com o ambiente.

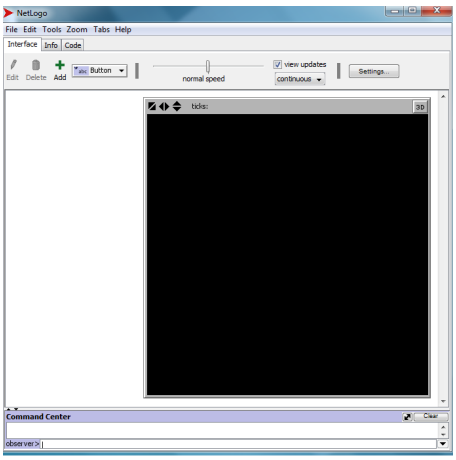


Figura 1. Interface do NetLogo

O ambiente do NetLogo é dividido em três abas (Figura 1): *Interface*, *Info* e *Code* cada aba tem suas próprias particularidades. A aba responsável pela criação da parte gráfica é a *Interface*. Nessa aba pode-se criar todo o ambiente de simulação.

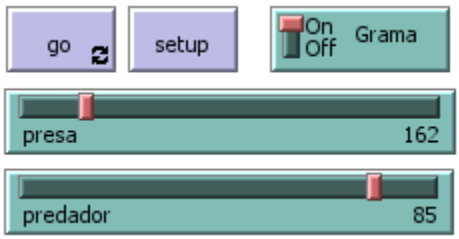


Figura 2. Exemplos de botões (*buttons*), chaves (*switches*), parâmetros (*sliders*).

Os itens que podem ser inseridos na interface são:

(I) *Buttom*: (Figura 2) responsável pelas ações dentro de nossa simulação;

(II) *Slider*: (Figura 2) usados para definir valores de parâmetros e variáveis do sistema;

(III) *Switche*: (Figura 2) são chaves de liga e desliga;

(IV) *Monitor*: (Figura 3) usado para monitorar valores textualmente em tempo de simulação;

(V) *Plots*: (Figura 3) usado para visualizar o estado de uma variável ao longo do tempo gerando gráficos de evolução.

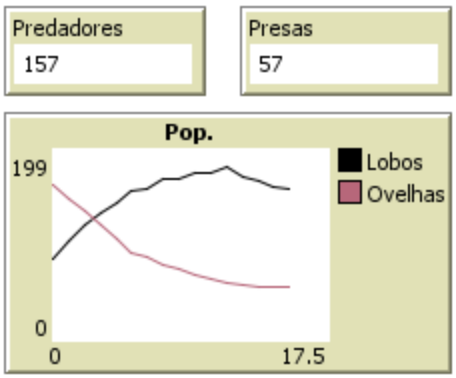


Figura 3. Exemplos de monitor (*monitor*), gráficos (*plot*).

Após a criação da interface, deve-se codificar a simulação. Esta etapa é realizada através de linguagem de programação. Os códigos são determinantes, pois são eles que ditam como cada agente irá se comportar no ambiente. Essa etapa é realizada na aba Code (Figura 4).

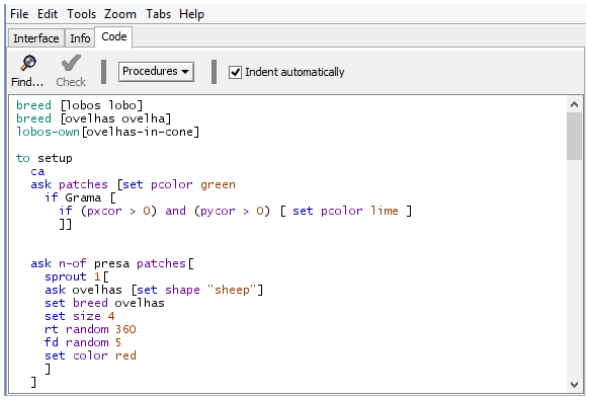


Figura 4. Exemplos de codificação no NetLogo.

A aba *Info* é responsável pela documentação do modelo criado. Nela podem ser inseridas as rotinas executadas no modelo, a finalidade, explicações sobre como usá-lo, futuras ampliações, entre outros.

5. Implementação

O modelo realizado, usando a linguagem NetLogo, será um MBA que simula as regras do modelo de Lotka-Volterra que é um MBE. Porém, o método de resolução desse modelo difere do método de resolução aplicado ao modelo de Lotka-Volterra. Enquanto no modelo de Lotka-Volterra é aplicado um método matemático como o método de Euler , o modelo criado se dá através da iteração entre os agentes ao longo do tempo, dentro de um ambiente pré-determinado.

Essa simulação busca representar o modelo de Lotka-Volterra pois, como visto anteriormente as equações de Lotka-Volterra são apenas em função do tempo sem nenhuma relação com espaço. E o NetLogo usa agentes como pontos dentro de um

espaço para representar o seu comportamento. Logo, o espaço é parte integrante da simulação e tem um papel importante para o resultado final.

A simulação tem como divisão temporal o *tick*, que é um intervalo de tempo dentro do ambiente de desenvolvimento NetLogo. A cada *tick,* os agentes irão realizar suas ações. Essas ações são predeterminadas no momento do desenvolvimento do código.

5.1. Agentes

Os agentes da simulação são divididos em duas classes: as presas e os predadores. Cada um deles terá sua atribuição no sistema conforme exposto abaixo:

5.1.1. Predador

Os predadores realizam basicamente duas funções: `caçarPresa` e `morrePredador`.

Na função de `caçarPresa`, o predador busca uma presa dentro de seu campo de visão e ao encontrá-la, ele começa a perseguição até estar sobre a presa, o que ainda não garante sua captura, pois o predador realiza um sorteio randômico de dois valores e se esses valores estiverem dentro do intervalo escolhido nas variáveis *k3* (taxa de predação) e *k4* (eficiência de predação), ele a captura e gera um novo predador em *patch* aleatório.

A morte dos predadores acontece ao chamar a função `morrePredador` que realiza o sorteio de um número randômico e se esse valor estiver dentro dos valores estipulados na interface na variável *k2* (taxa de mortalidade de predadores), ocorre a morte do agente atual.

5.1.2. Presa

A presa realiza duas funções a `nascePresa` e a `movePresa`.

Na função `nascePresa` é feito um sorteio de um número randômico e se esse número estiver dentro do intervalo escolhido na variável *k1* (taxa de nascimento de presas) é criada uma nova presa dentro de nosso sistema num *patch* aleatório.

Já na função `movePresa` é escolhido randomicamente uma direção qualquer e a presa anda um passo nessa direção.

5.2. Interface

A interface da simulação (Figura 5) é onde pode-se selecionar os parâmetros iniciais da simulação, como o número de presas iniciais, número de predadores iniciais, a taxa de nascimento de presas, a taxa de mortalidade dos predadores, a taxa de predação de lobos e a taxa de eficiência de predação. Além da seleção de parâmetros iniciais, também é possível visualizar o gráfico da população de presas e predadores, a quantidade textual atual de presas e predadores e uma interface gráfica mostrando o comportamento dos agentes dentro do ambiente.

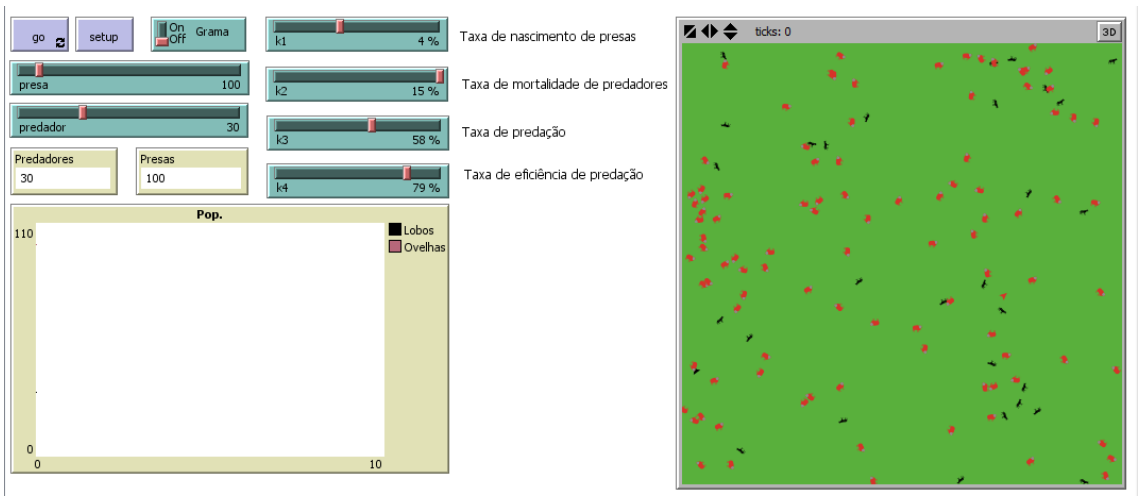


Figura 5. Interface do modelo Predador-Presa com a opção de grama desligada.

5.3 Patch

O *Patch* é cada área do terreno ocupado ou não por agentes, nesta simulação tem-se um ambiente de 201x201 *patches.* Inicialmente todos os *patches* são verdes e não interferem no comportamento dos agentes (Figura 5). Num segundo momento criam-se *patches* diferentes e assim modifica-se o comportamento dos agentes quando estiverem dentro destes *patches.* Essa modificação ocorreu em alguns *patches* que tinham uma cor mais escura e foi considerado que, dentro desses *patches,* os predadores teriam uma visão prejudicada como se fosse, por exemplo, uma vegetação fechada (Figura 6).

Esta funcionalidade foi inserida através da chave *Grama*. Quando a chave estiver ligada alguns dos *patches* serão mais escuros e modificarão o comportamento da caça dos predadores.

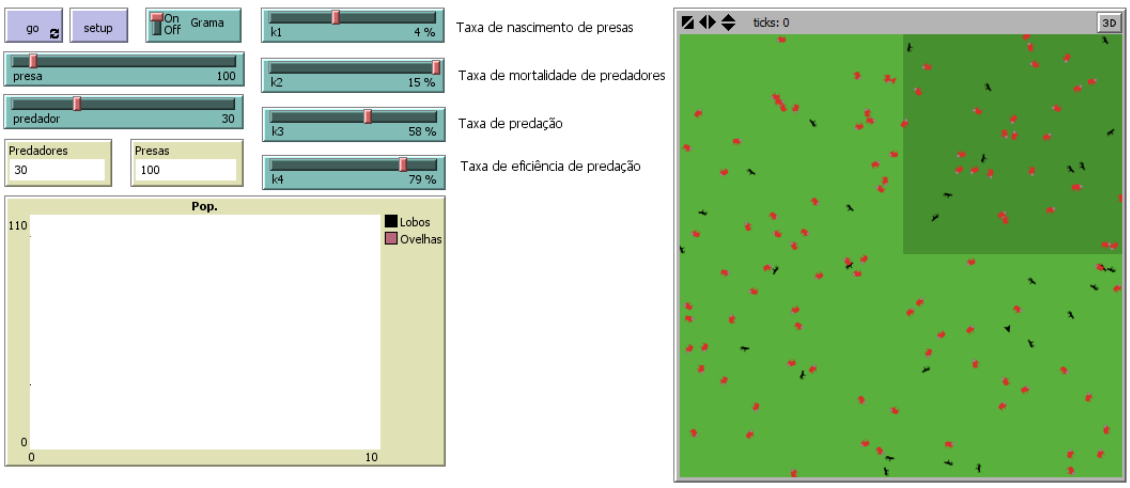


Figura 6. Interface de simulação Predador-Presa com a opção de grama ligada.

6. Resultado e Discussões

Os resultados obtidos através das simulações foram comparados com os resultados do sistema dinâmico obtidos através do método de Euler. Na figura 7 tem-se o resultado da simulação usando o MBA criado com NetLogo sem a opção *Grama* ligada. Enquanto que na figura 8 tem-se o resultado do MBE usando o método de Euler para a resolução do sistema de equações de Lotka-Volterra formado pelas equações (1) (2).

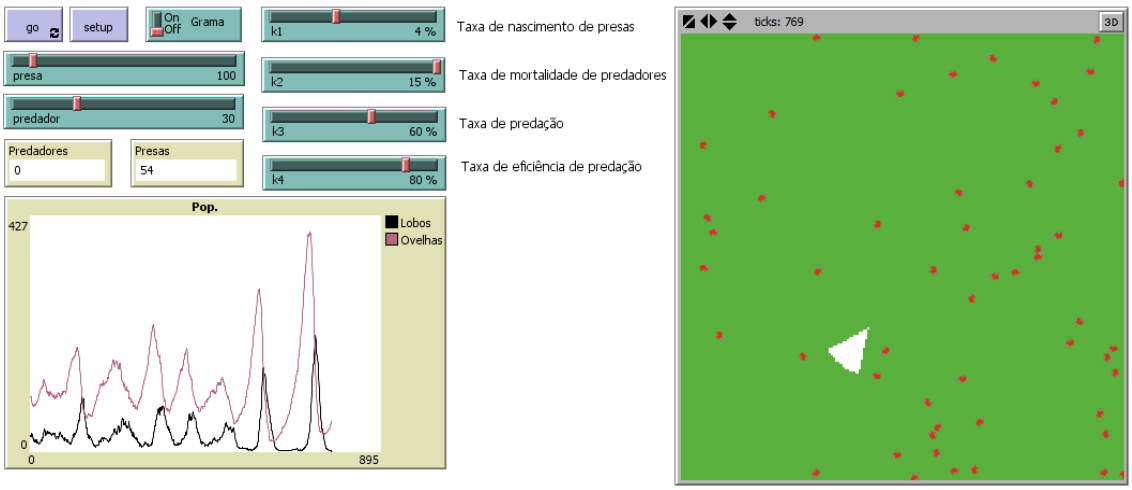


Figura 7. Simulação sem a utilização de ambiente diferenciado.

É importante verificar que o comportamento de ambas as simulações seguiu um padrão em relação às linhas de presas e predadores (na simulação lobos e ovelhas) ao longo do tempo. Apesar das pequenas mudanças ainda existe uma sinuosidade diferenciada que pode ter sido influenciada pela não previsibilidade de encontros e movimentos de presas e predadores dentro do ambiente delimitado que não é considerado quando trabalhamos com equações.

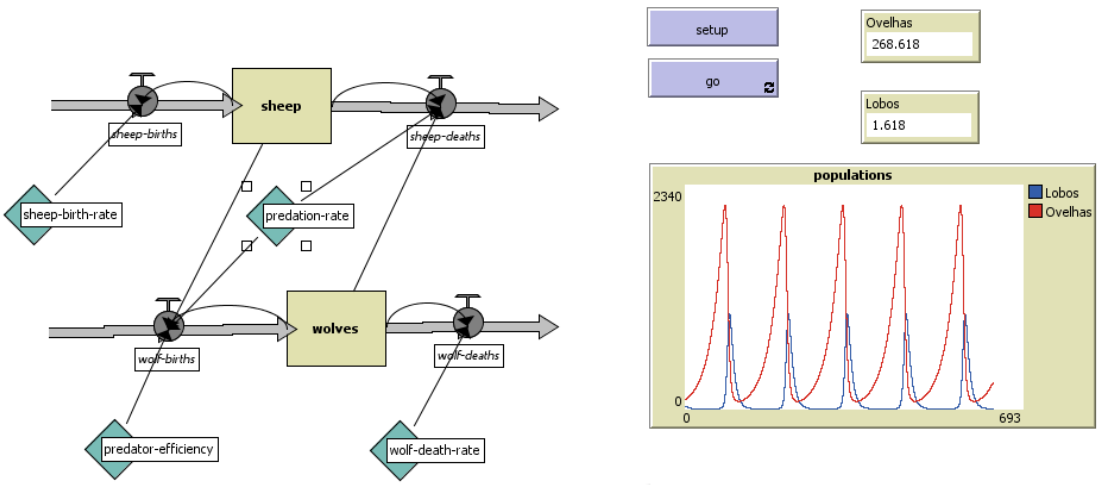


Figura 8. Simulação usando método de Euler.

Na simulação realizada com um ambiente diferenciado, onde uma porção do ambiente com vegetação fechada é evidenciado (Figura 9), pode-se ver também a semelhança entre os gráficos da simulação e pelo método de Euler, apesar de os resultados encontrados seguirem sempre um padrão no MBA, tem-se algumas modificações relacionadas ao ambiente.

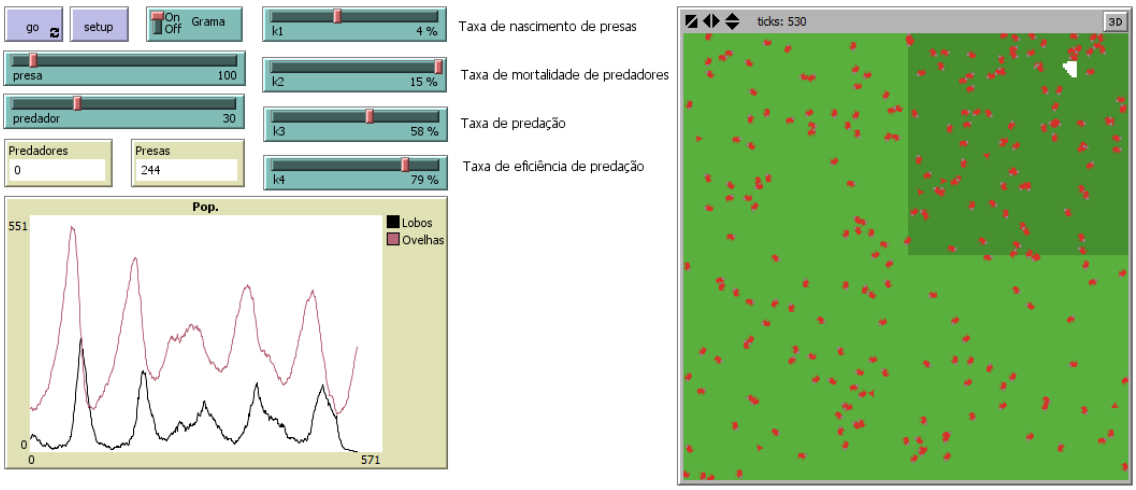


Figura 9. Simulação com a utilização de ambiente diferenciado.

Dentro das simulações baseadas em agentes consegue-se visualizar com mais perfeição alguns detalhes úteis. Um desses detalhes, vistos nos exemplos é que quando tem-se uma área onde a visão do predador é reduzida e a concentração de presas nesse ambiente aumenta (Figura 10) consegue-se verificar detalhes que não são possíveis quando utiliza-se MBE na simulação.

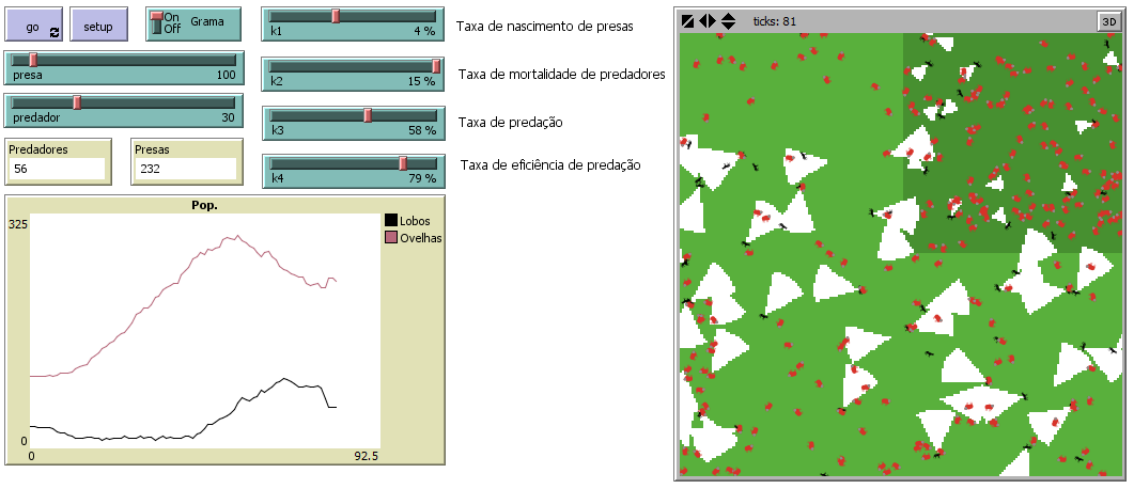


Figura 10. Simulação com a utilização de ambiente diferenciado.

Com esse exemplos, verifica-se que o MBA tem vantagens quando necessita-se de detalhes sobre a simulação, quando é preciso que a modelagem evolua com o tempo e inserção de novas variáveis e novos detalhes. Porém, nota-se que o resultado gráfico das simulações acaba sempre seguindo um mesmo padrão, o que nos leva a concluir que

o MBE consegue criar o comportamento geral da simulação, sem a necessidade da criação do modelo e inserção detalhada do funcionamento.

7. Conclusão

Ao avaliar o modelo predador-presa a partir de uma visão analítica (equações) e do modelo baseado em agentes, conseguiu-se observar o comportamento resultante das interações entre eles, foi possível identificar de forma clara que as oscilações dependiam dos parâmetros e o quanto isso influenciava no modelo. Constatou-se que o modelo de simulação multiagente é bastante simples pois, sua criação é realizada através da visualização do ambiente e descrição dos comportamentos dos agentes envolvido. Por outro lado, os de equações são mais complexos devido ao grande estudo matemático envolvido.

Também é possível observar que o MBE gera uma resposta mais abstrata, devido aos seus resultados serem para um cenário mais geral e utilizado para qualquer situação predador-presa em qualquer ambiente, já no MBA foi possível inserir mais detalhes da simulação, como o caso da vegetação mais fechada, a limitação da visão do predador e a forma de caça. Mesmo com essas diferenças os resultados encontrados foram muito semelhantes.

Dessa forma acreditasse que a escolha pelos modelos sejam baseados em equações ou em agentes depende do quão detalhado deseja-se o resultado final da simulação. Por fim, ressalta-se que é possível que as duas formas trabalhem associadas na busca por um resultado geral do MBE aliado ao detalhamento do MBA.

Referências

Adamatti, Diana. (2011) Simulação Baseada em Multiagentes como Ferramenta em Estudos Interdisciplinares. RENOTE, v. 9, n. 1.

Costa, Antônio CR; Jeannes, Fernanda M.; Cava, Ulisses A. (2010) Equation-based models as formal specifications of agent-based models for social simulation: preliminary issues. In: Social Simulation (BWSS), 2010 Second Brazilian Workshop on. IEEE, p. 119-126.

Momm, Deise R. C. (2012) Desenvolvimento de Robótica Inteligente em Ambiente Virtual para Múltiplos Robôs, Relatório Anuais, Departamento de Engenharia Elétrica, Puc-Rio, Rio de Janeiro.

Nadelhoffer, K. J. (2005) Predation and Parasitism. http://www.globalchange.umich.edu/globalchange1/current/lectures/predation/predation.html, 2005. Acesso em: 15/10/2014.

Nguyen, N. D., Taillandier, P., Drogoul, A., & Auster, P. (2012). Inferring equation-based models from agent-based models: a case study in competition dynamics. In Principles and Practice of Multi-Agent Systems(pp. 413-427). Springer Berlin Heidelberg.

Odum, Eugene P. (1988) Ecologia. Rio de Janeiro: Guanabara.

Parunak, H. Van Dyke; Savit, Robert; Riolo, Rick L. (1998) Agent-based modeling vs. equation-based modeling: A case study and users' guide. In:Multi-agent systems and agent-based simulation. Springer Berlin Heidelberg. p. 10-25.

Rahmandad, Hazhir; Sterman, John. (2008) Heterogeneity and network structure in the dynamics of diffusion: Comparing agent-based and differential equation models. Management Science, v. 54, n. 5, p. 998-1014.

Rebonatto, M. T. (2000) Simulação paralela de eventos discretos com uso de memória compartilhada distribuída. Master's thesis, Programa de Pós-Graduação em Computação, UFRGS, Porto Alegre, 2000. Mestrado em Ciência da Computação.

Souza, L.A. (2007) Sustentabilidade da pesca através da inclusão do homem em modelos predador presa: Um estudo de caso no Lago Preto. Dissertação (Doutorado em Ciência Biológicas). Programa de Pós-graduação e Biologia Tropical e Recursos Naturais do convênio INPA/UFAM.

Tang, Yuqing. Parsons, Simon. Skalar, Elizabeth. (2006) Modeling human education data: From equation-based modeling to agent-based modeling Proceedings of the Workshop on Multiagent-based Simulation, Hakodate, Japan.

Integrating a Tropos Modeling Tool with a MDA Methodology for Engineering Multi-agent Systems

João Victor Guinelli[1], Carlos Eduardo Pantoja[1], Ricardo Choren[2]

[1]CEFET/RJ – UnED Nova Friburgo – Av. Gov. Roberto da Silveira, 1900 – Nova Friburgo – RJ – Brasil

[2]Instituto Militar de Engenharia – IME/RJ – Pça Gen Tibúrcio 80 – Rio de Janeiro – 22270-290 – RJ – Brasil

jvguinelli@gmail.com, pantoja@cefet-rj.br, choren@ime.eb.br

Abstract. *This paper presents an integration between a Model-Driven Architecture (MDA) methodology for Multi-Agent System (MAS) development and the TAOM4E, which is a tool for graphical modeling that gives support to the Tropos methodology. Additionally, the MDA methodology uses the FAML, which is a metamodel that includes concepts of several agent-oriented methodologies (including Tropos) into a single model as Platform Independent Model and the JaCaMo metamodel, composed of Jason, Moise+ and CArtAgO, as Platform Specific Model. The methodology is also able to transform FAML concepts to JaCaMo concepts and generate code to the Jason/Moise+ from JaCaMo metamodel. The objective of this is paper is to allow the MAS designer to generate Jason/Moise+ code directly from a Tropos model using the proposed integration. The transformation set between the graphical modeling tool and the FAML metamodel was specified using the Query-View-Transformation language. The paper also presents an example using the integrated solution.*

1. Introduction

Engineering Multi-Agent Systems (MAS) is an effort to construct intelligent systems in distributed environment, capable of reasoning and communicating with other systems or intelligent agents. In last decade, several methodologies and approaches to design and develop MAS were presented in order to support the MAS engineering. Among these approaches, the Model-Driven Architecture (MDA) took an important role trying to fill the gap between several methodologies and modeling languages for MAS [Nunes et al., 2011].

The MDA provides a software construction divided in different and independent abstraction levels that are supported by metamodels. Applying metamodels in Model-Driven Development (MDD) allows a point of connection between the developed solution and existent solutions, since it is provided standardized specifications for metamodels such as Query-View-Transformation (QVT) [OMG, 2011]. There are several methodologies for engineering a MAS that are supported by MDD tools such as Tool for Agent Oriented Modeling for Eclipse platform (TAOM4E) [Morandini, 2008], for Tropos; and Prometheus Design Tool (PDT) [Sun et al., 2010] for Prometheus. Although these methodologies use a MDA approach, they constraint the developer in either modeling language or language specification. Since they do not use a generic metamodel,

the real potential of the MDA approach is minimized, once both are bound to their specific methodologies.

There is a MDA approach for MAS development [Pantoja; Choren, 2013] which uses a generic metamodel for MAS specification named FAML [Beydoun et al., 2009]. This approach allows the developer to choose a preferred design technology among the methodologies, modeling languages and notations adherent to FAML. After that, a set of QVT specifications to transform the FAML concepts into JaCaMo [Boissier et al., 2011] concepts in order to generate Jason [Bordini et al., 2007] and Moise+ [Hubner et al., 2002] codification. The methodology is not bound to JaCaMo because it can be extended to any specific programming language, since a cartridge with a new set of transformations (from FAML to this specific technology) can be developed. The methodology is also supported by a MDD tool. However, this tool only provides semi-automatic code generation (since it needs manual interventions in its metamodels) because there is no graphical environment associated with it in order to automatically instantiate these metamodels.

The objective of this paper is to integrate the MDA Tool TAOM4E with the MDA methodology [Pantoja; Choren, 2013] to provide a fully automatic code generation environment for Jason/Moise+ frameworks. For this, a new set of QVT transformations (between TAOM4E metamodel to FAML) is specified. An example using the integrated solution for eclipse is shown as proof for the QVT transformations. This paper is structured as it follows: in section 2 a background for the MDA methodology for engineering a MAS is presented; section 3 confers the integration between both MDD tools and the new set of QVT transformations; in section 4 an example for a conference submission system is shown; section 5 discusses some related work; section 6 presents some concluding remarks.

2. The MDA Methodology for Engineering MAS

The MDA methodology for MAS development presented in this section was firstly proposed by Pantoja and Choren (2013). In this methodology the MAS development is divided in different levels of abstraction and it uses Model-To-Model and Model-To-Text transformations to make possible the generation of MAS code for a Platform Specific model from the MAS specification. The proposed methodology is shown in the Figure 1.

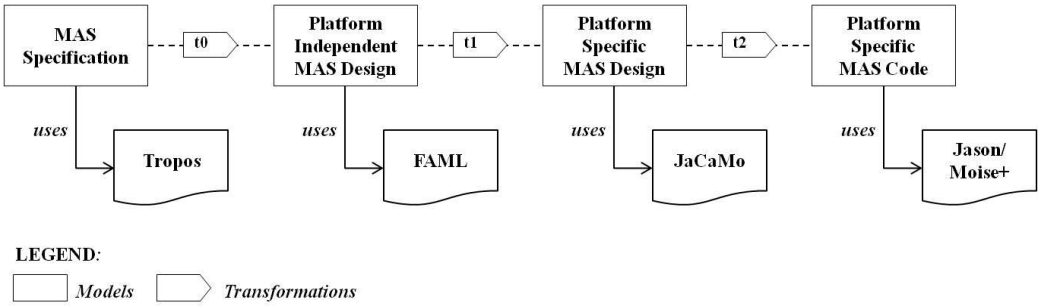


Figure 1. The MDA Methodology for MAS.

The core of this methodology is the platform independent MAS design level, which describes the MAS application in both perspectives internal level (agent) and external level (organization). One of the main purposes of this level is to provide a set of

generic concepts that can be useful for any agent-oriented modeling language and methodology like Tropos, Gaia, INGENIAS and Adelfe. To achieve this purpose it was created a metamodel that implements concepts present in the FAML that is a generic metamodel for SMA specification and development. The FAML unifies, inside the same software engineering domain for MAS development, different agent-oriented modeling languages.

Even though the FAML gives more flexibility to the development and specification of SMA, the concepts of model, system, environment and agents must be present in the modeling in order to the methodology works correctly. In the figure 2 it is possible to see how the internal level of an agent is structured in FAML, where the Agent, Mental State and Plan concepts are main roles.

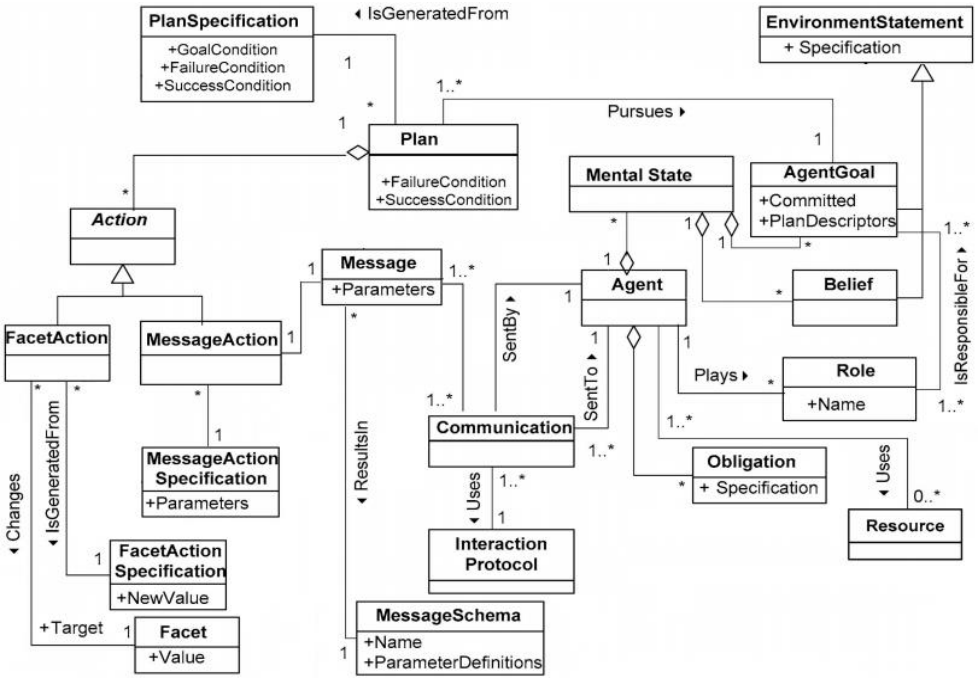


Figure 2. The FAML internal level metamodel.

The methodology starts from a MAS specification using any adherent FAML methodology (e.g. Tropos) which will create the Platform Independent MAS design after *t0* transformation. To transform the models created in the Platform Independent MAS design to the Platform Specific MAS there are the QVT transformations *t1*, which perform a Model-To-Model transformation that receive an instance of FAML model and return an instance of JaCaMo model. The JaCaMo metamodel is composed of concepts from both the agent programming and organizational programming. It uses the Jason and the Moise+ and can be edited before the *t2* transformation.

Some modifications were performed in the JaCaMo metamodel to become it more adaptable to the methodology. These modifications consist in a simplification of the environmental dimension and in a extension of the organizational dimension; they were necessary because the methodology uses the organizational model Moise+ for the organizational dimension, the agent-oriented language Jason for the dimension of the

agent and, in the environmental dimension, the standard Java from Jason to implement the environment where the agents are inserted.

As the methodology do not use artifacts to implement the agent environment, it uses only Java classes for it, it is not necessary to use the CArtAgO , so the simplification of the environmental dimension consisted in to keep the *Environment* class to represent the environment, the *Percept* class to represent the perceptions and in to not use the CArtAgO.

Besides, the extension in the organizational dimension was performed adding the class *Link* in the metamodel and creating some self-relationship like *supertype*, in the class *Role*; *monitoringScheme*, in the class *Scheme*; and *subGoal*, in the class *Goal*. All these modification intent to structure the organizational Moise+ coding file considering the functional, structural and normative specifications.

The class *Link* was added to represent the relationship between groups and roles of the group specification from structural model. Moreover the *monitoringScheme* was created to permit the identification of monitoring scheme from each existing scheme, whereas the self-relationships *supertype* and *subGoal* were created to represent the role hierarchy and the goals hierarchy from the functional specification, respectively. The figure 3 shows the JaCaMo metamodel adapted to the methodology.

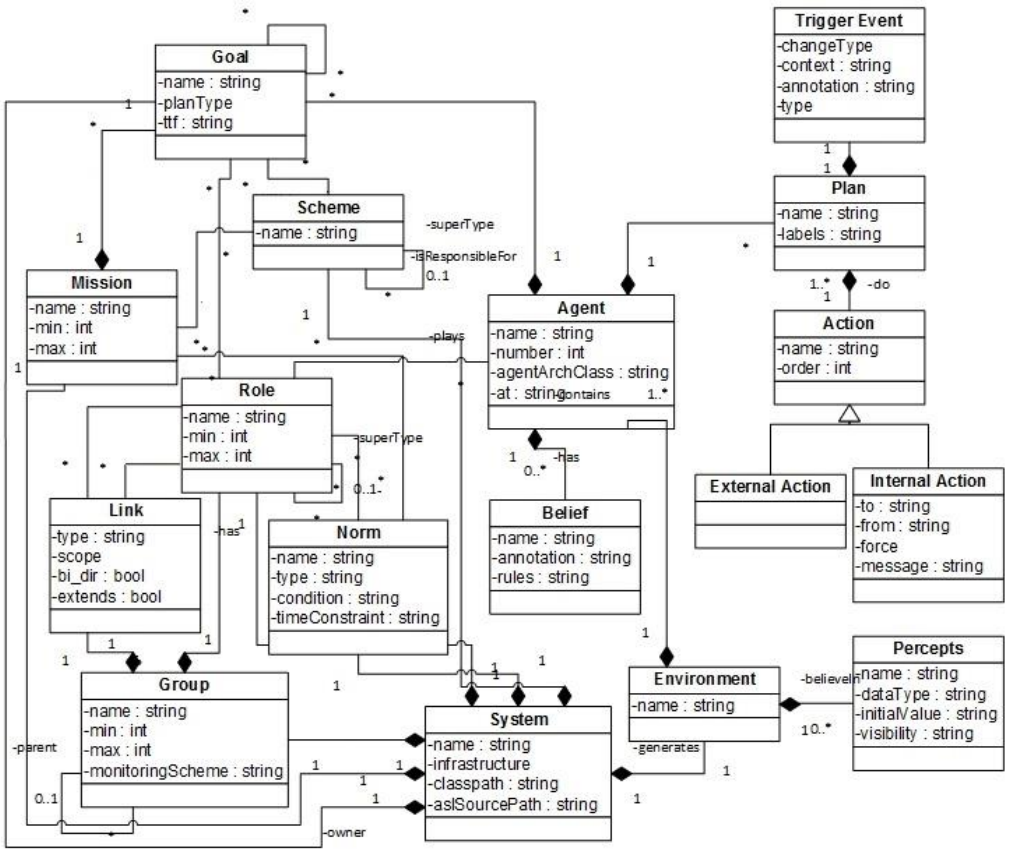


Figure 3. The Jacamo metamodel.

Finally, the transformations represented by *t2* are responsible to generate the execution code for the platform Jason and Moise+. These transformations are implemented in the M2T language, which is an OMT standard that uses templates to generate text artifacts from models, and receive as input an instantiated JaCaMo model. In addition, these transformations also contain an extension that enable the code generation for Unmanned Aerial Vehicle using AgentSpeak (UAVAS) [Hama et al., 2011], which is a Jason extension.

The UAVAS uses the Jason to program its agents, but it uses a different form to send and receive messages. Whereas in the standard Jason is used the internal actions *.send* and *.broadcast* for communicating, the Jason from UAVAS uses the external actions *request*, *inform*, *ask* and *ack*. So the transformation will generate code in two different ways, it depends if the modeled system is a Jason or UAVAS system.

3. The TAOM4E Integration to FAML Metamodel

In this section, a new set of transformations between the metamodel (*t0* transformation) used by TAOM4E and the FAML metamodel is presented (the hierarchical set can be seen in figure 4). The TAOM4E metamodel is divided in two models: the *Core*, which holds the Tropos modeling concepts; and the *View*, which defines the graphical elements of the diagram. The *Core* metamodel consists of all elements instantiated in a graphical modeling project (e.g. actors, hard goal, soft goals, resources, etc.). The transformation rules are based on mapping all those instanced concepts in *Core* metamodel to a relative concept on FAML metamodel (observing the Tropos-to-FAML adherence [Beydoun et al., 2009]).

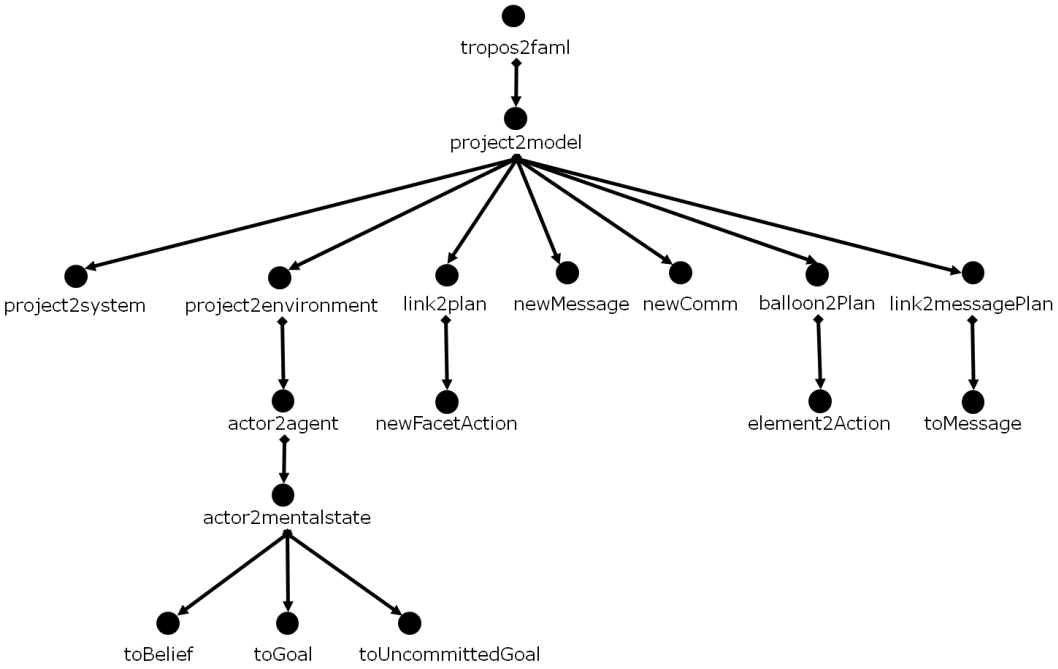


Figure 4. The transformation set from Tropos Tool to FAML metamodel.

So the transformation starts from a TAOM4E modeling project, which is mapped into a FAML project. Likewise, the Tropos model (which gathers Tropos concepts) is mapped to a FAML model. This first step of the transformation is just to create the base to hold the main concepts of the modeling. After that, the graphical components begin to be mapped. The Tropos project generates the FAML system (including references to the MAS environment). This same project also generates the environment, which maps all actors and its mental states. In FAML, a mental state is composed of beliefs and goals (that an agent can be committed or not). So, a Tropos *Resource* concept is mapped to a FAML *Belief* concept and the Tropos *Hard Goal* concept is mapped to a FAML *Agent Goal* concept. To decide if an agent is committed or not to a goal, it is observed the hard goals that an agent holds at modeling. The *Soft Goals* concepts are not materialized into code, so it was decided to not map it (but the FAML provides concepts for this kind of goals). The *actor2mentastate* transformation can be seen in figure 5.

The next steps map all plans and actions from the modeling project. Basically, the FAML has two types of actions that guide the plans mapping: a *Message Action*, which is an action of exchanging messages between agents; and a *Facet Action*, which gathers all other kind of actions (e.g. environment and reasoning actions). In the modeling project any *Hard Goal* or *Resource* that is sent to another agent becomes a message plan (specific plans containing only messages from an agent to another), otherwise, they are mapped as simple plans (plans containing the other type of actions). When a message plan is being mapped, it is necessary to create a communication concept because in FAML messages concepts only exists into a communication concept (which holds the message and all agents involved).

```
transformation actor2mentalstate(in troposModel : tropos, out famlModel : faml );

mapping model::informalcore::Actor :: actor2mentalstate() : faml::MentalState {
init { log("Transforming a Tropos Actor concept into a FAML MentalState concept.");}
    result.believeIn += self.outcomingRelations.targets[model::informalcore::Resource]->map resource2belief();
    result.hasObjective += self.outcomingRelations.targets[model::informalcore::Resource]->map resource2uncommittedgoal();
    result.hasObjective += self.outcomingRelations.targets[model::informalcore::HardGoal]->map hardgoal2goal();
    result.hasObjective += self.incomingRelations.targets[model::informalcore::HardGoal]->map hardgoal2uncommittedgoal();
    result.hasObjective += self.incomingRelations.targets[model::informalcore::Resource]->map resource2goal();
    //mapping opened balloons
    result.hasObjective += self.ownedElements[model::informalcore::HardGoal]->hardgoal2uncommittedgoal();
end { log("Mapping actor2mentalstate finished.");}
}
```

Figure 5. The *actor2mentalstate* QVT transformation.

The exploded balloons of the modeling project are mapped to simple plans, where every hard goal generates a FAML plan. In this case, a *Hard Goal* can have a link decomposition. Each decomposition, then, is mapped as a FAML *Facet Action* (the difference between a *AND* and a *OR* link is done in the platform specific model). Finally, the activities' diagrams complete the action definitions for a simple plan. After all, a complete and instantiated FAML model is provided, and the methodology can go on with subsequent transformations *t1* (from FAML to JaCaMo) and *t2* (from JaCaMo to Code).

4. A Simple Modeling Example

In this section it is presented a simple example for a Conference Management System (CMS) [DeLoach, 2002]. First it is used the TAOM4E for modeling the CMS (for Early Requirements phase) which can be seen in figure 6.

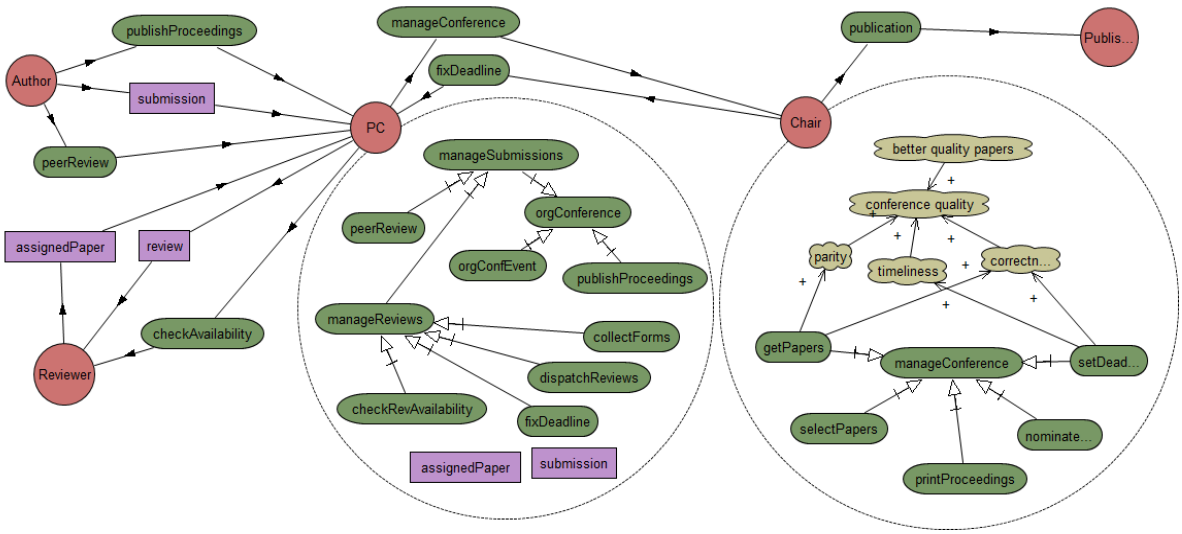


Figure 6. The TAOM4E modeling example of a Conference Management System [Morandini et al., 2008].

Second, it is necessary to perform the set of QVT transformations from Tropos to FAML (*t0* transformation) which will generate a model with the mapped concepts present in CMS modeling. These transformations were implemented using the Model-to-Model (M2M) project for eclipse. Afterwards, it is necessary to select the specific platform (executing the *t1* transformation from FAML to JaCaMo). It will generate a JaCaMo model instantiated from equivalent FAML concepts. The FAML and JaCaMo instances is shown in figure 7.

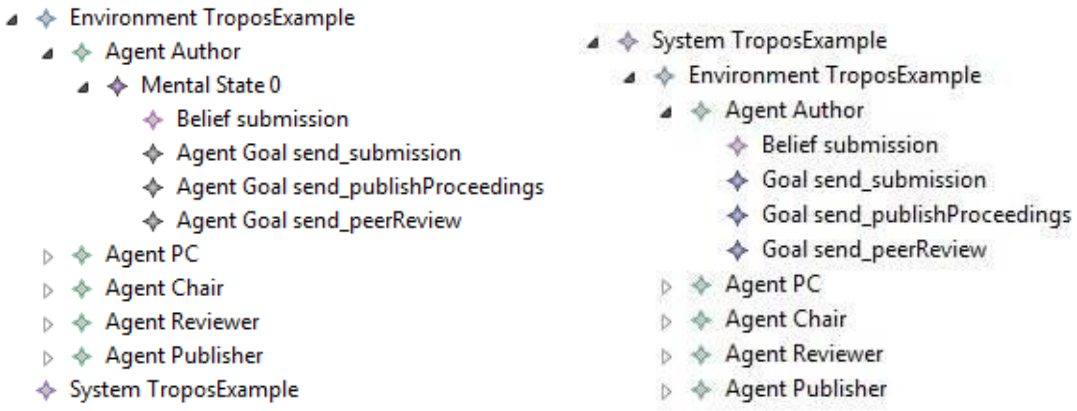


Figure 7. The instances of FAML (left) and JaCaMo (right) for the environment.

Finally, the *t2* transformation generates the code for Jason/Moise+ after a set of Model-To-Text (M2T) from the JaCaMo model. The generation provides files for the system (mas2j), environment (Java), agents (asl) and organization (xml). The agent PC code can be seen in figure 8.

```
review.

!send_checkAvailability.
!send_manageConference.

+!orgConfEvent: true <-
    orgConfEvent.

+!send_review: true <-
    .send(Reviewer,achieve,review).

+!manageSubmissions: true <-
    !peerReview;
    !manageReviews.

+!send_checkAvailability: true <-
    .send(Reviewer,achieve,checkAvailability).

+!dispatchReviews: true <-
    dispatchReviews.

+submission: true <-
    submission.

+!checkRevAvailability: true <-
    checkRevAvailability.

+!publishProceedings: true <-
    publishProceedings.

+assignedPaper: true <-
    assignedPaper.

+!collectForms: true <-
    collectForms.

+!fixDeadline: true <-
    fixDeadline.

+!manageReviews: true <-
    !checkRevAvailability;
    !fixDeadline;
    !dispatchReviews;
    !collectForms.

+!orgConference: true <-
    !manageSubmissions;
    !publishProceedings;
    !orgConfEvent.

+!send_manageConference: true <-
    .send(Chair,achieve,manageConference).

+!peerReview: true <-
    peerReview.
```

Figure 8. The PC code in Jason.

5. Related Works

In this section it is discussed two related works that use MDA for designing MAS: the PDT and TAOM4E. Both solutions use the Eclipse Modeling Framework (EMF) to provide a graphical environment for MAS design. The EMF provides a software construction centered in metamodels (named Ecore), which can be interconnected with other eclipse solutions such as Object Constraint Language, Model-to-Model and Model-to-Text transformation. The proposed integration uses the EMF as part of the tool develop, so allowing interconnection with any other solution.

The PDT is a tool for the Prometheus methodology that generates code for JACK agent-oriented language. Although the PDT can be integrated to the Islander methodology, the tool still depends on Prometheus Ecore Metamodel (PEMM) and any methodology that needs to be integrated has to be compliant with it. The MDA methodology used in this work uses the FAML, which is a generic metamodel for several methodologies (including Prometheus and Islander) enabling the methodology integration with any tool adherent to FAML. It is assumed the possibility to integrate the PDT to this MDA methodology (using a new set of QVT transformation) taking advantage of both solutions.

The TAOM4E is tool based on Tropos methodology and generates code for both JACK and JADE/JADEX. In the same way, the tool is tied to a specific metamodel, limiting the MDA possibilities. Integrating the TAOM4E with this MDA methodology extends the amplitude of both solutions, allowing code generation for Jason/Moise+ (for TAOM4E), and using a graphical modeling environment to design and engineering MAS (for this MDA methodology).

6. Conclusion

This paper presented an integration between a MDD tool for modeling MAS using the Tropos methodology and a MDA methodology for engineering MAS which uses a generic metamodel for several extant methodologies. Yet, this methodology generates code for Jason/Moise+. This integration allows automatic code generation directly from a graphical modeling environment using Tropos methodology, which is a well-known approach for MAS design. This process is supported by a MDA methodology that transforms the initial modeling concepts until the MAS skeleton be generated (ensuring that all initial concepts will be presented in the final software system). Moreover, the use of metamodel enables several point of connections for existent solutions (since FAML is generic for several methodologies). The MDA methodology integrated to the TAOM4E do not aims to provide another solution for design and engineering MAS, instead, it allows to truly use the real power promised by the MDA, that is to integrate solutions and transform different abstraction models until get a software implementation.

For future work it is necessary to refine the QVT transformations applying some OCL rules in order to generate constrained models; besides of the use of JaCaMo metamodel, the solution just generates code for Jason/Moise+. So it is also necessary to extend the code genera-tion to CArtAgO, and integrate the MDA methodology to the PDT providing a methodology selection using a centered and generic metamodel.

7. References

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomezsanz, J. J., Pavon, J. e Gonzalez-Perez, C. (2009) "FAML: a generic metamodel for MAS development." IEEE Trans. Softw. Eng.

Boissier, O., Bordini, R. H., Hubner, J. F., Ricci, A. e Santi, A. (2011) "Multi-agent oriented programming with jacamo" Science of Computer Programming.

Bordini, R. H., Hubner, J. F. e Wooldridge, W. (2007) "Programming Multi-Agent Systems in AgentSpeak using Jason" Jonh Wiley and Sons, London.

DeLoach, S. A. (2002) "Modeling organizational rules in the multi-agent systems engineering methodology". In R. Cohen and B. Spencer, editors, Canadian Conference on AI, volume 2338 of Lecture Notes in Computer Science, pages 1–15. Springer, 2002.

Hubner, J. F., Sichman, J. S. A. e Boissier, O. (2002) "A model for the structural, functional, and deontic specification of organizations in multiagent systems", In Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence, SBIA '02, London, UK. Springer-Verlag.

Hama, M. T. ; Allgayer R. S. ; Pereira, C. E. ; Bordini, R. H. (2011) "UAVAS: An Agent Oriented Infrastructure for Unmanned Aerial Vehicles Development" In: AutoSoft@CBSoft, 2011, São Paulo. II Workshop sobre Sistemas de Software Autônomos. São Paulo: CBSoft, v. 10. p. 15-21.

Morandini, M., Nguyen, D. C., Perini, A. e Siena, A. Angelo Susi (2008) "Tool-Supported Development with Tropos: The Conference Management System Case Study". In: Luck, M., Padgham, L. Agent-Oriented Software Engineering VIII.

Lecture Notes in Computer Science Volume 4951, 2008, pp 182-196. Springer, Germany (2008).

OMG (2011). Meta object facility (MOF) Query/View/Transfomation specification.

Nunes, I., Cirilo, E., Lucena, C. J. P., Sudeikat, J., Hahn, C. e Gomez-Sanz, J. J. “A survey on the implementation of agent oriented specifications”. In: Gleizes, M., Gomez-Sanz, J. J. (eds.). Agent-Oriented Software Engineering X, pp. 169-179. Springer, Germany (2011).

Pantoja, C. E. e Choren, R. (2013) “A MDA Methodology to Support Multi-Agent System Development” In: Proceedings of 5th International Conference on Agents and Artificial Intelligence: volume 1, ICAART'13, Barcelona.

Sun, H., Thangarajah, J. e Padgham, L. (2010) “Eclipse-based prometheus design tool” In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1, AAMAS '10, Richland, SC.

Engenharia de Software Orientada a Agentes: Um Estudo Comparativo entre UML e Metodologias que Suportam o Processo de Desenvolvimento de Sistemas Multiagente

Rafhael R. Cunha [1], Diana F. Adamatti[1], Cléo Z. Billa[1]

[1]Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)
Av. Itália km 8 – Bairro Carreiros – Rio Grande - RS - Brasil

{rcrafhaelrc,dianaada,cleo.billa}@gmail.com

Resumo. *Este artigo pertence ao domínio da Agent Oriented Software Engineering (AOSE) e Sistemas MultiAgente (SMA). O trabalho apresenta brevemente as metodologias Prometheus, Tropos, MaSE e Ingenias, elencando suas etapas e características de desenvolvimento. O objetivo deste trabalho é analisar as metodologias exploradas, comparando-as com os artefatos presentes na Unified Modelling Language (UML). Ao final deste trabalho, conclui-se que cada uma das metodologias analisadas possui enfoques distintos, atribuindo ao projetista a tarefa de escolher a que melhor se adeque as suas necessidades.*

1. Introdução

No decorrer dos anos, a área de engenharia de software (ES) tem procurado definir processos de desenvolvimento de software e linguagens de modelagem que visem estabelecer etapas bem definidas para a construção de um software. Este fato tem como propósito tornar a produção de um software mais robusta, rápida, organizada, confiável e de fácil manutenção e reutilização [Guedes 2012].

Na área de inteligência artificial, o paradigma orientado a agentes tem sido pesquisado e utilizado para minimizar a complexidade e aumentar a eficiência de softwares distribuídos [Jayatilleke et al. 2007] [Rodriguez et al. 2011]. Esta prática tem se mostrado eficiente para a construção de softwares com essas características, viabilizando um aumento no desenvolvimento de sistemas multiagente (SMA) [Guedes 2012].

Entretanto, devido à complexidade e distribuição desse tipo de sistema, novos desafios para a área de ES tradicional foram encontrados. Esses desafios ocorreram em virtude de metodologias conceituadas para se fazer modelagem na área, como *Unified Modelling Language* (UML), não suprirem as características presentes neste grupo de sistemas. Essa ausência levou ao surgimento de uma subdivisão da área que mescla conceitos de engenharia de software e inteligência artificial, chamada de *Agent Oriented Software Engineering* (AOSE) ou Engenharia de Software Orientada a Agentes. Seus objetivos principais são propor métodos e linguagens/notações para projetar e modelar softwares orientados a agentes [Guedes 2012].

Dentro desse contexto, diversas metodologias foram propostas buscando suprir a demanda de softwares orientados a agentes [Bergenti et al. 2004]. Essas metodologias foram criadas pelos mais variados motivos. Algumas, basearam-se em melhorias nos diagramas presentes na UML, outras, criaram seus próprios meta-modelos e notações para seu uso.

Este trabalho tem como motivação apresentar uma revisão de algumas metodologias para construção de sistemas multiagente, mostrando as etapas presentes nessas metodologias e os artefatos utilizados para apoiar essas etapas. Por fim, como objetivo principal na realização deste trabalho, pretende-se fazer uma análise comparativa, especificando a aproximação dos artefatos gerados por estas metodologias em relação aos presentes na linguagem UML.

2. Metodologias AOSE

Segundo [Brandão 2014], as metodologias existentes até o presente momento para modelar SMA sobre a perspectiva da engenharia de software são as ilustradas pela figura 1.

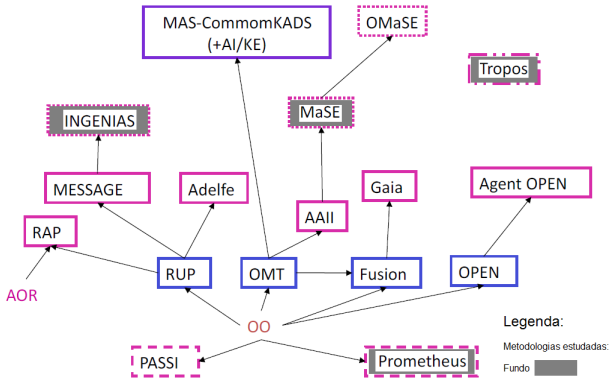


Figura 1. Metodologias AOSE [Brandão 2014]

A escolha das metodologias exploradas neste trabalho baseou-se na seleção de algumas metodologias distintas que compõem ramos heterogêneos da figura 1.

2.1. Prometheus

Segundo [Padgham and Winikoff 2005] e [Khallouf and Winikoff 2009] Prometheus é uma metodologia que consiste em três fases: A fase de especificação do sistema; A fase de projeto arquitetural e a fase de projeto detalhado.

A fase de especificação do sistema é a primeira fase da metodologia Prometheus. Essa fase é composta por duas etapas: Determinar o ambiente do sistema (percepções e ações) e determinar os objetivos e funcionalidades do sistema (objetivos e cenários de casos de uso). De acordo com [Padgham and Winikoff 2002], simultaneamente ao descobrimento ou especificação das percepções e ações, o usuário da metodologia deve começar a descrever quais são as funções do agente em um contexto mais amplo, ou seja, quais são as funcionalidades do sistema.

Na fase de projeto arquitetural, toma-se a principal decisão acerca do sistema [Padgham and Winikoff 2002]. De acordo com [Padgham and Winikoff 2002], é nesta fase que se define quais agentes devem existir no sistema a ser modelado. Todavia, o diagrama crucial desta etapa de desenvolvimento é o diagrama de visão geral do sistema. Neste diagrama são reunidos agentes, eventos e objetos de dados compartilhados.

O objetivo final dessa etapa da metodologia é especificar completamente a interação entre os agentes. Para tanto, são utilizados os diagramas de interação, como

uma ferramenta inicial para modelar essa interação. Para a evolução dessas interações entre os agentes, [Padgham and Winikoff 2005] avançaram essas interações para protocolos de interação, tendo como finalidade definir precisamente quais sequências de interações são válidas dentro do sistema.

De acordo com [Padgham and Winikoff 2002] a fase do projeto detalhado foca no desenvolvimento da estrutura interna de cada um dos agentes e como os mesmos irão realizar suas tarefas dentro do sistema modelado. É nesta fase do projeto que a metodologia se torna específica para agentes baseados na arquitetura BDI. Outro ponto de destaque dessa fase é a definição de capacidades (módulos dentro dos agentes), eventos internos, planos e estruturas de dados detalhados.

[Padgham and Winikoff 2005] citam que a metodologia Prometheus não se baseou na notação presente na UML para a criação de seus diagramas, gerando dois pontos distintos. O ponto positivo consiste na descentralização de agentes em objetos, favorecendo a modelagem desse tipo de sistemas. Sobre o ponto negativo, este é identificado em razão da não utilização de uma notação padrão, o que contrariamente, favoreceria a metodologia visto que uma padronização contribuiria para a sua aceitação.

2.2. Tropos

Segundo [Bresciani et al. 2004], a metodologia Tropos tem como propósito apoiar todas as atividades de análise e projeto do desenvolvimento de software. Tropos consiste de cinco fases distintas no seu processo de desenvolvimento, sendo: Requisitos Iniciais; Requisitos Finais; Projeto Arquitetural; Projeto Detalhado e Implementação.

O objetivo da análise de requisitos em Tropos é fornecer um conjunto de requisitos funcionais e não funcionais para o sistema [Bresciani et al. 2004]. Essa análise é dividida em duas fases: Análise dos Requisitos Iniciais e Análise dos Requisitos Finais.

Na primeira fase de análise dos requisitos, o usuário da metodologia identifica os agentes de domínio e modela-os como atores sociais, que dependem uns dos outros para atingirem os objetivos, planos e compartilharem dos mesmos recursos. Na segunda fase, o modelo conceitual é estendido de modo a incluir um novo ator, o qual representa o sistema, além de uma série de dependências com outros atores do ambiente.

[Bresciani et al. 2004] afirmam que o projeto arquitetural e as fases de projeto detalhado se concentram na especificação do sistema, em maneiras de garantir as exigências resultantes das fases anteriores. O projeto arquitetural define a arquitetura global do sistema em termos de subsistemas, interligados através de dados e fluxos de controle. O projeto arquitetural também prevê um mapeamento dos atores do sistema para um conjunto de agentes de software, cada um caracterizado pelas capacidades específicas.

A fase do projeto detalhado visa especificar as capacidades dos agentes e suas interações. Neste ponto, geralmente a plataforma de aplicação já foi escolhida, possibilitando a construção de um projeto detalhado que irá mapear diretamente o código.

A etapa de implementação segue passo a passo, de uma forma natural, a especificação do projeto detalhado com base no mapeamento estabelecido entre as construções da plataforma de execução e as noções do projeto detalhado [Bresciani et al. 2004].

2.3. MaSE

A metodologia *Multiagent Systems Engineering* (MaSE) foi proposta por [DeLoach et al. 2001]. É uma metodologia desenvolvida para análise e projeto de sistemas multiagente. MaSE utiliza da abstração fornecida por SMA para ajudar projetistas a desenvolver sistemas de software distribuídos inteligentes [Henderson-Sellers and Giorgini 2005].

A metodologia MaSE consiste de duas fases principais que resultam na criação de um conjunto de modelos complementares [Henderson-Sellers and Giorgini 2005]. A primeira fase é análise e a segunda é projeto.

A primeira fase da metodologia inclui três etapas: captura de objetivos, aplicação de casos de uso e refinamento de papéis. A segunda fase é composta por quatro etapas: criação de classes de agentes, construção das conversações, montagem das classes dos agentes e o projeto do sistema.

A finalidade da primeira etapa da fase de análise é capturar os objetivos do sistema extraindo-os da especificação inicial do sistema. Após a extração, os objetivos são estruturados em um diagrama chamado de hierarquia de objetivos. Posterior a elaboração deste diagrama, o analista deve associar cada objetivo com um papel e um conjunto de classes de agentes responsáveis para satisfazer esses objetivos.

Na etapa de aplicação de casos de uso, o analista transforma os objetivos e sub-objetivos em casos de uso. Esses casos de uso são descritores narrativos de uma sequência de eventos que define um comportamento desejado para o sistema. Além disso, a partir dos casos de uso é possível desenvolver um conjunto de diagramas de sequência que são semelhantes aos utilizados na UML.

A terceira etapa presente na fase de análise da metodologia MaSE é o refinamento de papéis. Esta etapa tem como propósito garantir que foram identificados todos os papéis necessários no sistema e desenvolver as tarefas que definem o comportamento dos papéis e os padrões de comunicação.

A primeira etapa da fase de projeto do sistema é chamada de criação de classes de agentes. Nesta etapa, as classes de agentes são identificadas a partir dos papéis refinados e são transcritas em um diagrama de classes de agentes. Além disso, são identificadas as conversas em que diferentes classes de agentes devem participar. Tais conversas são geralmente derivadas de comunicações externas dos papéis atribuídos ao agente. Nesse contexto, o projetista pode combinar papéis múltiplos em uma única classe de agentes ou mapear uma única função para ser exercida por classes de agentes múltiplos, objetivando eficiência e simplificação.

A próxima fase presente na metodologia é chamada de construção de conversações. Até o início dessa etapa, o projetista apenas identificou as conversas. O objetivo desta etapa é definir os detalhes dessas conversas, baseados nos detalhes internos de tarefas simultâneas.

Os elementos internos dos agentes são projetados durante a etapa de montagem das classes dos agentes, que inclui duas sub-etapas: a definição da arquitetura de agentes e a definição dos componentes da arquitetura. Analistas/projetistas tem a escolha de projetar sua própria arquitetura ou usar arquiteturas pré-definidas, como BDI. Da mesma

forma, um projetista pode usar componentes pré-definidos ou desenvolvê-los a partir do zero. Segundo [Bergenti et al. 2004], componentes consistem de um conjunto de atributos, métodos, e possivelmente uma sub-arquitetura.

Projeto do sistema é o último passo da metodologia MaSE. Nessa etapa é utilizado o diagrama de implantação para demonstrar os números, tipos e locais das instâncias dos agentes no sistema.

2.4. Ingenias

Segundo [Henderson-Sellers and Giorgini 2005], a metodologia Ingenias fornece uma notação para modelagem de SMA através de uma coleção bem definida de atividades, para orientar o processo de desenvolvimento. Especificamente, nas tarefas de análise, projeto, verificação e geração de código. A notação Ingenias, baseia-se em cinco metamodelos que definem os diferentes pontos de vista e conceitos a partir dos quais um SMA pode ser descrito.

2.4.1. Processo Ingenias

Segundo [Henderson-Sellers and Giorgini 2005], o processo Ingenias ajuda os desenvolvedores a produzir uma especificação SMA e sua implementação.

Sobre os pontos de vista, cada um deles é construído através de conjuntos de atividades estruturados em diagramas de atividades.

Um conjunto visa elaborar uma visão no nível de análise, enquanto o outro se concentra no projeto. No total, um desenvolvedor tem dez diagramas de atividades que propõem uma centena de atividades. Essas atividades foram numeradas para ajudar os desenvolvedores na sua aplicação. Sua enumeração é única dentro de um ponto de vista.

Durante a análise, o foco é sobre a definição de um modelo de organização que esboce a estrutura do SMA e identifique seus principais componentes. Este resultado, o equivalente a uma arquitetura SMA inicial, mais tarde é refinado, identificando os objetivos da organização e tarefas relevantes em relação a esses objetivos. Essas tarefas posteriormente são executadas por cada agente na organização.

Na fase de elaboração do projeto, mais detalhes são adicionados ao definir fluxos de trabalho entre os diferentes agentes, completando a definição de fluxo de trabalho com as interações do agente, e refinando o estado mental do agente, como consequência.

Ingenias baseia a sua proposta de implementação nas facilidades do *Ingenias Development Kit* (IDK) em mapear elementos de especificação em entidades computacionais. No domínio de agente, entidades computacionais relevantes poderiam ser arquiteturas de agentes, arquiteturas SMA, ou plataformas de agentes. Este mapeamento é realizado na IDK por módulos.

Segundo [Henderson-Sellers and Giorgini 2005], atualmente Ingenias apoia a verificação e validação utilizando a *activities theory* (AT). No IDK, é apenas apoiado a etapa de verificação. Ainda segundo os autores, qualquer módulo pode fornecer capacidades de verificação se um desenvolvedor segue as instruções básicas para a construção do módulo.

3. Comparativos entre Metodologias AOSE x UML

O objetivo dessa comparação é verificar se os artefatos criados pelas metodologias não poderiam fazer uso dos diagramas presentes na UML. Para tal finalidade, realizou-se um mapeamento dos diagramas através de tabelas. [Guedes 2004], [Bezerra 2007] e [Sommerville 2007] não definem com exatidão quais diagramas devem ser usados em quais fases das metodologias. Todos os autores deixam implícito que a escolha e o uso dos diagramas varia conforme a necessidade do projeto. Todavia, as tabelas ilustradas nas próximas seções tiveram a sua coluna UML organizada de acordo com a sugestão de uso presente no livro [Guedes 2004]. A seguir, é explicado cada uma das comparações.

3.1. Comparação com o Prometheus

Tabela 1. Comparação entre os artefatos gerados pela Metodologia Prometheus X Linguagem UML

UML	Prometheus
Diag. de Casos de Uso/Detalhado	Diag de Cenários; Formulário de Descrição do agente; Diag. de Ligação dos papéis dos agentes; Formulário Descritor de Planos
Diag. de Classes	
Diag. de Objetos	
Diag. de Estrutura Composta	
Diag. de Sequência	Diag. de Interação
Diag. de Comunicação	Diag. de Familiaridade de Agentes
Diag. de Máquina de Estados	
Diag. de Atividades	
Diag. de Componentes	Diag. de Acoplamento de Dados
Diag. de Implantação	
Diag. de Pacotes	Diag. de Visão Geral de Objetivos
Diag. de Interação	
Diag. de Tempo	
Diags. extras das Metodologias	Diag. de Papéis do Sistema
	Diag. de Visão Geral do Sistema
	Diag. de Protocolos de Interação
	Diag. de Visão Geral do Agente
	Diag. de Capacidades Modelando Planos

A metodologia Prometheus é composta de 13 artefatos para auxiliar o projetista na sua utilização, confirme visto na tabela 1. Alguns desses artefatos, podem ser modelados utilizando os diagramas presentes na UML sem perder a sua expressividade. É o caso dos seguintes artefatos:

- **Diagrama de Cenários:** Este artefato tem como finalidade mostrar uma visão generalizada do sistema [Padgham and Thangarajah 2004]. Para [Guedes 2004], o diagrama de casos de uso da UML tem o poder de expressar uma ideia geral sobre como o sistema irá se comportar. Se os dois diagramas tem praticamente a mesma finalidade, é possível a sua utilização sem perda nenhuma de sentido.

- **Diagrama de Visão Geral de Objetivos:** De acordo com [Padgham and Thangarajah 2004], este artefato tem como propósito representar os objetivos do sistema. [Guedes 2004] explica que os diagramas de pacotes representam os subsistemas englobados por um sistema de forma a determinar as partes que o compõem. Se pensarmos que cada objetivo principal é um sistema, e seus subobjetivos são subsistemas, é possível fazer uso do diagrama de pacotes para modelar os objetivos do SMA na metodologia.
- **Diagrama de Familiaridade de Agentes:** Conforme [Guedes 2004], o diagrama de comunicação é um complemento para o diagrama de sequência. Ademais, o diagrama de comunicação também desempenha a modelagem do vínculo entre os objetos do sistema e suas trocas de mensagens durante o processo. Se pensarmos que cada agente é um objeto, esse diagrama pode ser utilizado para modelar a familiaridade de agentes. Os objetivos são semelhantes, visto que o diagrama de familiaridades de agentes, consiste em ligar um agente com demais agentes que ele possui alguma interação [Padgham and Winikoff 2002].
- **Diagrama de Acoplamento de Dados:** Para [Padgham and Winikoff 2002], o diagrama de acoplamento de dados tem como propósito identificar os tipos de dados que necessitam ser armazenados pelo sistema e as relações que os papéis identificados tem com esses dados. [Guedes 2004] explica que o diagrama de componentes serve para representar os componentes do sistema quando o mesmo for ser implementado em termos de módulos de código fonte, bibliotecas, formulários, arquivos de ajuda, etc. Se pensarmos que cada papel do sistema e cada tipo de dado é um componente do sistema, esse diagrama consegue replicar fielmente a representação do diagrama de acoplamento de dados.
- **Diagrama de ligação de papel agente:** Segundo [Padgham and Winikoff 2002], o diagrama de ligação de papel de agente auxilia no agrupamento dos papéis identificados anteriormente, além da ligação aos agentes que irão executá-los. Fazendo uma analogia com o diagrama de casos de uso presente na UML, os papéis podem ser representados por "funcionalidades", já que os agentes devem executá-los, e os atores podem representar os agentes, conseguindo expressar similarmente a transição de um diagrama para outro sem perder a sua essência.
- **Diagrama de Interação:** Conforme [Padgham and Winikoff 2002], este diagrama contribui para especificar completamente a interação com os agentes. [Guedes 2004] afirma que o diagrama de sequência presente na UML é utilizado para modelar a troca de mensagens entre os objetos de um sistema. Se pensarmos que os agentes podem ser equiparados com objetos para a utilização do diagrama de sequências, podemos utilizá-lo sem perder o significado do diagrama de interação.
- **Formulário Descritor de Agente e Formulário Descritor de Planos:** Ambos os formulários sugeridos pela metodologia Prometheus permitem a escrita de texto em linguagem natural. Essa característica também é encontrada no diagrama de casos de uso detalhado, que compõe a UML.

Os demais diagramas presentes na metodologia Prometheus não oportunizam ser expressados utilizando os diagramas da linguagem UML, pois apresentam características que vão além das encontradas nesses diagramas.

Tabela 2. Comparação entre os artefatos gerados pela Metodologia Tropos X Linguagem UML

UML	Tropos
Diag. de Casos de Uso/Detalhado	
Diag. de Classes	
Diag. de Objetos	
Diag. de Estrutura Composta	
Diag. de Sequência	Diag. de Interação
Diag. de Comunicação	
Diag. de Máquina de Estados	
Diag. de Atividades	Diag. de Capacidade
Diag. de Componentes	
Diag. de Implantação	
Diag. de Pacotes	
Diag. de Interação	
Diag. de Tempo	
Diags. extras das Metodologias	Diag. de Autor
	Diag. de Objetivos
	Diag. de Objetivos Modelando Planos

3.2. Comparação com o Tropos

A metodologia Tropos é composta de 5 artefatos para auxiliar o projetista na sua utilização, confirme visto na tabela 2. Dois desses artefatos, podem ser transcritos utilizando diagramas presentes na ferramente UML. É o caso dos seguintes artefatos:

- **Diagrama de Capacidade:** Conforme [Bresciani et al. 2004], este diagrama é representado por meio de diagramas de atividades presente na linguagem UML. A metodologia utiliza esse diagrama por padrão, ou seja, ele satisfaz as necessidades fundamentais para modelar o conceito de capacidades que os agentes necessitam.
- **Diagrama de Interação:** Segundo [Bresciani et al. 2004], o diagrama de interação da metodologia Tropos é modelado utilizando o diagrama de interação presente na AUML. Entretanto, a diferenciação deste diagrama em relação ao de sequência da UML ocorre em virtude do primeiro modelar as interações dos atores com os agentes, ao invés de atores com objetos igual ao segundo caso. Portanto, se representar os objetos como sendo agentes, é possível transcrever as interações por meio do diagrama de sequências sem perda de informações.

Os outros três artefatos presentes na metodologia não podem ser modelados utilizando os artefatos da UML, pois englobam características que vão além do domínio suportado pelos diagramas da UML.

3.3. Comparação com o MaSE

A metodologia MaSE é composta de 8 artefatos para auxiliar o projetista na sua utilização, confirme visto na tabela 3. Segundo [Henderson-Sellers and Giorgini 2005], a metodologia MaSE foi construída sobre as técnicas orientadas a objeto existentes, porém, foi especializada para o domínio de SMA. Por essa razão, é a metodologia que mais apresenta

Tabela 3. Comparação entre os artefatos gerados pela Metodologia MaSE X Linguagem UML

UML	MaSE
Diag. de Casos de Uso/Detalhado	
Diag. de Classes	Diagrama de Classe de Agentes; Diagrama de Arquitetura de Agentes;
Diag. de Objetos	
Diag. de Estrutura Composta	
Diag. de Sequência	Diag. de Sequência
Diag. de Comunicação	
Diag. de Máquina de Estados	
Diag. de Atividades	Diag. de Tarefas Concorrentes; Diag. de Classe de Comunicação
Diag. de Componentes	
Diag. de Implantação	Diag. de Implantação
Diag. de Pacotes	Diag. de Hierarquia de Objetivos
Diag. de Interação	
Diag. de Tempo	
Diags. extras das Metodologias	Diag. de Modelo de Papéis

afinidades com os diagramas da linguagem UML. Dos 8 artefatos presentes na metodologia, 7 podem ser expressados utilizando a UML. Eles são:

- **Diagrama de Hierarquia de Objetivos:** Segundo [Henderson-Sellers and Giorgini 2005], este diagrama serve para estruturar os objetivos em ordem de importância. Fazendo a mesma analogia feita com o diagrama de Diagrama de Visão Geral de Objetivos da metodologia Prometheus, o diagrama de Hierarquia de Objetivos presente na metodologia MaSE também pode ser ilustrado utilizando o diagrama de pacotes presente na UML.
- **Diagrama de Sequência:** Conforme [Henderson-Sellers and Giorgini 2005], este diagrama é semelhante ao utilizado na UML. Sua diferenciação ocorre, devido ao da metodologia MaSE representar sequências de eventos entre os papéis, ao invés de objetos igual ao da UML. Entretanto, ambos tem a mesma notação gráfica e elementos, por isso podem ser equiparados.
- **Diagrama de Tarefas Concorrentes:** Para [Henderson-Sellers and Giorgini 2005], este diagrama serve para modelar as tarefas simultâneas realizadas pelos agentes. Um diagrama da UML que modelaria sem perda de expressividade essa situação, é o diagrama de atividades, visto que o mesmo também possui a notação para representar tarefas paralelas, além deste diagrama também possuir as condições de guarda.
- **Diagrama de Classe de Agente:** Este diagrama é semelhante ao de classes presente na UML. Sua diferenciação ocorre em virtude de um representar as classes de agentes e outro as classes de entidades do mundo real. É possível transcrever o diagrama de classes da agentes utilizando o de classes da UML sem nenhuma perda.
- **Diagrama de Classes de Comunicação:** De acordo com [Henderson-Sellers and Giorgini 2005], uma conversa define um protocolo

de coordenação entre dois agentes, e isto pode ser documentado utilizando o diagrama de classes de comunicação. Este diagrama tem uma notação semelhante ao de atividades da UML. Portanto, é possível transpassar os dados de um diagrama ao outro sem extravios de informação.

- **Diagrama de Arquitetura dos Agentes:** Este diagrama serve para especificar o conjunto de atributos, métodos, e sub-arquiteturas que possam conter os agentes [Henderson-Sellers and Giorgini 2005]. Sua notação gráfica é semelhante a presente no diagrama de Classes da UML. Por esta razão, é plausível reproduzir um diagrama no outro sem redução de expressividade.
- **Diagrama de Implantação:** Este diagrama serve para demonstrar números, tipos e locais das instâncias dos agentes no sistema [Henderson-Sellers and Giorgini 2005]. Sua notação gráfica é similar a encontrada no diagrama de implantação da UML. A diferença entre eles, ocorre devido ao diagrama de implantação da UML ter como objetivo a determinação das necessidades de hardware do sistema, as características físicas como servidores, estações, protocolos de comunicação, etc [Guedes 2004]. Devido a sua similaridade de notação gráfica, é cabível transcrever um diagrama no outro sem perda de sentido.

O número elevado de similaridades entre os diagramas da metodologia MaSE e os da UML decorre de MaSE ser baseado no paradigma de orientação a objetos. O único diagrama que não pode ser expressado pela UML é o diagrama de papéis, pois apresenta informações distintas não suportadas por diagramas UML.

3.4. Comparação com Ingenias

A metodologia Ingenias se difere das demais por ter sido construída utilizando pontos de vista distintos. Segundo [Henderson-Sellers and Giorgini 2005], a instanciação desses pontos de vista com entidades que representem problemas concretos, são o ponto base para o processo de desenvolvimento dessa metodologia. A fim de padronização, neste trabalho chamou-se os diagramas desta metodologia de diagramas de ponto de vista.

A tabela 4 exibe o comparativo entre os artefatos da metodologia Ingenias com os da linguagem UML. Conforme é ilustrado, a metodologia Ingenias praticamente não possui nenhuma similaridade em relação a UML, somente em um caso:

- **Diagrama do Ponto de Vista de Interação:** [Henderson-Sellers and Giorgini 2005] definem as notações utilizadas para modelar as interações entre os agentes e entre os agentes e os seres humanos. Eles explicam que essas interações também podem ser modeladas utilizando o diagrama de comunicação presente na UML, entretanto, este diagrama contém algumas restrições que fazem o modelo perder um pouco de sua expressividade conforme aumenta a complexidade das interações. Para tanto, os autores da metodologia propuseram uma versão de diagramas de Comunicação da UML, chamada de Diagrama de Interação GRASIA.

Nos estudos realizados acerca dessa metodologia, notou-se que os autores propuseram em sua ferramenta intitulada IDK um conjunto de diagramas para apoiar a modelagem do SMA utilizando da sua metodologia. Nas bases científicas abertas, não encontrou-se nenhum trabalho que explicasse integralmente as possibilidades que a ferramenta IDK

Tabela 4. Comparação entre os artefatos gerados pela Metodologia Ingenias X Linguagem UML

UML	Ingenias
Diag. de Casos de Uso/Detalhado	
Diag. de Classes	
Diag. de Objetos	
Diag. de Estrutura Composta	
Diag. de Sequência	
Diag. de Comunicação	Diag. do ponto de vista de Interação
Diag. de Máquina de Estados	
Diag. de Atividades	
Diag. de Componentes	
Diag. de Implantação	
Diag. de Pacotes	
Diag. de Interação	
Diag. de Tempo	
Diags. extras das Metodologias	Diag. do ponto de vista da Organização
	Diag. para especificar Fluxo de Trabalho
	Diag. do ponto de vista do Agente
	Diag. do ponto de vista de tarefas/objetivos
	Diag. do ponto de vista do Ambiente

podem proporcionar a seu usuário. Portanto, acredita-se, com base nas definições dessa metodologia, que os diagramas presentes no IDK, nos quais os autores intitularam GRASIA, servem para modelar o SMA utilizando das notações presentes nos diagramas de ponto de vista.

4. Conclusões e trabalhos futuros

Este trabalho teve como objetivo apresentar uma revisão de algumas metodologias para modelagem de sistemas multiagente. Também foi realizado um estudo comparativo entre as metodologias estudadas (Prometheus, Tropos, MaSE e Ingenias) e a linguagem UML, de forma a apresentar similaridades e diferenças.

Com base na exploração das metodologias, percebe-se que cada uma possui enfoque em características distintas. Enquanto Prometheus e Tropos focam sua modelagem no âmbito de agentes, MaSE e Ingenias possuem conceitos que propiciam também a modelagem de recursos e arquiteturas alternativas de agentes. Percebe-se também que a linguagem UML não é capaz de suprir os conceitos envolvidos no paradigma multiagente e que as metodologias AOSE ainda apresentam disparidade de objetivos, enfraquecendo sua normalização.

Em suma, a área de AOSE deve progredir para atingir uma padronização. Sugere-se que seja feito um mapeamento dos requisitos necessários para desenvolver qualquer tipo de SMA. Com isso, as entidades responsáveis por tal finalidade, deverão chegar há um consenso em relação a melhor metodologia para modelar SMA atualmente. Posterior a isso, deve-se trabalhar na metodologia selecionada para que esta atenda todos os conceitos envolvidos nas demais metodologias.

5. Agradecimentos

Os autores agradecem à Universidade Federal do Rio Grande - FURG e a Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul - FAPERGS pelo suporte financeiro na realização do presente trabalho.

Referências

Bergenti, F., Gleizes, M.-P., and Zambonelli, F. (2004). *Methodologies and software engineering for agent systems: the agent-oriented software engineering handbook*, volume 11. Springer.

Bezerra, E. (2007). *Princípios de Análise e Projeto de Sistemas com UML*. Elsevier.

Brandão, A. A. F. (2014). Apresentação de oficina no wesaac 2014 - engenharia de software orientada a agente. Enviado por e-mail pela autora (anarosabrandao@gmail.com), em 17 julho 2014.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

DeLoach, S. A., Wood, M. F., and Sparkman, C. H. (2001). Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(03):231–258.

Guedes, G. (2004). Uml: uma abordagem prática. *Novatec Editora*.

Guedes, G. T. A. (2012). *Um metamodelo UML para a modelagem de requisitos em projetos de sistemas multiagentes*. PhD thesis, Universidade Federal do Rio Grande do Sul.

Henderson-Sellers, B. and Giorgini, P. (2005). *Agent-oriented methodologies*. IGI Global.

Jayatilleke, G. B., Padgham, L., and Winikoff, M. (2007). Evaluating a model driven development toolkit for domain experts to modify agent based systems. In *Agent-Oriented Software Engineering VII*, pages 190–207. Springer.

Khallouf, J. and Winikoff, M. (2009). The goal-oriented design of agent systems: a refinement of prometheus and its evaluation. *International Journal of Agent-Oriented Software Engineering*, 3(1):88–112.

Padgham, L. and Thangarajah, J. (2004). Tutorial prometheus. http://www.cs.rmit.edu.au/agents/pdt/docs/Tutorial.pdf.

Padgham, L. and Winikoff, M. (2002). Prometheus: A methodology for developing intelligent agents. *John Wiley & Sons*.

Padgham, L. and Winikoff, M. (2005). *Developing intelligent agent systems: A practical guide*, volume 13. John Wiley & Sons.

Rodriguez, L., Insfran, E., and Cernuzzi, L. (2011). *Requirements Modeling for Multi-Agent Systems*. INTECH Open Access Publisher.

Sommerville, I. (2007). *Engenharia de Software*, volume 8. Person Education.

Modelagem de Agentes BDI-Fuzzy Submetidos ao Processo de Reputação

Henrique D. N. Rodrigues[1], **Diana F. Adamatti**[1], **Graçaliz P. Dimuro**[1]

[1]Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)
Rio Grande – RS – Brasil

{henriquedonancio,dianaada,gracaliz}@gmail.com

Abstract. *This paper presents a reputation model applied to a multi-agent system for simulating regulatory policies and reputation of the social organization of Sán Jerónimo Vegetable Garden, located in Seville, Spain. Therefore, was used BDI agents with fuzzy beliefs to investment analysis and satisfaction of services, as well as a reputation model as a performance measure of their activities within the project.*

Resumo. *Este artigo apresenta um modelo de reputação aplicado a um sistema multiagente para simulação de políticas normativas e reputação da organização social da Horta Urbana Sán Jerónimo, localizada em Sevilha, Espanha. Para tanto, foi utilizado agentes BDI com crenças fuzzy para análise de investimento e satisfação sobre serviços, assim como um modelo de reputação como medida de desempenho de suas atividades dentro do projeto.*

1. Introdução

Este trabalho é parte de um projeto que tem como objetivo o desenvolvimento de ferramentas SMA para simulação de processos de produção e gestão social de ecossistemas urbanos, em particular, o projeto social da Horta Urbana "Sán Jerónimo", localizada em Sevilha, Espanha, coordenado pela ONG "Ecologistas en Acción".

Os agentes participantes desse projeto são os hortelãos e aspirantes a hortelãos, que são os agentes sociais. Técnicos e secretaria assumem o papel de agentes governamentais, responsáveis pela verificação das normas aplicadas aos agentes sociais.

Os hortelãos, uma vez incluídos no projeto, têm direito a utilização da parcela (área designada ao cultivo e manejo da horta) por um prazo de dois anos prorrogáveis, desde que cumpram as normas e regras estabelecidas no regulamento definido pela ONG.

O regulamento da horta é um conjunto que conta um total de quarenta normas que visa estabelecer melhor convívio entre os usuários da horta e a administração, além de resguardar seus direitos e orientar suas ações.

O modelo de reputação adotado é baseado em modelos tais como REGRET [Sabater and Sierra 2001] e [Hübner et al. 2009]. A análise da reputação é divida em três dimensões: Dimensão Social, Dimensão Individual, Dimensão Ontológica, como proposto no modelo REGRET. Na Dimensão Social é analisada a efetividade do agente para com o seu grupo social, já na Dimensão Individual é analisada as trocas diretas entre os agentes. Por fim, têm se a Dimensão Ontológica, onde Dimensão Social e Individual se combinam para uma análise final.

Este artigo apresenta um modelo de reputação aplicado a um SMA para simular trocas que não envolvem bens materiais, como também a política interna da organização, tendo como métrica de desempenho social a reputação do agente dentro do contexto do projeto Horta Sán Jerónimo. Entende-se como trocas que não envolvem bens materiais, aquelas que tanto na contratação e/ou realização do serviço não há bens envolvidos, gerando apenas débitos e créditos virtuais.

O modelo de agentes com representação de crenças *fuzzy* (BDI-Fuzzy) foi escolhido devido a sua maior complexidade e abrangência em análises subjetivas e individuais de serviços, através da *Atitude de Avaliação de Serviços*.

Para implementação prévia do modelo foi utilizado a plataforma Jason[Bordini et al. 2007], um interpretador da linguagem AgentSpeak(L), baseada na arquitetura BDI [Rao 1996, Rao and Georgeff 1992], juntamente com o arcabouço CArtAgO [Ricci et al. 2014] e o arcabouço para prover a simulação de políticas públicas MSPP (Modeling and Simulation of Public Policies) [Santos and Costa 2012, Santos et al. 2012b].

O artigo está organizado como descrito a seguir. A Seção 2 apresenta a plataforma Jason, o arcabouço CArtAgO, e o modelo organizacional . A Seção 3 aborda características do framework MSPP adaptado para inserção das políticas normativas da organização.A Seção 4 expõe as características do modelo de reputação adotado juntamente com caracteríticas fuzzy de avaliação de trocas entre os agentes sociais. A Seção 5 apresenta a prévia simulação do SMA baseado na Horta San Jéronimo, assim como resultados preliminares.

2. A Plataforma JaCaMo

A plataforma JaCaMo [Bordini and Hübner 2014] é um arcabouço para programação de Sistemas Multiagentes constituído de três ferramentas.

O Jason [Bordini et al. 2007] é um interpretador da linguagem AgentSpeak(L), baseada na arquitetura BDI. Um aspecto importante dessa plataforma é sua implementação em Java, e portanto multi-plataforma. A comunicação entre agentes no Jason é baseada na teoria de atos de fala. Os agentes ao se comunicarem, geram crenças e estas por sua vez podem desencadear planos.

O arcabouço CArtAgO (Common ARTifact infrastructure for AGents Open environments) [Ricci et al. 2014] é baseado no modelo Agentes e Artefatos (A & A) para modelar e projetar Sistemas Multiagente. Com essa ferramenta é possível criar artefatos estruturados em espaços abertos onde agentes podem se unir de forma a trabalhar em conjunto.

Os Artefatos são recursos e ferramentas construídas de forma dinâmica, usados e manipulados por agentes para apoiar/realizar suas atividades individuais e coletivas. O ambiente como também os recursos disponíveis no mesmo podem ser modelados na forma de um artefato CArtAgO.

O modelo organizacional MOISE+ [Hübner 2003] é uma ferramenta com intuito de modelar a organização de SMA. Consiste na especificação de três dimensões: a estrutural, onde definem-se papéis e ligações de heranças e grupos; a funcional, onde é estabelecido um conjunto de planos globais e missões para que as metas sejam atingidas;

e a deôntica, que é a dimensão responsável pela definição de qual papel tem obrigação ou permissão para realizar cada missão.

3. O arcabouço MSPP (Modeling and Simulation of Public Policies) e sua aplicação na Horta Sán Jerónimo

O arcabouço MSPP [Santos and Costa 2012, Santos et al. 2012b] para inserção de políticas públicas concretiza-se no formato de artefatos no modelo CArtAgO. Estão incluídos neste arcabouço dois tipos de artefatos normativos: NormObrig e NormPrb, modelando normas de obrigação e proibição, respectivamente.

Além dos artefatos, estão previamente inseridos agentes para executar/verificar tais normas. São eles o agente governamental, responsável por emitir as normas, os agentes sociais que estão submetidos às normatizações e buscam atingir objetivos próprios, e também os agentes governamentais detectores/efetuadores responsáveis por detectar o cumprimento das normas da política como também características e recursos do ambiente, aplicar possíveis sanções a ações que caracterizarem o descumprimentos de normas e regularizar os recursos disponíveis no ambiente.

Em um estudo preliminar, verificou-se que o arcabouço MSPP poderia atender as necessidades normativas para o estudo de caso da Horta San Jerónimo (HSJ), salve algumas modificações discutidas em [Rodrigues et al. 2013]. Dentre as modificações feitas para adequar a estrutura das normas ao caso, está a inclusão de dois novos tipos de normas: normas de permissão e normas de direito, além da adaptação da composição de uma norma, uma vez que o arcabouço MSPP é maleável nesse sentido.

Normas de permissão foram inseridas pois algumas ações são possíveis de acordo com os interesses da horta. Por exemplo, o agente hortelão que deseje plantar uma árvore que tenha um ciclo maior que o de dois anos, precisa requerer uma autorização já que seu contrato de uso da parcela é de apenas dois anos. Normas de direito são aquelas que garantem a execução da ação, como o caso em que cada agente hortelão pode possuir um tonel para água.

4. Modelo de Reputação

O modelo de reputação segue a estrutura adotada em REGRET [Sabater and Sierra 2001], onde a reputação é uma composição de três dimensões: Dimensão Social, Dimensão Individual e Dimensão Ontológica.

4.1. Dimensão Social

A dimensão social avalia aspectos coletivos em relação ao agente. Em [Hübner et al. 2009] essa avaliação é feita analisando a efetividade do agente em relação a normas (obrigações), a participação do agente e seus resultados.

No projeto HSJ, os agentes devem desempenhar algumas obrigações, tais como pagamento de mensalidades e renovação de suas inscrições, além do cumprimento das normas. A participação também é um fator a ser levado em conta uma vez que o projeto determina que o agente participe de reuniões que servem para discutir e orientar suas práticas. Os resultados, são todas ações que o agente venha executar coletivamente.

Cada agente tem como crença um respectivo valor referente ao grau de importância[1] que ele atribui a estes aspectos que são compartilhados entre o grupo. A avaliação individual dada por cada individuo é dada pela Equação (1) [Hübner et al. 2009]:

$$avaliacao(\alpha) = \frac{\gamma p(\alpha) + \delta o(\alpha) + \epsilon r(\alpha)}{\gamma + \delta + \epsilon} \quad (1)$$

Onde os fatores γ, δ e ϵ definem a importância da participação, obediência e resultados, respectivamente.

Esses fatores são independentes entre os agentes sociais, podendo assumir valores por convenção entre 1 a 10, definindo assim o grau de relevância que o agente determina para o atributo multiplicado pelo fator. Dessa forma, o valor mínimo dado para reputação de determinado agente é 0 e o valor máximo é 1.

4.2. Dimensão Individual

A dimensão individual é resultado das interações diretas entre os agentes. Em [Sabater and Sierra 2001], essa dimensão é tratada como a mais confiável, pois expressa resultados de interações diretas com o agente alvo, ou seja, uma avaliação dada pelo resultado da interação entre os agentes envolvidos.

Na abordagem desse trabalho, os agentes trocam serviços não econômicos que não envolvem trocas monetárias ou de bens. Os agentes hortelãos no projeto HSJ, tem como base de avaliação o investimento em realizar um serviço e a satisfação ao receber um serviço. Ao avaliar um investimento, o agente leva em consideração três fatores [Farias et al. 2013]:

- A *dificuldade*, que representa o quão difícil é para o agente realizar o serviço solicitado. Assume valores no intervalo de 0 e 10, onde valores próximos de 10 representam um grau de dificuldade maior.

- O *custo*, que representa os gastos para realizar o serviço. Assume valores entre 1 e 100, onde valores próximos de 100 indicam gastos maiores na realização do serviço.

- O *tempo*, que representa o tempo gasto na execução do serviço. Assume valores entre 1 e 90, onde valores próximos de 90 indicam um tempo maior empregado na realização do serviço.

Para a análise da satisfação como resultado de um serviço recebido é levado em consideração:

- A *qualidade*, que representa o grau de excelência do serviço. Assume valores entre 0 e 10 onde quanto maior o valor, maior o grau de excelência do serviço.

- O *preço*[2], que representa o valor dos gastos com a contratação do serviço. Assume

[1] O valores são independentes para cada agente afim de descentralizar o modelo proposto em [Hübner et al. 2009], tendo o agente uma avaliação mais individual dos aspectos coletivos.

[2] Para o estudo de caso esse fator não é levado em consideração.

valores entre 0 e 100, onde valores próximos de 0 indicam gastos menores.

- O *tempo*, que representa o tempo gasto esperando a execução do serviço. Assume valores entre 1 e 90, onde quanto menor o valor, menor o tempo esperado na realização do serviço.

Um agente ao requisitar que outro agente preste um serviço, calcula a satisfação esperada através de uma função de pertinência fuzzy e uma base de regras e com base na sua "Atitude de Avaliação de Serviço" [Farias et al. 2013].

Tabela 1. Base de Regras Fuzzy para Análise do Investimento Baseado na Dificuldade

Base de Regras Fuzzy do Investimento (Dificuldade)
$R^{(1)}$: IF dificuldade IS baixa THEN investimento IS baixo
$R^{(2)}$: IF dificuldade IS média THEN investimento IS médio
$R^{(3)}$: IF dificuldade IS alta THEN investimento IS alto

A Atitude de Avaliação de Serviço é a composição de um ou mais atributos que pertencem a esse serviço. Isto significa que o agente pode analisar o investimento na realização do serviço "plantar", baseando-se exclusivamente na dificuldade de se executar essa ação, e em contrapartida analisar a satisfação em receber este serviço analisando por exemplo, o tempo e a dificuldade. Cada agente possui sua Atitude de Avaliação de Serviço independentemente.

Ao receber o serviço o agente calcula a satisfação real, assim como o agente que fez o investimento calcula o investimento real. A relação entre satisfação esperada e satisfação real, gera crédito para quem prestou o serviço e débito para quem recebeu o serviço.

Tabela 2. Base de Regras Fuzzy para Análise do Débito

Base de Regras Fuzzy (Satisfação Esperada X Satisfação Real)
$R^{(1)}$: IF satisfação esp. IS baixa AND satisfação real IS baixa THEN débito IS médio
$R^{(2)}$: IF satisfação esp. IS baixa AND satisfação real IS média THEN débito IS alto
$R^{(3)}$: IF satisfação esp. IS baixa AND satisfação real IS alta THEN débito IS alto
$R^{(4)}$: IF satisfação esp. IS média AND satisfação real IS baixa THEN débito IS baixo
$R^{(5)}$: IF satisfação esp. IS média AND satisfação real IS média THEN débito IS médio
$R^{(6)}$: IF satisfação esp. IS média AND satisfação real IS alta THEN débito IS alto
$R^{(7)}$: IF satisfação esp. IS alta AND satisfação real IS baixa THEN débito IS baixo
$R^{(8)}$: IF satisfação esp. IS alta AND satisfação real IS média THEN débito IS baixo
$R^{(9)}$: IF satisfação esp. IS alta AND satisfação real IS alta THEN débito IS médio

4.3. Balanço Material e Virtual

O balanço material de um agente (α), no modelo BDI-Fuzzy, é obtido através de uma avaliação fuzzy (qualitativa) obtido pelos valores materiais de investimento R_a e

satisfação S_a, gerados na troca de serviços com um outro agente, segundo uma avaliação pessoal do agente [Farias et al. 2013].

A escala com os termos linguísticos para representação do balanço material e virtual são:

T_{bm} = $<$ *muito desfavorável, desfavorável, equilibrado, favorável, muito favorável* $>$

Tendo o seu valor normalizado representado pela função de pertinência fuzzy [Figura 1] :

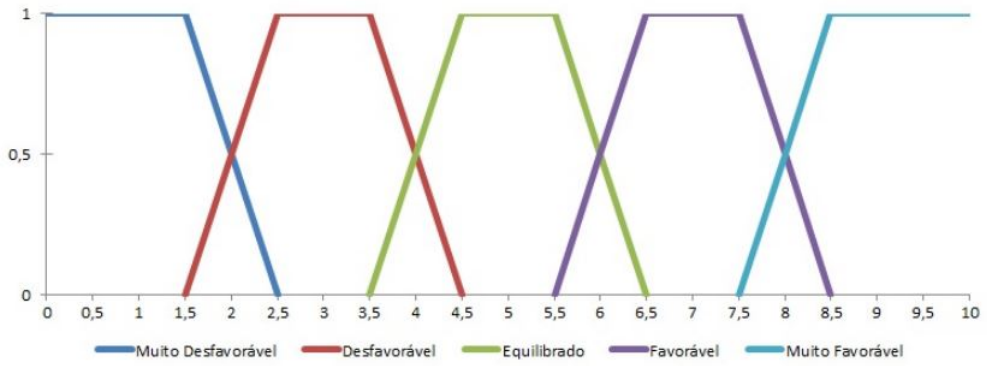


Figura 1. Escala para variáveis linguísticas [Farias et al. 2013]

A cada troca realizada onde o agente presta um serviço (faz um investimento), e recebe um serviço (gera uma satisfação) com um agente alvo, é gerado um balanço material. A cada troca o agente registra o balanço material em um vetor *v* . Esse acumulado de trocas gera a reputação com o agente alvo, de acordo com a equação (2):

$$\sum_{i=1}^{\text{size}(v)} \frac{v_i * a_n}{\text{size}(v)}, \tag{2}$$

onde a_n = a_1 + (n - 1) * β, sendo β dado por $\beta = \frac{a_1+1}{\text{size}(v)+1}$ e a_1 = 0.1

Uma vez que o agente presta um serviço, ele gera crédito em relação a quem recebe o serviço, e o agente que recebe o serviço contrai um débito para quem prestou o serviço. Uma vez realizada a troca, ou seja, o agente realiza um investimento ao prestar um serviço e gera uma satisfação ao receber um serviço, os valores de crédito e débito gerados são analisados através do "Balanço Virtual".

O Balanço Virtual é atualizado a cada novo crédito e débito gerados na prestação ou recebimento de um serviço através da base de regras [Farias et al. 2013]:

Tabela 3. Avaliação Fuzzy do Balanço Virtual Quando o Agente Realiza um Serviço (Balanço Virtual x Crédito)

	baixo	médio	alto
muito desfavorável	muito desfavorável	desfavorável	equilibrado
desfavorável	desfavorável	equilibrado	favorável
equilibrado	equilibrado	favorável	muito favorável
favorável	favorável	muito favorável	muito favorável
muito favorável	muito favorável	muito favorável	muito favorável

Tabela 4. Avaliação Fuzzy do Balanço Virtual Quando o Agente Recebe um Serviço (Balanço Virtual x Débito)

	baixo	médio	alto
muito desfavorável	muito desfavorável	muito desfavorável	muito desfavorável
desfavorável	desfavorável	muito desfavorável	muito desfavorável
equilibrado	equilibrado	desfavorável	muito desfavorável
favorável	favorável	equilibrado	desfavorável
muito favorável	muito favorável	favorável	equilibrado

O agente ao ser solicitado a executar uma das três tarefas que compõe o cenário deste estudo: plantar, irrigar e colher, sempre requisita auxílio a outro agente. O critério de escolha acerca de quem poderá ser solicitado é baseado no Balanço Virtual e segue os seguintes passos:

1. Balanço Virtual *muito desfavorável*: Esse tipo de balanço é preferível uma vez que o agente possui muito mais créditos do que débitos em relação ao agente alvo.

2. Balanço Virtual *desfavorável*: Uma relação com o agente alvo com balanço virtual desfavorável é a segunda opção quando não há uma relação com balanço virtual muito desfavorável, pois o agente possui uma quantidade de créditos relativamente maior que débitos.

3. Balanço Virtual *equilibrado*: O agente possui uma relação equilibrada de trocas com o agente alvo. Esse balanço virtual ainda é preferível a pedir um desconhecido uma vez que o agente terá que pedir auxílio e consequentemente gerar um débito, porém trocas anteriores atingiram um equilíbro.

4. Balanço Virtal *favorável/muito favorável*: Quando o agente não possui sequer um balanço equilibrado com outro agente, ele então requisita a quem ele contraiu mais débitos para que indique outro agente. O agente não requisita o agente com que possui mais débitos pois seu balanço já é negativo e ele busca atingir um equilibrio.

- *Indicação*: Os agentes ao serem solicitados para que indiquem alguém para fazer um serviço, buscam indicar aqueles com que: tiveram uma satisfação alta em receber o serviço, como segunda opção aqueles com que teve uma satisfação media, e como última escolha, não havendo nenhuma das demais opções, aqueles com que obteve satisfação baixa.

4.4. Dimensão Ontológica

A Dimensão Ontológica é a combinação das dimensões Social e Individual. Como já dito, a Dimensão Individual é mais confiável em relação a Dimensão Social, uma vez que a primeira expressa as relações diretas entre os agentes. A equação que produz essa combinação é expressa em (3):

$$D_o(\alpha) = \frac{\vartheta D_s(\alpha) + \varphi D_i(\alpha)}{\vartheta + \varphi} \tag{3}$$

Onde os fatores ϑ e φ definem a importância da Dimensão Social e Dimensão Individual respectivamente, sendo $\vartheta < \varphi$ e aplicados a todos os agentes. O valor desses fatores deve ser escolhido de acordo a necessidade de se valorizar a Dimensão Individual em relação a Dimensão Social. Ressalta-se que D_i e D_s devem estar normalizados.

5. Simulação e Resultados Preliminares

Cada agente, possui valores de investimento esperado que em um primeiro momento é informado ao agente que receberá o serviço, gerando uma satisfação esperada. Em uma segunda etapa, ao concluir o serviço o agente gera uma satisfação real, que corresponde aos valores reais investidos na realização do serviço avaliado tanto no primeiro momento quanto no segundo de acordo com a Atitude de Avaliação de Serviço de cada parte envolvida.

Tabela 5. Investimento Esperado/Real do Agente Cícero

	Dificuldade	**Custo**	**Tempo**
Serviço Plantar	7.3	60	80
Serviço Colher	5	40	50

Tabela 6. Investimento Esperado do Agente Genaro

	Dificuldade	**Custo**	**Tempo**
Serviço Plantar	8	70	62
Serviço Colher	7.5	65	51

Tabela 7. Investimento Real do Agente Genaro

	Dificuldade	**Custo**	**Tempo**
Serviço Plantar	9	92	45
Serviço Colher	9.2	80	40

No cenário criado existem dois agentes, Cícero e Genaro, que trocam serviços entre si. As trocas nesse caso específico, acontecem até que o Balanço Virtual atinja estados através da Heurística de Busca de Parceiros a condição de *favorável* ou *muito favorável*, quando então o agente pede indicação para o agente alvo sobre quem possa executar o serviço. Assim, agentes que investem menos, acabam realizando mais serviços para alcançar uma condição de balanço favorável ou muito favorável, além do fato que a cada serviço realizado, sua reputação experimenta variação de valores inferiores aqueles que por exemplo, superam as expectativas, como é o caso do agente Genaro, como mostra a tabela [8].

Tabela 8. Variação da Dimensão Individual Baseada na Atitude de Avaliação de Serviço

Avaliação de Cícero para Genaro	Avaliação de Genaro para Cícero
0.000	0.000
0.400	0.468
1.075	1.281
2.025	–

As simulações iniciais indicam que o modelo híbrido de reputação adotado contempla tanto o desempenho coletivo, tanto o individual dos agentes envolvidos de forma satisfatória, favorecendo aqueles que melhor desempenham seus papéis tanto em relações diretas de trocas como também no coletivo em relação a política normativa (ver [Rodrigues et al. 2013].

Em trabalhos futuros espera-se que mais agentes possam interagir, trocando informações e indicando agentes para serviços, como proposto na heurística de busca por parceiros, assim como integrar aspectos sociais coletivos afim de obter a Dimensão Ontológica aqui apresentada.

Agradecimentos

Este trabalho é apoiado pelo CNPq (Proc. 560118/10-4, 305131/2010-9, 476234/2011-5) , FAPERGS (Proc. 11/0872-3) e Projeto RS-SOC (FAPERGS Proc. 10/0049-7).

Referências

Bordini, R. H. and Hübner, J. F. (2014). Jacamo project. http://jacamo.sourceforge.net/.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, New Jersey.

da Silva, V. T., Hermoso, R., and Centeno, R. (2009). *A Hybrid Reputation Model Based on the Use of Organizations*. Springer.

Farias, G. P., Dimuro, G., Dimuro, G., and Jerez, E. D. M. (2013). Exchanges of services based on Piaget's theory of social exchanges using a BDI-fuzzy agent model.

Hill, M. (2004). *The Public Policy Process*. Pearson Longman, 4 edition.

Hübner, J. F. (2003). *Um Modelo de Reorganização de Sistemas Multiagentes*. PhD thesis, Universidade de São Paulo, São Paulo.

Hübner, J. F., Vercouter, L., and Boissier, O. (2009). *Instrumenting Multi-Agent Organisations with Artifacts to Support Reputation Processes*. Springer.

Huynh, T. D., Jennings, N. R., and Shadbolt, N. R. (2006). An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, pages 119–154.

Jurca, R. and Faltings, B. (2002). Towards incentive-compatible reputation management. In *Trust, Reputation, and Security: Theories and Practice*, pages 138–147.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, LNCS. Springer, Berlin.

Rao, A. S. and Georgeff, M. P. (1992). An abstract architecture for rational agents. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92), Cambridge, MA, October 25–29, 1992*, pages 439–449. Morgan Kaufmann.

Ricci, A., Santi, A., and Piunti, M. (2014). CArtAgO (common atifact infrastructure for agents open environments).

Rodrigues, H. D. N., Santos, F. C. P., Dimuro, G., Adamatti, D. F., Jerez, E. M., and Dimuro, G. P. (2013). A mas for the simulation of normative policies of the urban vegetable garden of san jerónimo, seville, spain.

Sabater, J. (2003). *Trust and Reputation for Agent Societies*. PhD thesis, Universidade Autónoma de Barcelona.

Sabater, J. and Sierra, C. (2001). Regret: A reputation model for gregarious societies. In *Proceedings of the Fourth Workshop on deception Fraud and Trust in Agent Societies*, pages 61–70.

Sabater, J. and Sierra, C. (2002). Reputation and social network analysis in multi-agent systems. In *Proceedings of the First Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Santos, F. C. P., Rodrigues, T. F., Dimuro, G., Adamatti, D. F., Dimuro, G. P., Costa, A. C. R., and Jerez, E. M. (2012a). Modelando organização social de um sma para simulação dos processos de produção e gestão social de um ecossistema urbano: o caso da horta san jerónimo da cidade de sevilla, espanha. pages 93–104. UFSC.

Santos, I. and Costa, A. C. R. (2012). Toward a framework for simulating agent-based models of public policy processes on the jason-cartago platform. In *Proceedings of the Second International Workshop on Agent-based Modeling for Policy Engineering in 20th European Conference on Artificial Intelligence (ECAI)- AMPLE 2012*, Montpellier. Montpellier University.

Santos, I., Mota, F. P., Costa, A. C. R., and Dimuro, G. P. (2012b). Um framework para simulação de políticas públicas aplicado ao caso da piracema, sob o olhar da teoria dos jogos. Porto Alegre. SBC.

Zacharia, G. M. P. (2000). Trust management through reputation mechanisms. *Applied Artificial Intelligence Journal*.

Modelando a Variação da Biomassa do Fitoplâncton no Estuário da Lagoa dos Patos através da Simulação Baseada em Multiagentes

Diego de Abreu Porcellis[1], Diana Adamatti[1], Paulo Abreu[2].

[1]Programa de Pós-Graduação em Modelagem Computacional – Universidade Federal do Rio Grande (PPGMC/FURG)
[2]Programa de Pós-Graduação em Oceanografia Biológica (PPGOB/FURG)

diegoporcellis@furg.br, dianaada@gmail.com, docpca@furg.br

Abstract. *This paper presents a study on the phytoplankton biomass variation in the estuary of the Patos Lagoon (ELP) and it aims to create a model that demonstrates the phytoplancton variability in the ELP and highlights the importance of water retention time in the phytoplankton biomass accumulation. To create this model, we have applied the Multi-Agent-Based Simulation using the NetLogo tool.*

Resumo. *Este artigo apresenta um estudo sobre variação de Biomassa de Fitoplâncton no Estuário da Lagoa dos Patos (ELP) e tem como objetivo criar um modelo que demonstre a variação de fitoplâncton no ELP e evidencie a importância do tempo de retenção da água no acúmulo de biomassa de fitoplâncton. Para a criação desse modelo, foi utilizada Simulação Baseada em Agentes e a implementação foi realizada na ferramenta NetLogo.*

1. Introdução

Estuários são ambientes de transição entre águas vindas do continente, água doce, e águas oriundas do mar, neste caso a água salgada [Pritchar, 1967]. O grande aporte de nutrientes e condições estressantes devido à variabilidade da salinidade faz com que poucas espécies consigam colonizar estes ecossistemas. Entretanto, as espécies que sobrevivem nos estuários encontram condições ótimas para se reproduzir o que leva a elevados índices de produção e acúmulo de biomassa, resultando em grandes pescarias A Lagoa dos Patos, no extremo Sul do Brasil, tem ao sul a sua ligação com o mar através de um canal de aproximadamente 800m de largura. A região do estuário tem uma área total de 900 km , e é um ambiente raso, onde 75% desta área tem 2 m, ou menos, de profundidade [Seeliger e Odebrecht 1997].

Devido a sua ligação estreita com o mar e a proximidade de um ponto anfidrômico, as marés nesta região são de pequena amplitude (em média 40 cm), e seu efeito é mais restrito a boca do estuário [Abreu et al 2010]. Devido a estas características a hidrologia do ELP é basicamente controlada pela direção e intensidade dos ventos e também pelos níveis de chuva e evaporação.

Devido à grande diversidade de dados e fenômenos envolvidos nesse ambiente, torna-se difícil e custoso obter dados conclusivos que permitam determinar os principais fatores controladores da variação de biomassa do fitoplâncton. A coleta de água de forma intensiva no tempo e no espaço, além das análises laboratoriais dos diversos parâmetros tornam o monitoramento do fitoplâncton neste ambiente. Por estes motivos objetivou-se neste estudo conhecer os principais fatores controladores da variação de biomassa de fitoplâncton neste ambiente, através da modelagem computacional, utilizando a técnica de Simulação Baseada em Multiagente.

A Simulação Baseada em Multiagentes tem sido muito utilizada atualmente devido ao aumento do poder computacional, a simulação baseada nesta técnica caracteriza-se pela interação de agentes em um ambiente. Através dela, existe a possibilidade de observar agentes individuais interagindo entre si e com o ambiente e assim ter um melhor entendimento do sistema como um todo.

Este artigo divide-se em seção 2, onde são discutidos os conceitos de fitoplâncton e produtor primário; na seção 3 são apresentada os materiais e os métodos que serão utilizados na criação do modelo; a seção 4 é feita uma discussão sobre o modelo proposto e alguns resultados preliminares e por fim na seção 5 são feitas as considerações finais.

2. Fitoplâncton / Produtor Primário

O plâncton que tem sua nomenclatura derivada da palavra grega *plágchton*, que significa errante. Esta é uma designação aplicada a todos os organismos marinhos que tem locomoção limitada com relação a corrente e acaba por ficar a deriva no ambiente. O Plâncton pode ser dividido em zooplâncton, que se refere aos pequenos animais e fitoplâncton, que designa todos os organismos planctônicos clorofilados [Lalli e Timothy 1997].

O fitoplâncton é o principal produtor primário do ambiente marinho. Os produtores primários são responsáveis por transformar substâncias inorgânicas em matéria orgânica através da fotossíntese. Esse processo é de vital importância para a cadeia alimentar dos ambientes aquáticos, pois o fitoplâncton fornece a matéria e energia que sustenta o crescimento de peixes, crustáceos e moluscos nos ecossistemas marinhos [Lalli e Timothy 1997].

O crescimento rápido de uma determinada espécie de fitoplâncton é chamado de *bloom* de fitoplâncton. Estudos em estuários, como [Howarth et al. 2000; Jassby 2008], tem o foco de conhecer os fatores que controlam os acúmulos de biomassa de fitoplâncton. Estes trabalhos mostraram a importância da luz, nutrientes e predação para a produção do fitoplâncton. Além desses, o tempo de transporte, conhecido também como tempo de retenção ou de circulação, é, em alguns estudos, considerado o principal fator de influência na variabilidade de biomassa de fitoplâncton, considerando os estudos de longo e curto prazo [Abreu et al 2010; Howarth et al. 2000].

No entanto, Lucas et al (2009) menciona que o tempo de transporte tem pouca influência na variabilidade de fitoplâncton, e que o que realmente afeta essa variabilidade são as taxas de duplicação e de perda de fitoplâncton. Já Abreu et al (2010), em um estudo de longo e curto prazo desenvolvido no estuário da Lagoa dos

Patos evidencia a importância do tempo de retenção na variabilidade de fitoplâncton neste ambiente.

Desta forma, para investigar e esclarecer a influência do tempo de transporte na variabilidade de fitoplâncton no ELP é necessário um estudo com grande quantidade de dados e amostras. Porém, o custo, tanto em termos de tempo quanto em termos financeiros, acaba se tornando alto. Devido a isso, foi proposto que seja criado um modelo computacional que simule esse ambiente e que possa realizar previsões precisas quanto ao acumulo de biomassa de fitoplâncton de modo que seja possível alterar condições iniciais e obter resultados precisos e rápidos.

3. Materiais e Métodos

A técnica computacional utilizada para realizar a modelagem computacional deste ambiente será os Sistemas Multiagentes (SMA), que é uma linha de pesquisa da Inteligência Artificial Distribuída (IAD) e é proposta com a finalidade de estudar o comportamento de agentes autônomos e sua evolução em um ambiente [Werlang, 2013]. Esta metodologia foi escolhida, pois, através dela, pode-se observar um ambiente natural que tenha comportamento inteligente, como o ambiente do estuário, e criar um modelo condizente com a sua estrutura [Johnson, 2001]. Os ganhos obtidos com essa técnica são:

- Facilidade na criação do modelo;
- Facilidade na modelagem de fenômenos emergentes;
- Rapidez na obtenção dos dados em relação aos estudos já realizados.

O SMA tem como objetivo estudar o comportamento de agentes autônomos que trabalham em conjunto cooperando ou competindo em um determinado ambiente [Werlang, 2013]. Por autônomos entende-se agentes independentes da existência de outros agentes, mas que podem sofrer influências em seu comportamento ou estado devido a outros agentes [Hubner, 2003].

O modelo criado será utilizado para previsão de cenários futuros e estudo destes cenários, além disso, podem ser inseridos cenários diversos para buscar um cenário otimizado para o sistema. É importante salientar, que já existe um estudo com esta finalidade sendo desempenhado no Instituto de Oceanografia da FURG, neste estudo são utilizadas técnicas da estatística para obtenção dos resultados do acumulo de biomassa de fitoplâncton, esse estudo servirá como base para esse trabalho e também, para a validação dos resultados do modelo proposto.

3.1 NetLogo

A ferramenta que será utilizada na criação do modelo de crescimento das algas é o NetLogo (figura 1). Está ferramenta possui uma linguagem de programação e um ambiente de desenvolvimento multiagente. O NetLogo é indicado para simulações de fenômenos naturais e sociais e suas principais vantagens são códigos simples e legíveis e uma interface intuitiva[Wilensky, 2013].

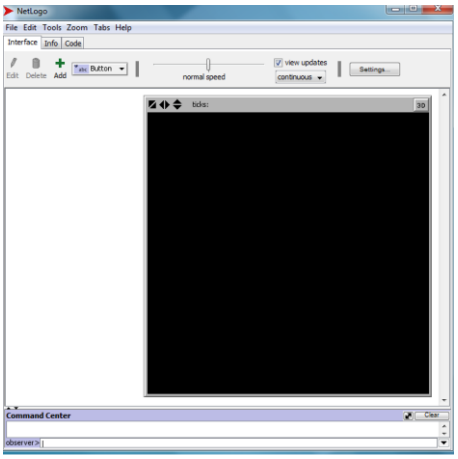


Figura 1. Interface do NetLogo

Existem três abas principais no NetLogo (Figura 1): *Interface*, *Info* e *Code*. A aba responsável pela criação da parte gráfica é a *Interface*. Nessa aba pode-se criar todo o ambiente de simulação. Neste ambiente podem ser inseridos os itens: Botões; Sliders (para definir valores de parâmetros); Monitores (para monitorar o conteúdo de determinados parâmetros); Plots (gráficos); entre outros.

A aba *Info* é utilizada para documentar o modelo criado, com informações sobre funções, parâmetros, modo de utilização e outras informações pertinentes ao modelo criado.

E por fim existe a aba *Code* (Figura 2) que é o local destinado a criação do código de funcionamento do modelo. O código criado determina o comportamento de cada agente e como ele se comunica com o ambiente e com outros agentes.

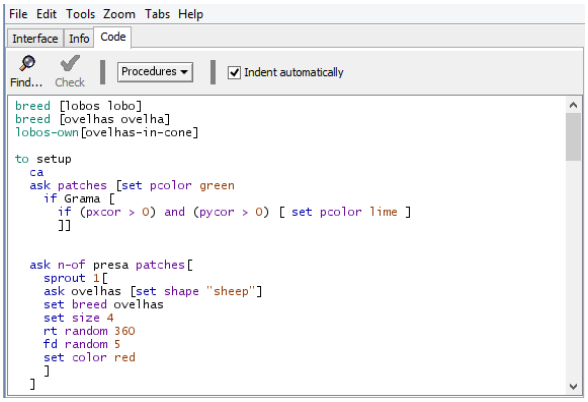


Figura 2. Exemplos de codificação no NetLogo.

4. Modelo Proposto

Em modelos preliminares já realizados foi possível verificar a importância do tempo de retenção no acumulo de biomassa de fitoplâncton. Num primeiro modelo (Figura 3) é possível escolher o número de algas iniciais, o tempo de duplicação dos fitoplânctons

(tempo em dias que demora para uma alga transformar-se em 2) e o tempo de retenção (tempo em dias que a abertura do estuário fica bloqueada devido à ação do vento) destes no estuário. Neste primeiro modelo não foram consideradas as perdas naturais, pois estas perdas são irrelevantes neste estuário [Abreu, Carstensen e Odebrecht no prelo]. O funcionamento desse modelo simula a ação do vento sobre o estuário gerando um tempo de retenção, assim que o vento para de agir as algas são escoadas para o mar.

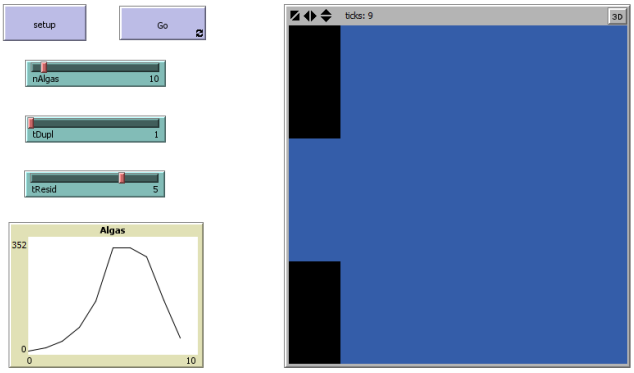


Figura 3. Modelo com tempo de duplicação e tempo de retenção.

Na sequencia, foi criado um modelo (Figura 4) que pudesse ser mais geral e pronto para ser utilizado em outros ambientes. Neste novo modelo foram utilizadas como dados de entrada o número inicial de algas, a taxa de duplicação (velocidade de duplicação das algas), a taxa e perda (aqui é considerado as perdas naturais), a taxa de retirada (velocidade de saída de algas do estuário) e o número de dias de simulação. Os resultados deste segundo modelo ainda estão em análise.

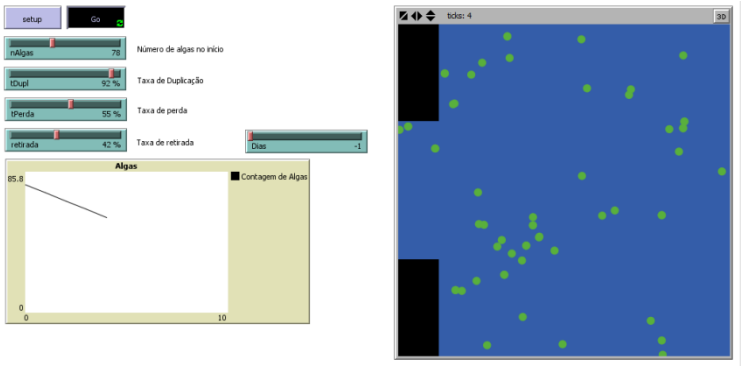


Figura 4. Modelo com tempo de duplicação e tempo de retenção.

5. Considerações finais

Espera-se através da realização deste trabalho, obter um modelo consistente com resultados encontrados em pesquisas *in loco*. O modelo será validado através dos dados obtidos durante os últimos anos no ELP. O modelo gerado também deve ser abstrato ao

ponto de em poucas alterações realizarem simulações de ambientes diversos a ELP, sendo assim adaptável a qualquer ambiente aquático.

6. Agradecimentos

Agradeço a CAPES pela bolsa de mestrado CAPES-DS, que me possibilitou dedicação exclusiva a meu estudo.

Referências

Lalli, Carol, e Timothy R. Parsons. (1997). Biological Oceanography: An Introduction: An Introduction. Butterworth-Heinemann.

Anacleto, Elisângela Inácia; Gomes, Eliana Traversim. (2009). Relações tróficas no plâncton em um ambiente estuarino tropical: Lagoa dos Patos (RS), Brasil. Saúde & Ambiente em revista, v. 1, n. 2.

Lucas, Lisa V., Janet K. Thompson, e Larry R. Brown. (2009). "Why are diverse relationships observed between phytoplankton biomass and transport time."Limnology and Oceanography 54.1 381-390.

Abreu, Paulo C., et al. (2010) "Short-and long-term chlorophyll a variability in the shallow microtidal Patos Lagoon estuary, southern Brazil." Estuaries and Coasts 33.2 554-569.

Abreu, Paulo C., Carstensen, Jacob., Odebrecht, Clarisse. (no prelo) Retention time as controlling factor of short-term phytoplankton blooms in a shallow microtidal subtropical estuary. Estuarine, Coastal and Shelf Science.

Hübner, Jomi Fred. (2003) Um modelo de reorganização de sistemas multiagentes. Tese de Doutorado. Universidade de São Paulo.

Werlang, Pablo Santos. (2013) Simulação da curva de crescimento do Mycobacterium tuberculosis utilizando sistemas multiagentes, Dissertação, Universidade Federal do Rio Grande/FURG.

Johnson, Steven, (2001) "Emergence: The Connected Lives of Ants, Brains, Cities, and Software." New York: Touchstone.

Wilensky, Uri. (2007) "NetLogo 5.0.4 User Manual." Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, Illinois, USA.

Seeliger, Ulrich, Clarisse Odebrecht. (1997) Introduction and overview In. Seeliger, Ulrich, Clarisse Odebrecht, e Jorge Pablo Castello, eds.Subtropical convergence environments: the coast and sea in the southwestern Atlantic. New York: Springer.

Howarth, R. W., D. P. Swaney, T. J. Butler, e R. Marino. (2000). Climatic control on eutrophication of the Hudson River Estuary. Ecosystems 3: 210–215.

Jassby, A. D. (2008). Phytoplankton in the Upper San Francisco Estuary: Recent biomass trends, their causes and their trophic significance. San Francisco Estuary Watershed Sci. 6: (issue 1, February, article 2) 1–24. Available online at http://repositories.cdlib.org/cgi/viewcontent.cgi?article51103&context5jmie/sfews.

Extensão do JaCaMo para Simulação do Gerenciamento de Projetos de Software

Davy Baia[1], Paulo Alencar[2], Rafael Rocha[1], Carlos P. de Lucena[1]

[1]Departamento de Informática – Pontifícia Universidade Católica (PUC-RIO)
Rua Marquês de São Vicente , 225 – 22.451-900 – Rio de Janeiro – RJ – Brasil

[2] David R. Cheriton School of Computer Science – University of Waterloo
Waterloo - Ontario, Canada

{dbaia;lucena;rrocha}@inf.puc-rio.br, palencar@uwaterloo.ca

Abstract *- Software project management is not a trivial task, especially with the changes that occur throughout the project development process. This leads to an increase in complexity and to the need to deal with dynamic aspects of the project, which makes project management decision making harder. In this paper, we present a multi-agent based simulation approach as a tool to support project managers in their decision-making. In this context, we use the JaCaMo to create an extension that we called JaCaMoPM. We present two outcomes: (i) the representation of processes related the project scope; and (ii) a visualization technique using a Work Breakdown Structure (WBS) to show the simulated dynamics of activity flow sequences.*

Resumo *- Gerenciar um projeto de software não é uma tarefa trivial, especialmente com as mudanças que ocorrem durante o processo de desenvolvimento do projeto. Isto resulta no aumento da complexidade e na necessidade de se lidar com aspectos dinâmicos do projeto, o que dificulta a tomada de decisão de gerentes de projetos. Neste artigo, apresentamos uma abordagem de simulação baseada em sistemas multi-agentes como uma ferramenta para auxiliar gestores de projeto em suas decisões. Nesse contexto, utilizamos o JaCaMo para criar uma extensão que chamamos de JaCaMoPM. Como resultado, temos: (i) um exemplo de representação de processos relacionado ao escopo de um projeto; e (ii) uma técnica de visualização, utilizando uma Estrutura Analítica de Projetos (EAP), para mostrar a dinâmica das sequências de fluxo de atividade na simulação.*

1. Introdução

Gerenciar um projeto de software não é uma tarefa trivial, especialmente com as mudanças que ocorrem durante o processo de desenvolvimento do projeto. Isto resulta no aumento da complexidade e na necessidade de se lidar com aspectos dinâmicos do projeto, o que dificulta a tomada de decisão de gerentes de projetos. Nesse contexto, o gerenciamento de projetos de software envolve inúmeros elementos, incluindo recursos, gerentes de projetos, atividades, as partes interessadas, patrocinadores e o ambiente do projeto. São esses elementos que sofrem diversas alterações durante o tempo de desenvolvimento. Além disso, estes elementos estão relacionados entre si de muitas

maneiras, a fim de resultar nas interações necessárias para o cumprimento das atividades do projeto e na finalização do projeto como o esperado. Assim sendo, torna-se imprescindivel representar e visualizar esses elementos e suas relações para, através de uma simulação e seus resultados, poder apoiar a gestão do escopo do projeto de software em suas tomadas de decisão.

Este artigo é uma continuidade de um trabalho em desenvolvimento que aborda simulação baseada em Sistemas Multi-Agentes (SMA), incorporando os atributos necessários para simular o desenvolvimento de projetos de software e, como consequência, apoiar a gestão de projetos de software [Baia 2015]. A modelagem de simulação baseada em SMA fornece a capacidade de capturar atributos relevantes do processos de desenvolvimento de software e de seus produtos, bem como dos recursos necessários para planejamento, acompanhamento e controle envolvidos na gestão de projeto de software [Agarwal and Umphress 2010].

Em particular na gestão de escopo de um projeto, que é o foco do nosso trabalho, as simulações precisam ser baseadas na representação das relações entre tarefas e suas dependências, seus recursos (humanos ou material), caminhos críticos e sequências de fluxo atividade. Os modelos de simulação também podem ser usados em simulações envolvendo recursos finitos, atrasos (tempo) e os custos estimados. Estas simulações são úteis para gerar cenários (opções) para a tomada de decisão como, por exemplo, cenários relacionadas ao custo e/ou cronograma estimado e, portanto, podem gerar cenários significativos para o gerente de projetos obter mais informações para sua decisões. Os gerentes de projeto, trabalhando em conjunto com suas equipes, podem usar essas simulações para planejar e coordenar seus esforços para que seus projetos tenham os resultado esperados pelas partes interessadas [Agarwal and Umphress 2010]. No entanto, torna-se necessário representar as relações entre os elementos de projeto e também proporcionar um suporte de visualização em um sistema de simulação. A representação pode ser baseada em regras, normas, missões, crenças, desejos e intenções (ou seja, suportar o modelo "Belief-Desire-Intention", BDI). O modelo BDI pode ser apoiado por ferramentas tais como o JaCaMo [Boissier et al. 2013], que usa plataformas que representam (i) os agentes e suas interações, (ii) suas organizações, e (iii) seus ambientes.

Neste artigo, apresentamos uma abordagem de simulação baseada em Sistemas Multi-Agente para gerar cenários, com complexidade e dinamismo do mundo real, que auxilie a gestão de projetos de software na tomada de decisões, com foco nos processos de gerenciamento de escopo. Nesse contexto, apresentaremos um exemplo da nossa abordagem em que oferecemos dois resultados essenciais para apoiar os aspectos da interface humana: (i) a representação dos processos de escopo; e (ii) uma técnica de visualização, utilizando um Estrutura Analítica de Projeto (EAP), para mostrar a dinâmica de sequências de fluxo de atividade. Para avaliar a nossa abordagem, implementamos as representações em JaCaMo, uma estrutura baseada em agentes de software, e, em termos de cobertura, mostramos como seus atributos podem ser mapeados para processos do método de gerenciamento de projetos, baseado no processo de gerenciamento de escopo do PMBOK. Com base em um estudo exploratório e em nossa experiência, acreditamos que estes resultados ajudam a avançar a área de pesquisa que envolve a interseção das áreas de agentes de software e gerenciamento de projetos, especialmente em termos de modelagem e simulação de agentes, organizações e seu ambiente em condições complexas e dinâmicas.

2. Motivação

O trabalho apresentado é uma extensão de uma abordagem de simulação baseada em Sistemas Multi-Agentes [Baia, Lucena, Alencar, Cowan et al. 2014a], que incorpora as práticas descritas no Guia do PMBOK nas áreas de tempo, custo, escopo e recursos humanos abordadas no guia. A abordagem possui um modelo de simulação único que pode ser aplicado aos processos de desenvolvimento de software (por exemplo, para o modelo em cascata). A Figura 1 mostra os componentes necessários para descrever um projeto de software. Em trabalhos anteriores [Baia, Lucena, Alencar, Cowan et al. 2014a], MABS foi aplicado com o ambiente de simulação chamado CORMAS [Bousquet et al. 1998], que fornece suporte aos aspectos sociais e de visualização. No trabalho atual, como citado anteriormente, utilizamos uma extensão do JaCaMo.

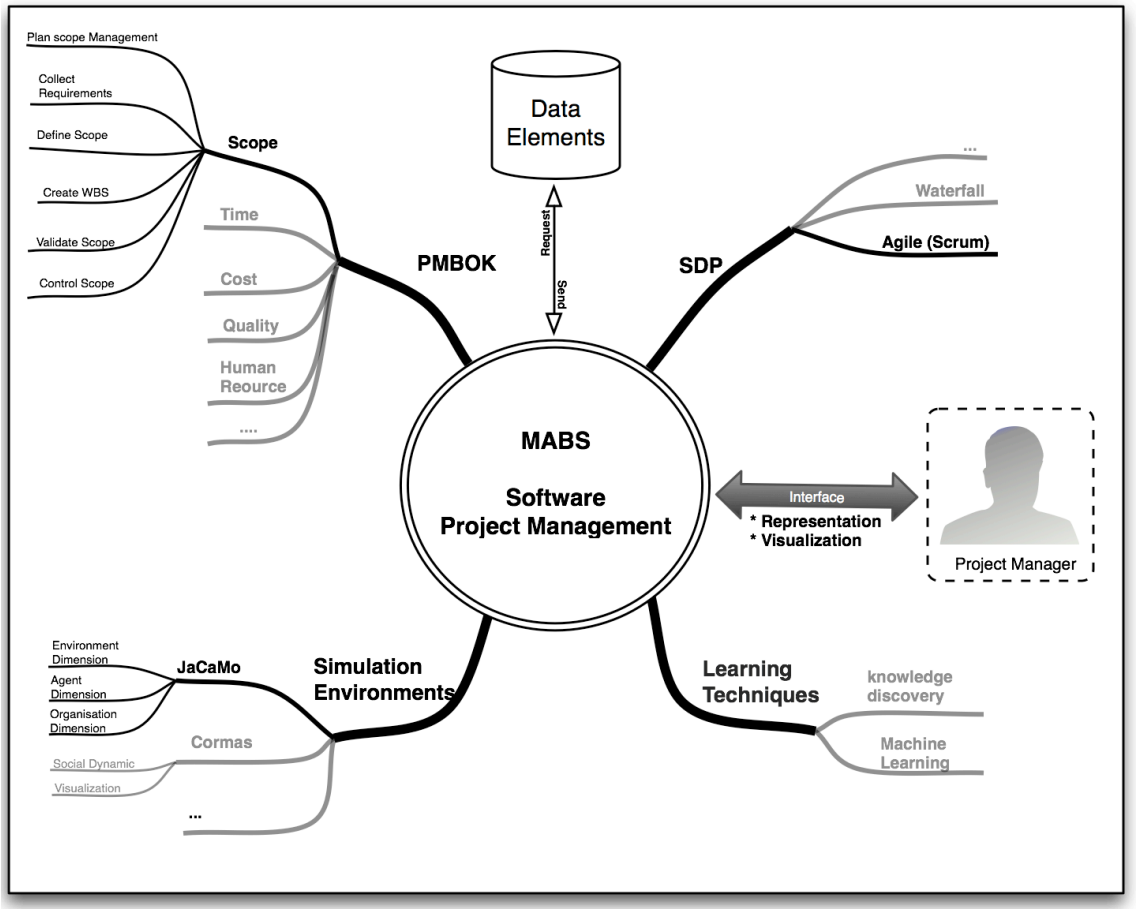


Figura 1. Motivação da abordagem.

A Figura 1 apresenta temas relacionados ao projeto, tais como PMBOK, processos de desenvolvimento de software, ambientes de simulação, e técnicas de aprendizagem e de interface. Neste artigo nos concentraremos nos temas representados pelas linhas mais escuras. Em termos de PMBOK, nos restringiremos a processos de gerenciamento de escopo. Em relação aos ambientes de simulação, estenderemos a plataforma JaCaMo, que suporta as características necessárias em nossa representação. Sobre os processos de desenvolvimento de software, nos concentramos em um método utilizado em nosso laboratório de engenharia de software, que é uma adaptação de um método ágil (SCRUM), mas não entraremos em detalhes sobre esse método por motivo

de espaço para discursão/apresentação. Por fim, como pode ser visto no lado direito central na figura 1, nos concentraremos também nos aspectos de interface, ou seja, representação e visualização. Como um todo, a extensão proposta, ilustrada pelas linhas escuras da figura 1, contribui para melhorar o conjunto de técnicas e ferramentas que podem ajudar os gerentes de projeto em seu processo de tomada de decisão.

3. Gestão de Projetos e Simulação Baseada em Sistemas Multi-Agentes

De acordo com o PMBOK [PMI 2013], "Gerenciamento de Escopo do projeto inclui os processos necessários para garantir que o projeto inclua todo o trabalho necessário, e somente o trabalho necessário, para completar o projeto como esperado". Para o PMBOK o gerenciamento de escopo de projeto de software envolve seis processos: plano de gerenciamento de escopo , coleta de requisitos, definição de escopo, criação da Estrutura Analítica do Projeto (EAP), validação do escopo e controle do escopo. Para cada processo mencionado, podemos formular as entradas, usando o JaCaMo para simular as saídas (cenários), assim criando um novo instrumento de apoio à tomada de decisão para o gerente de projeto. Utilizaremos o processo de definição de escopo para auxiliar o entendimento deste trabalho.

3.1. Simulação Baseada em Sistemas Multi-Agente com JaCaMo

JaCaMo é uma plataforma que suporta programação orientada à multi-agentes. Esta plataforma foi construída sobre três componentes existentes: Jason para a programação de agentes autônomos, Moise para programação de organizações de agentes, e Cartago para programação de ambientes compartilhados. Como resultado, JaCaMo fornece uma perspectiva unificadora para programar agentes, organizações e ambientes [Boissier et al. 2013]. Porém, JaCaMo não oferece a Estrutura Analítica do Projeto (EAP), uma técnica de visualização utilizadas por gerente de projetos. Entretanto, o nosso trabalho tem como resultado uma extensão do JaCaMo, o JaCaMoPM, que cria uma EAP para visualização e acompanhamento da simulação.

De acordo com Boissier et al [Boissier et al. 2013], na dimensão dos agentes em JaCaMo, o agente é uma entidade constituída por (i) um conjunto de crenças que representam o estado atual do agente e conhecimento sobre o ambiente em que se encontra; (ii) um conjunto de objetivos, que correspondem às tarefas que o agente tem que alcançar; e (iii) um conjunto de planos que são cursos de ação, internos ou externos, desencadeadas por eventos que os agentes podem compor de forma dinâmica, instanciar e executar para alcançar seus objetivos. Os eventos podem ser relacionados a mudanças, a base de crenças do agente, ou ainda, aos seus objetivos. Por exemplo, os agentes de recursos disponíveis têm crenças específicas em um determinado estado e realizam determinadas atividades para atingir seus objetivos com base em uma estratégia (ou seja, um plano). Na próxima seção, mostraremos como representar o processo de gerenciamento de escopo e a definição de escopo e, como resultado, teremos uma visualização baseada em uma EAP. Mais detalhes sobre a representação de todos os seis processos relacionado a escopo podem ser encontrados em [Baia, Lucena, Alencar, Rocha et al. 2014b].

3.2 Definição de Escopo

Definir escopo é um processo pertencente à fase de planejamento. Este processo é importante porque uma preparação detalhada da declaração do escopo do projeto se

torna fundamental para o sucesso do projeto. Além disso, este processo contém as principais variaveis de entrada, premissas e restrições que estão documentadas durante o início do projeto. Para isso, podemos usar o JaCaMo e seus recursos para formalizar a definição do escopo (entradas) e suas relações complexas. Por exemplo, podemos usar a documentação de requisitos (uma entrada) para apoiar a criação dessas tarefas. Assim, podemos simular as tarefas e verificar se as metas relacionadas são cumpridas.

```
1 +!sw_Requirement_Specification  <-
      specify_software_requirement.
2 +!software_Prototyping_done  <- softwarePrototype.
```

Figura 2. Atividade por objetivo.

A Figura 2 mostra um exemplo desta formalização. Como já mencionado anteriormente, os objetivos do projeto estão relacionadas a tarefas específicas. Portanto, cada objetivo tem uma lista de tarefas. Por exemplo, na Figura 2, para cumprir com a meta *sw_Requirement_Specification* a tarefa *specify_software_requirement* precisa ser executada. Assim, podemos usar estas especificações para apoiar a criação da declaração do escopo do projeto e para atualizar os documentos do produto (por exemplo, o registro das partes interessadas ou a documentação de requisitos). O objetivo desse processo é desenvolver uma descrição detalhada do projeto e do produto. De acordo com o PMBOK [PMI 2013], o principal benefício desse processo é que ele descreve os limites do produto, serviço ou resultado, definindo quais dos requisitos coletados serão incluídos ou excluídos do escopo do projeto.

3.3 Estrutura Analítica do Projeto e Dinâmica dos Estados

A Figura 3 mostra as simulações sendo executadas. O simulador cria uma EAP com todas as tarefas que devem ser realizadas. Desta forma, a simulação pode fluir a partir de uma tarefa para outra à medida que cada tarefa é executada. Nesta figura, cada caixa azul representa uma tarefa agendada, que tem o nome da tarefa e não têm inicialmente o nome de recurso associado. Depois de uma tarefa ser executada, a sua caixa muda de cor (de azul para verde) e também é marcado o nome do recurso que completou a tarefa. Deste modo, não só o método proposto pode suportar a representação das características necessárias nos processos de gestão de escopo, como também pode proporcionar uma visualização de uma EAP. Uma característica especial da visualização EAP é que ela permite que os gerentes de projeto, através da alteração da cor, possam acompanhar a dinâmica da sequência das atividades, o que ajuda a apoiar a validação e controle do escopo do projeto.

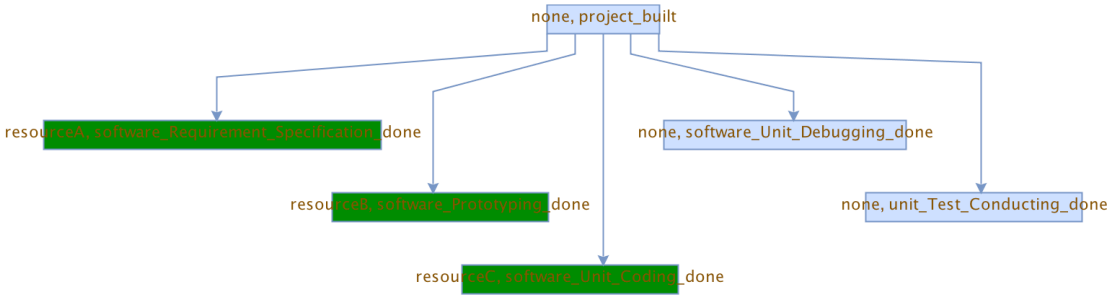


Figura 3. EAP criado para simulação.

4. Conclusões e Trabalhos Futuros

Neste artigo apresentamos um resumo da nosso abordagem de simulação baseada sistemas multi-agentes para auxiliar gestores de projetos em suas decisões, com foco nos processos de gerenciamento de escopo. Nesse contexto apresentamos um exemplo implementado em JaCaMo. Porém, foi necessário fazer uma extensão no JaCaMo para suportar a representação de atributos do gerenciamento de escopo e uma técnica de visualização relacionada com a EAP. Chamamos esta extensão de JaCaMoPM.

Para avaliar a nossa abordagem, implementamos um exemplo de representação baseado em agentes e utilizando o processo de definição de escopo do PMBOK. Ilustramos também uma Estrutura Analítica do Projeto (EAP), que refere-se às técnicas de visualização usadas para mostrar a dinâmica de sequências de fluxo de atividade. A criação da EAP é um diferencial que o JaCaMoPM possui, além de fornecer uma abordagem orientada para simulação de projetos de software, e recursos valiosos para apoiar os gestores do projeto durante todo o processo de desenvolvimento de software. Com base em nossa experiência, acreditamos que estes resultados ajudam a avançar a área de pesquisa que envolve a interseção entre as áreas de agentes e gerenciamento de projetos, especialmente em termos da modelagem e simulação complexa e dinâmica de situações envolvendo agentes, suas organizações e ambientes.

Referencias

Agarwal, R. and Umphress, D. (2010). A Flexible Model for Simulation of Software Development Process. *Proceedings of the 48th Annual Southeast Regional Conference*,

Baia, D. (2015). An Integrated Multi-Agent-Based Simulation Approach to Support Software Project Management. *37th International Conference on Software Engineering - ICSE*, (accept submitted), p. 1–4.

Baia, D., De Lucena, C., Alencar, P., Cowan, D., et al. (2014a). A MultiAgent-Based Simulation Model to Support Management Decision Making in Software Development. *Technical Report, David R. Cheriton School of Computer Science, University of Waterloo*, p. 1–9.

Baia, D., De Lucena, C., Alencar, P., Rocha, R., et al. (2014b). MultiAgent-Based Simulation in Software Project Management: Scope Management Represetantion and Visualization. *Technical Report, David R. Cheriton School of Computer Science, University of Waterloo*, p. 1–8.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A. and Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, v. 78, n. 6, p. 747–761.

Bousquet, F., Bakam, I., Proton, H. and Le Page, C. (1998). Cormas: Common-pool resources and multi-agent systems. p. 826–837.

PMI (17 aug 2013). Project Management Body Of Knowledge (PMBOK®) Guide. p. 1–418.

Uma Ferramenta para a Modelagem de Sistemas Multi-agentes Culturais

Igor Hideki Trindade[1], Karen da Silva Figueredo[2]

Instituto de Computação – Universidade Federal do Mato Grosso (UFMT)
Cuiabá, Mato Grosso, Brasil

igorhideki@gmail.com[1], karen@ic.com.br[2]

Abstract. *The inclusion of cultural elements in open multi-agent systems supports the sharing of behavior patterns among the system agents by promoting coordination and cooperation between them. As consequence, the inclusion of cultural elements increases the complexity of these systems in order to modeling them. This paper proposes a Multi-Agent Systems design tool which includes cultural elements and it is able to identify possible inconsistencies in the models created in design time.*

Resumo. *A inclusão de elementos culturais em Sistemas Multi-agentes abertos facilita o compartilhamento de padrões de comportamento entre os indivíduos, promovendo a coordenação e cooperação entre eles. Por sua vez, a inclusão dos elementos culturais aumenta a complexidade destes sistemas, tornando as atividades de design mais elaboradas. Para auxiliar na modelagem de tais sistemas, este artigo propõe uma ferramenta de design de Sistemas Multi-Agentes que inclui elementos culturais e é capaz de identificar possíveis inconsistências nos modelos em tempo de design.*

1. Introdução

Em computação, agente é um software ou parte de um software capaz de perceber o que ocorre no ambiente em que está situado, tendo autonomia para realizar um repertório de possíveis ações, podendo assim alterar o ambiente em que habita. O sistema que possui um ou mais agentes é chamado de Sistema Multi-agentes (SMA). Em um SMA, os agentes nele contidos interagem entre si podendo colaborar, cooperar ou competir para alcançar um objetivo em comum [Bordiniet al. 2007].

Assim como existe o Paradigma Orientado a Objetos, com a emergência dos SMA surge o Paradigma Orientado a Agentes e junto com ele a necessidade de desenvolvimento de ferramentas para apoio às atividades de análise, projeto, desenvolvimento e teste desse tipo de sistema [Jennings 1999].

Um SMA que permite que os agentes em seu sistema sejam autônomos ao mesmo tempo que estes agentes agregam comportamentos sociais do seu grupo através da cultura construída por eles pode ser caracterizado como um SMA cultural [Marques e Figueiredo 2014]. Culturas são definidas como coleções de padrões de comportamento que são aprendidos, transmitidos e compartilhados com o objetivo de adaptar os indivíduos de um grupo [Keesing 1974].

Em SMA, quando conceitos culturais como papéis, normas e valores são incluídos é possível promover a coordenação, cooperação e autonomia entre os

indivíduos do sistema [Antunes 1997]. Por sua vez, a inclusão destes elementos aumenta a complexidade destes sistemas, tornando mais difíceis as atividades de *design*.

Desta forma, o objetivo deste trabalho é contribuir com uma ferramenta de modelagem que permita a criação de modelos de SMA Culturais por analistas de sistemas orientados a agentes, que seja ainda capaz de identificar possíveis inconsistências e conflitos nos modelos em tempo de *design*.

2. Métodos e Técnicas

A ferramenta proposta foi desenvolvida como um *plugin* para o ambiente de desenvolvimento Eclipse, através da própria Eclipse IDE[1]. O Eclipse foi escolhido como IDE a ser utilizada como alvo e no desenvolvimento de nossa ferramenta pensando em sua ampla utilização por analistas e desenvolvedores de sistemas, permitindo que os mesmos trabalhem em um ambiente integrado nas etapas de *design* e implementação. A versão do Eclipse utilizada neste trabalho foi a Luna.

Para a construção da ferramenta também foi utilizado o *plugin* GMF. O GMF (*Graphical Modeling Framework*[2]) é um framework para desenvolvimento de editores gráficos para modelos de domínio dentro da plataforma Eclipse. Ele foi baseado em outros dois frameworks denominados GEF (*Graphical Editing Framework*), utilizado para a criação de editores gráficos genéricos e EMF *(Eclipse Modeling Framework*), que permite ao desenvolvedor construir metamodelos e gerar códigos Java referidos ao mesmo, desta forma foi possível construir uma ferramenta gráfica de modelagem que é consistente com um metamodelo bem definido.

O metamodelo descrito como entrada no EMF foi elaborado com base na ontologia para SMA Culturais proposta por Marques e Figueiredo (2014) que descreve todos os elementos populares em SMA Culturais e os relacionamentos entre eles.

3. CSMA Modeling Tool

O resultado obtido com a realização deste trabalho é a ferramenta de modelagem de SMA culturais *CSMA Modeling Tool*. Utilizando a ferramenta proposta, analistas de sistemas orientados a agentes podem realizar a modelagem das entidades do SMA e seus relacionamentos de acordo com a notação gráfica descrita a seguir elaborada com base na ontologia para SMA Culturais de Marques e Figueiredo (2014).

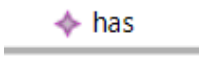
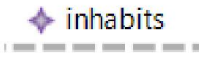
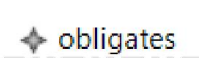
Tabela 1. Notação gráfica das entidades da CSMA Modeling Tool

					
Environment (Ambiente)	***Organization (Organização)***	***Culture (Cultura)***	***Agent (Agente)***	***Role (Papel)***	***Belief (Crença)***

[1] http://eclipse.org
[2] http://eclipse.org/modeling/gmp/

				
Goal (Objetivo)	***Plan (Plano)***	***Action (Ação)***	***Norm (Norma)***	***Value (Valor)***

Tabela 2. Notação gráfica dos relacionamentos da CSMA Modeling Tool

Relacionamento	*Nome*	*Entidades*
	***Has (Tem)***	Entre Agentes e Crenças/Objetivos/ Planos/Ações/ Normas/Valores, Ambientes/Papéis e Normas, Organizações e Normas/Cultura, Culturas e Papéis/Crenças/Valores, Planos e Ações/Objetivos
	***Inhabits (Habita)***	Entre Agentes e Organizações, Organizações e Organizações, Organizações e Ambientes
	***Plays (Desempenha)***	Entre Agentes e Papéis, Organizações e Papéis
	***Demotes (Rebaixa)***	Entre Valores e Ações
	***Promotes (Promove)***	Entre Valores e Ações
	***Obligates (Obriga)***	Entre Normas e Ações
	***Permits (Permite)***	Entre Normas e Ações
	***Prohibits (Proíbe)***	Entre Normas e Ações

A ferramenta funciona como um *plugin* para o Eclipse a sua aparência final é apresentada na Figura 1, onde podemos observar a área de modelagem à esquerda e a paleta de entidades e relacionamentos à direita, bastando ao analista arrastar os elementos da paleta para a área de modelagem para criar os mesmos.

No modelo da Figura 1, está representado um agente (*Igor*) que habita uma organização (*Computer Science Department*), que possui uma cultura (*CSDCulture*) e habita um ambiente (*UFMT*). Este agente desempenha um papel (*Computer Science Student*) da cultura desta organização, que possui uma norma (*N1*) que obriga a executar uma ação (*PresentInternshipReport*).

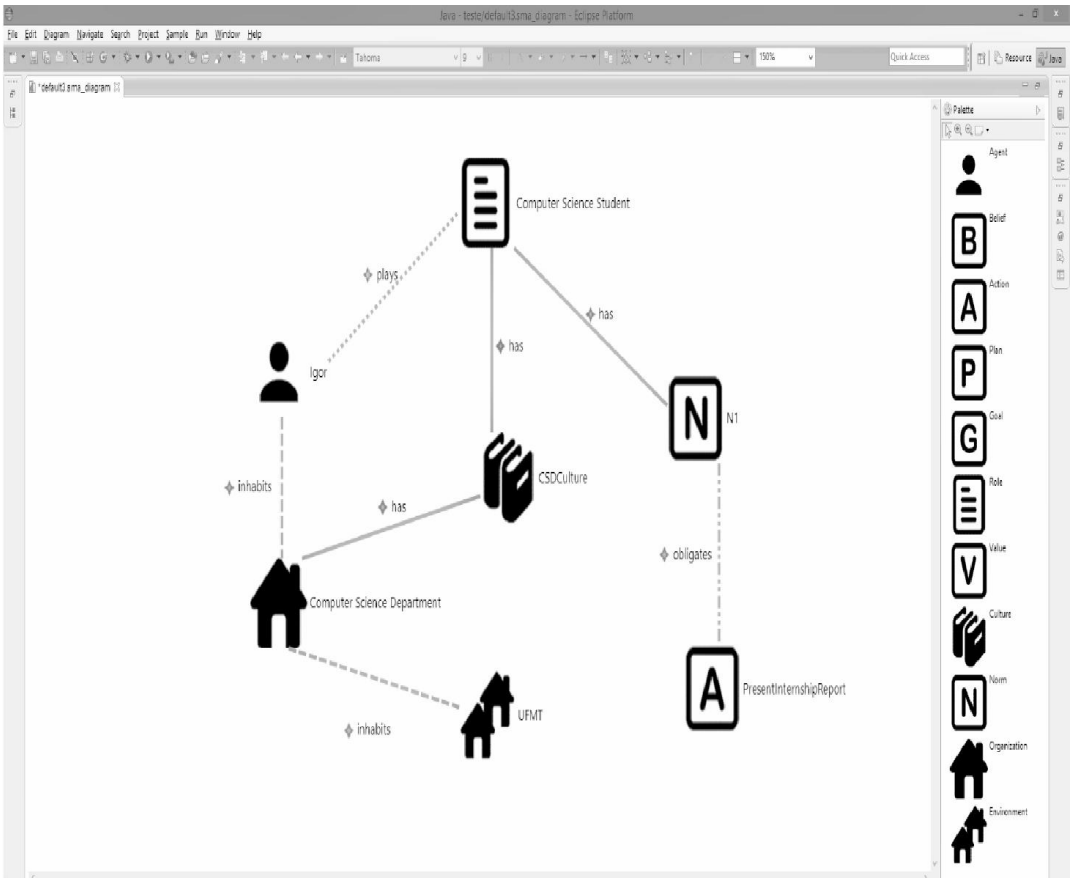


Figura1. Exemplo de uma modelagem na ferramenta

O modelo de domínio (metamodelo) utilizado para a criação da ferramenta foi baseado na ontologia apresentada por Marques e Figueiredo (2014), respeitando todas as classes, relacionamentos e cardinalidades presentes na ontologia. Desta forma, quando o analista tenta criar um relacionamento que não é possível (considerando as regras da ontologia/metamodelo - ver Tabela 2) entre duas entidades, a ferramenta identifica a possível inconsistência e impede que o relacionamento seja definido.

4. Conclusão

O trabalho exposto baseia-se em conceitos relacionados a agentes, SMA, ontologias e modelagem além de trabalhar com as ferramentas: Eclipse IDE, Linguagem de programação Java, e *GraphicalModeling Framework* (GMF). Este trabalho foi o resultado preliminar da disciplina de Estágio Supervisionado do aluno de graduação do curso de Ciência da Computação autor deste artigo.

Ao realizar este trabalho o objetivo principal era desenvolver uma ferramenta de modelagem que permitisse a criação de modelos de SMA Culturais por analistas de sistemas, pois até então não existiam ferramentas com esta finalidade. Este objetivo foi alcançado com o estudo dos conceitos e a utilização das ferramentas citadas acima. A ferramenta desenvolvida e apresentada neste trabalho está disponível para download[3].

[3] CSMA Modeling Tool - https://www.dropbox.com/s/jhkte80l1d16z39/CSMAModelingTool.zip?dl=0

Espera-se que esta nova ferramenta auxilie os analistas de sistemas e estudantes da área de computação em suas criações de modelos de SMA Culturais e que possa ajudar no crescimento deste novo tema, auxiliando na criação de trabalhos futuros relacionados e também realizando melhorias na própria ferramenta e realizando testes com analistas em diversos cenários.

Referências

Antunes, L. (1997) "Towards a model for value-based motivated agentes", In: Proceedings of MASTA97.

Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason* (Vol. 8). John Wiley & Sons.

Jennings, N. R. (1999). Agent-oriented software engineering. In *Multiple Approaches to Intelligent Systems* (pp. 4-10). Springer Berlin Heidelberg.

Keesing, R. M. (1974) "Theories of culture", In: Annual Review of Anthropology 3, pp.73 – 97.

Marques, V. F.; Figueiredo, K. S (2014). Uma Ontologia para a Representação de Sistemas Multiagentes Culturais. In *Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, (pp. 149-154). Porto Alegre: PUCRS.

Simulação da dispersão de ovas e larvas da Garoupa-Verdadeira utilizando simulação baseada em agentes

Tiago F. Otero[1], Diana F. Adamatti[1]

[1]Centro de Ciências Computacionais – Universidade Federal do Rio Grande (C3/FURG)
Av. Italia, s/n km 08 – Rio Grande – RS – Brazil

{tiagofotero, dianaada}@gmail.com

Resumo. *Este artigo apresenta um modelo de simulação baseado em agentes. Este modelo é biologicamente inspirado na dispersão das ovas e larvas da Garoupa-Verdadeira, uma espécie de peixe encontrada nas proximidades da cidade de Rio Grande, em meio ao ambiente oceânico. O foco de estudo é demonstrar como ocorre a dispersão das ovas e larvas da Garoupa-Verdadeira entre o Parcel do Carpinteiro e os Molhes da Barra, em Rio Grande/RS.*

1. Introdução

Ao longo da história, a observação do mundo natural permitiu que o ser humano pudesse criar muitas teorias sobre o comportamento das diversas partes que compõe este ambiente. Como exemplo, pode-se citar a lei da física de Newton, ou o modelo de órbita planetária de Kepler. Além da criação de teorias, produtos foram criados baseados em princípios observados na natureza e na biologia. O desenvolvimento e progresso da ciência da computação, da engenharia e das tecnologias em geral contribuiu significativamente para o estudo e entendimento do mundo natural [Castro e Timmis, 2002].

Segundo Castro e Timmis (2002), os conhecimentos nas áreas de engenharia e de computação são enriquecidos por meio da introdução de processos e funções baseados em modelos obtidos através da modelagem e simulação de ambientes naturais. Muitos problemas computacionais foram solucionados adotando-se soluções baseadas na biologia, por exemplo, os algoritmos neurais, autômatos celulares e a bioinformática. Segundo os autores estas soluções podem receber o nome de computação biologicamente motivada.

A garoupa-verdadeira (*Mycteroperca marginata*) é uma espécie de peixe de médio e grande porte encontrada em diversos locais da costa brasileira. Esta espécie é incomum em zonas rasas ao longo da costa gaúcha, provavelmente porque os fundos arenosos que predominam na região sul não são habitats propícios para a sua ocorrência. No entanto, os molhes da barra de Rio Grande constituem uma importante exceção por constituir um *habitat* adequado para que a garoupa exista. Este peixe apresenta hábitos territorialistas e sedentários, vivendo associado a fundos rochosos comuns em águas costeiras, especialmente em parcéis, recifes e costões rochosos. Além dos molhes da barra, existe um local próximo no qual há uma grande população de garoupas. Este local é o Parcel do Carpinteiro, que é um alto topográfico submerso situado na antepraia do

Rio Grande do Sul. Ele está localizado próximo a desembocadura da Laguna dos Patos a aproximadamente 40 quilômetros dos Molhes da Barra, na cidade de Rio Grande. Estudos sobre a conectividade populacional, a troca de indivíduos da mesma espécie entre unidades espaciais, revelam que a garoupa-verdadeira não se reproduz na região litorânea dos molhes da barra, mas sim na região do Parcel do Carpinteiro [Seyboth et al., 2011]. Pesquisas sugerem que a reprodução de garoupas não ocorre nos molhes da barra e, assim sendo, para que essa região seja habitada por esses peixes, ela depende da dispersão de indivíduos provenientes do Parcel do Carpinteiro [Condini, 2012]. Baseando-se nos hábitos dessa espécie, é pouco provável que as garoupas em fase adulta migrem entre as duas regiões. Por essa razão, sabe-se que as ovas e larvas da garoupa, através da dispersão da corrente oceânica, acabem por chegar aos molhes da barra.

Este artigo apresenta um estudo inicial sobre a dispersão das ovas e larvas da Garoupa-Verdadeira entre o Parcel do Carpinteiro e os molhes da barra do Rio Grande/RS, utilizando uma simulação baseada em agentes como ferramenta computacional para simular este fato. Simulações baseadas em agentes podem ser utilizadas para ilustrar uma situação que ocorra no mundo real. Tais simulações podem ser utilizadas para simples observação, como elementos no apoio a decisão em diversos problemas. Segundo Adamatti (2007), muitas aplicações em sistemas baseados em agentes são desenvolvidas com o objetivo de simular alguma situação da realidade, observando-se o fato real e utilizando-se como base uma teoria.

2. A Garoupa-Verdadeira

A Garoupa-Verdadeira (*Mycteroperca marginata*) é uma espécie marinha que possui uma larga distribuição geográfica, ocorrendo em diversos lugares distribuídos pelo mundo. Segundo Figueiredo e Menezes (1980), na porção ocidental do oceano atlântico, sua distribuição é restrita á costa sul da América do Sul, que se inicia no estado do Rio de Janeiro e vai até a Argentina. Esta espécie possui corpo robusto, podendo atingir um comprimento de 120 cm e um peso de 60 Kg. A Garoupa é uma espécie de hábitos sedentários e solitários, principalmente nos indivíduos maiores de 20 cm, os quais permanecem a maior parte do tempo em suas tocas, saindo apenas para alimentar-se [Condini, 2012]. Enquanto juvenis, menores que 20 cm, preferem áreas abertas, vagando a procura de alimento, podendo formar pequenos grupos de até três indivíduos. Seus hábitos alimentares são predominantemente carnívoros, sendo uma espécie predadora de crustáceos, peixes e moluscos [Condini, 2012].

Segundo Seyboth et al. (2011), o crescimento populacional da garoupa-verdadeira é baixo devido a sua estratégia complexa de reprodução ser hermafrodita protogínica monântrica, ou seja, todos os indivíduos nascem fêmeas e, a partir de um determinado tamanho (geralmente acima dos 52 cm), ou devido a pressão populacional, entre os 7 a 17 anos de idade, pode ocorrer a reversão sexual para macho [Bruslé e Bruslé, 1975].

De acordo com a União Internacional pela Conservação da Natureza (IUCN), a garoupa-verdadeira é considerada, atualmente, como uma espécie ameaçada de extinção. Segundo Fennessy (2006), os fatores que levam a esta espécie a esta condição são a sua

sobre-exploração facilitada pelos seus hábitos territorialistas e sedentários, seu baixo crescimento populacional, sua estratégia reprodutiva e, finalmente, sua maturação tardia.

A garoupa-verdadeira é incomum em zonas rasas ao longo da costa gaúcha, pois os fundos arenosos que predominam nessa região não são habitats propícios para a sua ocorrência [Condini, 2012]. No entanto, os molhes da barra de Rio Grande constituem uma importante exceção devido ao seu substrato rochoso constituir um habitat adequado para que a garoupa-verdadeira ocorra em abundância suficiente para permitir sua pesca regular. Mesmo que dados científicos precisos não estejam disponíveis, contagens históricas obtidas de pescadores locais sugerem que as garoupas veem sendo capturadas nesta localidade desde a década de 70 [Condini, 2012[. Os molhes da barra de Rio Grande são uma formação rochosa criada artificialmente na desembocadura da Lagoa dos Patos no início do século 20, que avançam 4,5 Km no Oceano Atlântico. Seu principal propósito é de manter um canal de navegação e garantir um acesso marítimo regular ao porto internacional de Rio Grande.

3. Modelo Proposto

O modelo proposto representa a dispersão destas partículas, ovas e larvas, em meio ao ambiente oceânico. O movimento realizado por estas partículas é influenciado por diversos fatores que simulam este ambiente oceânico, como a maré e o sentido e intensidade dos ventos. O modelo foi implementado utilizando o software NetLogo. São três os parâmetros que influenciam na dispersão: número de ovas, ângulo do vento e intensidade do vento, todos podem ser setados pelo usuário ao início da simulação. O primeiro varia entre 1 e 25000; o segundo entre 0 360; e o terceiro entre 0 e 120.

A cada *tick*, medida de iteração do Netlogo, diversos fatores influenciam as partículas (ovas e larvas da Garoupa-Verdadeira). Inicialmente, todas as partículas iniciaram seu movimento com seu sentido sendo igual a do ângulo do vento setado pelo usuário. Porém, com o desenvolvimento da simulação, outros fatores podem modificar esta medida. A cada iteração, as partículas se moverão para frente, no sentido para a qual estão atualmente apontadas, de acordo com a intensidade do vento. Além do deslocamento para frente, as partículas sofrem uma alteração no ângulo do sentido para a qual estão apontando a cada tick. Esta alteração ocorre de forma a simular a maré, que tende a encaminhar as partículas para as proximidades da beira do mar. A intensidade com a qual o sentido das partículas é alterada, depende de qual é o seu sentido atual. Se uma partícula tem um sentido oposto a maré, a intensidade da alteração de seu sentido atual será maior. A Figura 1 apresenta como funciona a “compensação” para simular a maré.

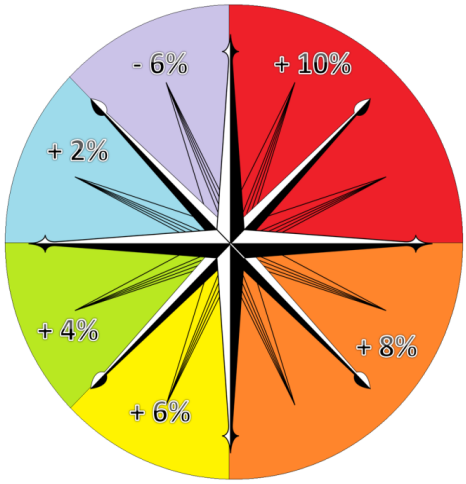


Figura 1: Intensidades de alteração do ângulo de sentido das partículas

Por exemplo, pode-se dizer que se o sentido de uma partícula está contida no intervalo compreendido entre 90 e 180, seu sentido receberá um acréscimo de 8% no seu valor atual. Na próxima iteração, seu valor de sentido será novamente testado e receberá uma nova alteração correspondente ao intervalo no qual se enquadra.

4. Análise dos Resultados Obtidos

Foram realizadas simulações de alguns cenários, tentando abranger diversas situações possíveis para as variáveis da simulação. Dependendo da forma com a qual o usuário configurar a simulação, as partículas podem atingir os molhes da barra, ou não. Algumas configurações, mesmo com a força favorável da maré, não atingiram o objetivo de alcançar a localidade desejada, quando executadas. Por exemplo, em um cenário no qual ocorra um vento de baixa intensidade no sentido sul, as partículas conseguiram alcançar uma localidade próxima aos Molhes da Barra. Em outro caso, se o vento for configurado como possuindo sentido norte e uma alta intensidade, as partículas não iram alcançar a localidade desejada.

A Tabela 1 contém os diversos cenários simulados. Cada linha representa um possível cenário que consiste em quatro colunas. A primeira coluna é o sentido do vento que é baseado nos principais pontos cardinais; a segunda coluna é a intensidade do vento, que pode variar entre leve (20), moderada (60) e intensa (100); a terceira coluna é a avaliação. Se a avaliação for favorável, indica que o resultado da simulação deste cenário é satisfatório, ou seja, as partículas se aproximam da região dos molhes da barra; no caso de avaliação desfavorável, as partículas não se aproximaram dos molhes da barra para o cenário que foi simulado. E a quarta coluna corresponde a medida de tempo (ticks).

Tabela 1: Resultados dos cenários simulados

Sentido do Vento	Intensidade do Vento	Avaliação	Tempo
Norte (0 ou 360)	Leve	Desfavorável	220
Norte (0 ou 360)	Moderada	Desfavorável	29

Norte (0 ou 360)	Intensa	Desfavorável	13
Nordeste (45)	Leve	Favorável	235
Nordeste (45)	Moderada	Favorável	58
Nordeste (45)	Intensa	Desfavorável	38
Leste (90)	Leve	Desfavorável	225
Leste (90)	Moderada	Favorável	44
Leste (90)	Intensa	Favorável	34
Sudeste (135)	Leve	Favorável	218
Sudeste (135)	Moderada	Favorável	36
Sudeste (135)	Intensa	Desfavorável	26
Sul (180)	Leve	Favorável	207
Sul (180)	Moderada	Favorável	27
Sul (180)	Intensa	Desfavorável	17
Sudoeste (225)	Leve	Favorável	213
Sudoeste (225)	Moderada	Favorável	28
Sudoeste (225)	Intensa	Favorável	13
Oeste (270)	Leve	Desfavorável	219
Oeste (270)	Moderada	Desfavorável	28
Oeste (270)	Intensa	Desfavorável	13
Noroeste (315)	Leve	Desfavorável	189
Noroeste (315)	Moderada	Desfavorável	26
Noroeste (315)	Intensa	Desfavorável	14

A partir da análise dos resultados obtidos, pode-se observar alguns padrões, como a relação inversamente proporcional entre a intensidade do vento e o número de ticks necessários para cada cenário. Quanto maior a intensidade do vento, menor o número de ticks necessário. O número de ticks é influenciado também pelo sentido do vento. Alguns sentidos de vento, como sul e sudoeste, possuem uma maior tendência a convergir para as proximidades dos molhes da barra. Este fator torna a execução da simulação mais rápida, pois as partículas atingem seu objetivo mais rapidamente.

5. Conclusões e Trabalhos Futuros

Os sistemas baseados em agentes são soluções computacionais que possibilitam criar simulações para diversos campos de conhecimento com a biologicamente inspirada.

A Garoupa-Verdadeira é uma espécie emblemática da fauna brasileira, sendo escolhida para ilustrar a nota de 100 reais, exatamente por ser um dos peixes mais conhecidos da costa nacional. Em Rio Grande, existem dois habitats da Garoupa-Verdadeira. O primeiro é a localidade dos molhes da barra, que é um local composto de um substrato rochoso criado artificialmente na desembocadura da Laguna dos Patos. O segundo é o Parcel do Carpinteiro. Esta localidade consiste em um elevado topográfico submerso situado a 16 milhas náuticas da costa.

Com o intuito de simular este ambiente de dispersão de ovas e larvas da Garoupa-Verdadeira entre seus dois habitats em Rio Grande, foi criada uma simulação baseada em agentes, no software Netlogo. Esta simulação demonstra o movimento destas partículas em meio ao ambiente oceânico utilizando parâmetros informados pelo usuário.

Como trabalhos futuros, havendo dados reais, coletados por pesquisadores do Instituto de Oceanografia da FURG, seria possível realizar uma simulação mais realista do problema abordado. Estes dados podem conter novas variáveis que influenciem no ambiente oceânico, como o sentido exato de maré, temperatura e salinidade da água.

Referências

Adamatti, D. F.. Inserção de Jogadores Virtuais em Jogos de Papéis para Uso em Sistemas de Apoio a Decisão em Grupo: Um experimento no Domínio da Gestão de Recursos Naturais. Tese de Doutorado em Engenharia Elétrica – Universidade de São Paulo, 2007.

Bruslé, J. e S. Bruslé. “Ovarian and testicular intersexuality in two protogynous mediterranean groupers, Epinephelus aeneus and Epinephelus guaza.” Intersexuality in the animal kingdom, 1975: 222-227

Castro, L. N. e Timmis, J. Artificial Immune Systens: A new computational Intelligence Approach. Canterbury: Springer, 2002.

Condini, M. V.. Biologia reprodutiva, determinação de estrutura etária e crecimento da Garoupa-Verdadeira Epinephelus marginatus (Lowe, 1983) em fundos rochosos no extremo sul do Brasil. Rio Grande. Dissertação de Mestrado em Oceanografia – Universidade Federal do Rio Grande, 2012.

Fennessy, Y. S. T. “Reprodutive biology and grouth of the yellowbelly rockcod Epinephelus marginatus (Serranidae) frin South-east Africa.” African Journal of Marine Science, 28 de 1 de 2006: 1-11.

Figueiredo, J. L. e N. A Menezes. Manual de peixes marinhos do sudeste do Brasil. São Paulo: Teleostei, 1980.

Seyboth, E., et al. Age, Grouth, and reproductive aspects of the dusky grouper Mycteroperca marginata (Actinopterygii: Epinephelidae) in a man-made rocky habitat in southern Brazil. Neotrop. ichthyol., vol.9, no.4, Porto Alegre, 2011.

A Framework for Supporting Simulation with Normative Agents

Marx Viana[1], Francisco Cunha[1], Baldoino Santos Neto[2], Paulo Alencar[3], Carlos J. P. de Lucena[1]

[1] Pontifical Catholic University (PUC-Rio) - Rio de Janeiro – RJ, Brasil

[2]Federal University of Alagoas - Alagoas – AL, Brasil

[3]University of Waterloo - Waterloo - Ontario, Canada

{mleles, fplacido, lucena}@inf.puc-rio.br, baldoino@ic.ufal.br, palencar@uwaterloo.ca

Abstract. *Multi-agent systems are societies in which autonomous agents work to achieve both common and individual goals. In this context, norms have been used as mechanisms to regulate the behavior of such agents and ensure a desirable social order where agents can work together. Although norms are a promising mechanism, it is necessary to evaluate the positive and negative impact of using them. Therefore, this work presents a framework for normative multi-agent systems simulation that provides the mechanisms necessary to understand the impact of norms on agents and the society they belong to.*

1. Introduction

Multi-agent systems are societies where entities, known as agents, are autonomous, heterogeneous and can work to achieve common or disparate goals [Wooldridge 2011]. In order to deal with autonomy and diversity of interests among the different members, such systems provide a set of norms that is used as a social control mechanism to ensure that a desirable social order in which agents can work is maintained [Silva 2008]. Although norms are promising mechanisms to regulate the behavior of agents, one should take into account that they are autonomous and, therefore, free to decide to fulfill or violate each system norm. This type of agent reasoning refers to normative strategies.

However, in order to discourage violations of specific norms by the agents, designers can provide rewards when the agents fulfill these norms and punishments when the norms are violated [Silva 2008]. In this context, there is a need for mechanisms that enable designers to develop and verify the impact of norms (i.e., rewards and punishments) as well as normative strategies not only on individual agents, but also on the whole society.

This paper proposes a framework (Normative Agent Java Simulation Framework, which we call JSAN) that supports the evaluation of norms through normative multi-agent system simulations. The JSAN framework provides the necessary mechanisms to understand the impact of norms on agents that adopt some given strategies to deal with these norms. These mechanisms enable (i) implementing of different types of normative strategies; (ii) collecting information about the simulated environment; (iii) displaying information about the simulated scenarios; (iv) supporting

the implementation of different movement strategies for the agents in the environment; (v) generating norms; and (vi) generating agent goals. This framework extends of JASON [Bordini et al. 2007], which enables the development and implementation of BDI agents (Belief, Desire, and Intention) [Machado and Bordini 2002]. We plan to evaluate the applicability of the JSAN Framework in two scenarios: the first is related to the mission of rescuing civilians who are in geoenvironmental risk areas [Cerqueira et al. 2009] and the second involves crime prevention [Bosse and Gerritsen 2010].

2. Motivation

The creation of JSAN Framework was due to the need to develop simulations in which it would be possible to insert norms to restrict the behavior of agents in a given environment. In addition, JSAN provides the necessary mechanisms for agents to be able to understand and deal with these norms. In the next paragraphs two scenarios are described to illustrate the application of JSAN to assist in decision-making, showing possible cases where norms are inserted to restrict the behavior of agents in the environment.

In a first application scenario, the simulation framework can be used to support the rescue of civilians who are in geoenvironmental risk areas. It is known that landslides are natural phenomena that are often difficult to predict since they depend on many factors (e.g., slope angle, climate, water content, vegetation) and complex relationships between these factors. The annual number of landslides is in the thousands in the city of Rio de Janeiro, and the infrastructure damage is worth more than a billion dollars [Santos Neto and Lucena 2010]. In this setting, there is a need to build platforms to assist experts in two areas, namely the analysis of geoenvironmental risk areas and the evacuation planning of civilians located in these areas [Cerqueira et al. 2009].

The evacuation planning can be assisted by simulations using the JSAN Framework, which aims to implement scenarios involving the creation of situations where civilians are in geoenvironmental risk areas and, thus, to provide different strategies for firefighter agents, which are regulated by norms, to rescue these civilians. The simulations are normative multi-agent systems that receive data about the geoenvironmental risk areas, such as weather conditions, information about the existence of civilians in geoenvironmental risk areas, redemption forms for the withdrawal of civilians from these risky locations (with troops, land vehicles or aircraft), norms that firefighter agents must follow during the rescue operation, and rescue plans to be used in the simulation. The supplied data is received by the Manager Agent (Chief Fireman) responsible for sending this information to the firefighter agents who are able to find different solutions to evacuate civilians from geoenvironmental risk areas.

In a second application scenario, the simulation framework can be used in the context of crime prevention, in which an important challenge is the analysis of the displacement of crime. Some of the many problems in this area relate to the prediction and prevention with respect to areas with a potentially high crime rate, making the study of the displacement of crime a promising research area [Bosse and Gerritsen 2010]. Certain types of crimes typically happen around specific locations in a city, especially in streets nearby shopping malls, train stations and highways. These areas are known as sites with high crime rates. However, these sites do not stay the same for a long period.

A number of known factors can cause the displacement of these criminality sites to another region. For example, the deployment of cameras in train stations can cause crime rates to decrease in that area [Bosse and Gerritsen 2010]. The insertion of norms in crime prevention support aims to regulate the actions of the police agents [Bosse and Gerritsen 2010]. For example, such norms may influence them to act to prevent the largest possible number of robberies, but not to think about their safety. Specifically, if a norm governing the police agent involves the case when he or she wishes to go to an area with many criminals and a few police agents, this norm could describe that it is necessary that the agent making the arrest calls for backup. In this way, if the agent complies with the norm, others police agents will go to the location and the arrest will be carried out successfully, and the agents will receive a reward that allows them to travel at a higher speed on the move to their targets. However, if the police agent violates the norm and puts his or her life at risk, their travel speed is reduced and they will lose some of their ability to make arrests.

3. Norms Definition

In this article, we adopted the definition of norms presented in [Lopez 2003] as: *Norm* (*addressees, deonticConcept, activation, deactivation, reward, punishments, elementRegulated*), where *addresses* refer to the set of agents that will be governed by the norms and *deonticConcept* is the deontic concept associated with the norm. *Activation* and *deactivation* are the activation and deactivation of the norm in the environment respectively. *Rewards* and *punishments* are the rewards and punishments attached to norm in the case it is fulfilled or violated, respectively. Finally, the entity governed by the norm is defined by the *elementRegulated* attribute.

To understand the definition of norms better, imagine that a firefighter receives the mission of rescuing civilians who are in geoenvironmental risk areas, and attempts to perform this rescue in accordance with the norms that were addressed. At this time the following norm is sent to the firefighter agents, "protect the lives of civilians at geoenvironmental risk areas" with the following attributes: the addressees are the firefighter agents, the required deontic concept is obligation, their reward when a norm is met is that the agent will get air or ground support in his mission, the punishment in case the norm is violated is that the firefighter agents will not get support in their rescue operation, the norm is activated if there is any person at risk, the norm is deactivated when all civilians are safe, and the element regulated by the norm is the action of using aircrafts because of the rescue costs.

4. Related Work

The n-BDI architecture [Criado et al. 2010] presents a model for designing agents capable of operating in environments governed by norms. This architecture considers that the selection of objectives should be performed based on the priority associated with each objective, where this priority is determined taking into account the priority of the norms that govern the objective. However, it is not clear in this approach how the components of a norm can be evaluated. In addition, the approach does not support a strategy to deal with the conflicts between norms.

In [Lopez and Marquez 2004], the authors propose a formal model, using the Z formal specification language, for modeling agents able to achieve their objectives

taking into account the norms of the system. According to [Lopez and Marquez 2004] an agent created from such a model is able to: (i) check if it is the one responsible for fulfilling a norm; (ii) verify the activation and deactivation of a norm taking into account the beliefs of the agent; (iii) evaluate and decide to fulfill or violate every norm of the system; and (iv) make the decision to fulfill or violate a norm, removing or adding agent goals. Besides not showing how the evaluation of a norm is performed, the authors do not focus on identifying and resolving conflicts between norms, checking fulfilled or violated norms, and showing the influence of norms on the plan selection process and intentions of the agents.

In [Lopez 2002], the author presents a set of strategies that can be adopted by agents to deal with norms. These strategies are: *Social*, *Pressured*, *Opportunistic*, and *Rebellious*. The *Social* strategy focuses on the agents complying with the rules without worrying about their individual goals. The *Pressured* strategy happens when agents fulfill the norms considering only the punishments will harm them to achieve their individual goals. Another strategy is the *Opportunistic* strategy, in which agents consider only the effects of rewards on their individual goals, and seek to fulfill only the norms for which the rewards of the individual goals are more important than those of the social goals. The *Rebellious* strategy implies that the agents will care only to achieve their individual goals, regardless of the punishments attached to the violation of the norms. Finally, the *Selfish* strategy is the combination of the *Pressured* and the *Opportunistic* strategies.

5. Normative Agent Java Simulation Framework - JSAN

JSAN provides the ability to create simulations that help to understand the impact of norms on agents capable of adopting different strategies to deal with the agent-related norms. Therefore, the framework enables the implementation of different normative strategies, collecting information of the simulated environment, and displaying information about the simulated scenarios that support the implementation of different movement strategies of the agents in the environment and supporting the generation of the norms and goals of the agents. The class diagram presented in Figure 1 shows the main classes and methods of the framework. Agents are represented by the class *NormativeAgent*, which is an extension of the original *Agent* class in the JASON framework [Bordini et al. 2007]. In addition, the simulation environment is described by the *EnvironmentSimulation* class, which extends the *Environment* class and provides support to create the simulation environment.

The *ExecuteAction* method is an extension of *Environment*, and much of the *Environment* code is written in it. Whenever an agent tries to perform an essential action, its identification and their chosen actions are passed to this method. For this reason, the code *ExecuteAction* method must verify that the action is valid and then do what is necessary for the action to be performed. The action may change the perceptions of agents. If this method returns true it means that the action was performed successfully. The *EnvironmentSimulation* class is also responsible for managing the creation of norms and individual goals of the agents, using instances of *GenerateNormsStrategy* and *GenerateGoalStrategy* classes, respectively. In addition, *EnvironmentSimulation* is an extension of the *MovementStrategy* class that is responsible for managing the strategies used by the agents to move in the environment as events occur during the simulation. Each norm in environment is an extension of the

Norm class (see the definition of norms provided in Section 3). Additionally, the framework provides a mechanism to report the impact of the norms on normative agents. To use this mechanism is necessary to extend the *ReportStrategy* class, passing the following parameters: (i) the environment in which the simulation is being carried out and (ii) an implementation of *NormStrategy* class, which contains the strategy used by the agent to handle the norms.

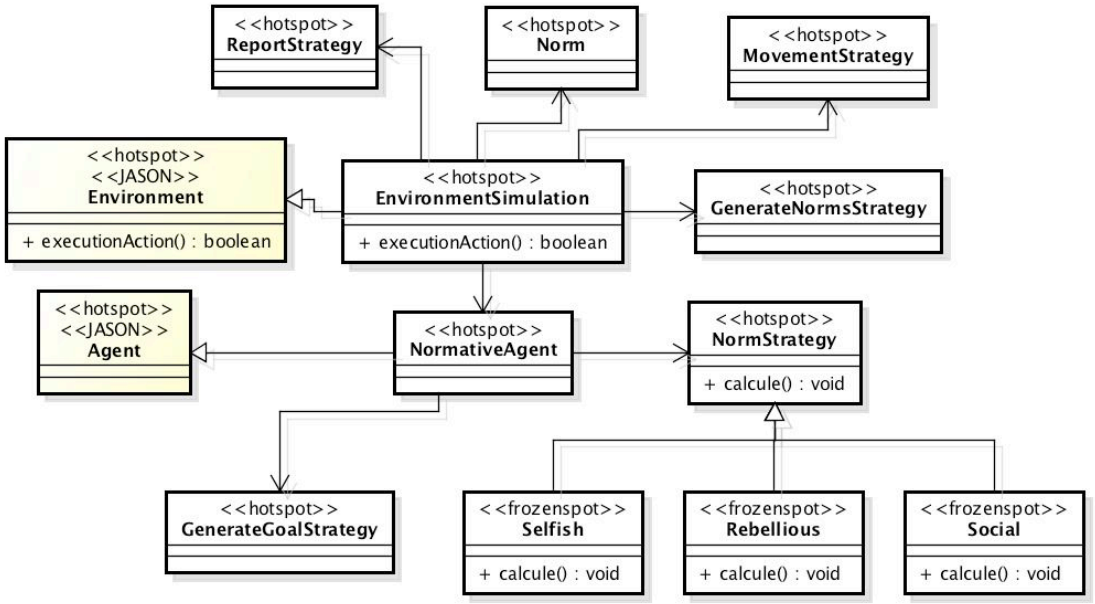


Figure 1 - Class Diagram

In short, to build a simulation of normative multi-agent systems using this framework it was necessary to extend some other classes. The *GenerateNormsStrategy* class needed to be extended to generate the norms that will be activated in the simulation environment. Further, the *GenerateGoalStrategy* class was extended so that we can generate the goals of the agents. By extending the *MovementStrategy* class, it is possible to generate different movement strategies for the agents in the simulated environment. In the *NormStrategy* class, a method called *calculate* was defined, which needed to be implemented to describe the strategy that will be used for agents dealing with the norms. We implement three types of strategies to deal with norms (*Selfish*, *Rebellious* and *Social*) (See definitions in Section 4). The *ReportStrategy* class was extended to report accurate information about each simulation.

7. CONCLUSION

This paper proposes a framework that allows the construction of simulations involving normative agents and provides the necessary mechanisms to understand the impact of norms on agents that adopt specific strategies to deal with these norms. Additionally, the framework provides many ways to represent computational normative concepts that can be used for better understanding norms related to the behavior and regulation of the agents. The proposed structure makes explicit what role the norm has in a society and the elements regulated by it, which in turn can be used by agents for decision-making in the society in which they live.

In order to evaluate the applicability of the framework, it will be applied in the evacuation of civilians from geoenvironmental risk areas [Cerqueira et al. 2009], and in crime prevention [Bosse and Gerritsen 2010]. For example, the insertion of norms in the simulation environment, which implements this framework, can be used to regulate the behavior of firemen agents and make them able to be norm-aware in the process of rescuing civilians from geoenvironmental risk areas.

As future work we plan to implement new mechanisms to deal with different levels of agent autonomy in order to show how different restriction levels and communities can influence the satisfaction of a norm application [Lopez 2002]. In the current version of the framework the autonomy-related restriction levels was not taken into account. However, the framework can be enhanced with different levels of restrictions, thus offering the possibility to achieve better results in terms of promoting a desirable social order.

References

Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007) "Programming multi-agent systems in AgentSpeak using Jason" (Vol. 8). John Wiley & Sons.

Bosse, T.; Gerritsen, C. (2010) "An Agent-Based framework to Support Crime Prevention", AAMAS, Toronto. 525-532.

Cerqueira, S. L. R., Santos Neto, B. F., Lucena, C. J. P., Campos, T. M. P., Moncada M. P. H. (2009) "*Plataforma GeoRisc Engenharia da Computação Aplicada à Análise de Riscos Geo-ambientais"*. PUC-RIO. Rio de Janeiro. Brasil.

Criado, N., Argente, E., Noriega, P., and Botti, V. (2010) "Towards a Normative BDI Architecture for Norm Compliance". COIN@ MAL- LOW2010.

Lopez, F. L.; Luck, M.; D'Inverno, M. (2002) "Constraining Autonomy through Norms", AAMAS.

Lopez, F. L. (2003) "Social Power and Norms" (Doctoral dissertation, University of Southampton).

Lopez, L. F. Marquez, A. A. (2004) "An Architeture for Autonomous Normative Agents", IEEE, Puebla, México.

Machado, R. Bordini, R. H. (2002) "Running AgentSpeak(L) agents on SIM AGENT", (ATAL-2001), August 1–3.

Santos Neto, B. F. D. Lucena, C. J. P. (2010) "JAFF: implementando agentes auto-adaptativos orientados a serviços", (PUC-Rio) Rio de Janeiro - Brasil.

Silva, V. T. (2008) "From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents". Autonomous Agents and Multi-Agent Systems, 17(1), 113-155.

Wooldridge, M. (2011). "Introduction to Multiagent Systems". John Wiley & Sons, Inc., New York, NY, USA. 1, 2.1, 2.7, 4.1, 4.1.1, 6.1.

Avaliação de arquitetura de Sistema Multiagente para efeito de aglomeração de nanopartículas

Alexandre de O. Zamberlan[1], Rafael H. Bordini[2], Solange B. Fagan[1]

[1]Programa de Pós-Graduação em Nanociências
Centro Universitário Franciscano

[2]FACIN-PUCRS

alexz@unifra.br, r.bordini@pucrs.br, solange.fagan@gmail.com

Abstract. *This article discusses about a proper Multiagent System architecture for the evaluation of nanoparticle agglomeration effect. Reactive or deliberative architectures of multiagent systems are employed in several simulations such as traffic control, rescue teams and evacuation procedures. Likewise, Nanotechnology, which projects, characterizes, produces and applies nano-scale structures, also uses simulations to nanostructures behavior assessments. Another important point, when investigating the universe on a very small scale, is the ability of matter self-organization, which is very similar to what happens with biological entities. In this case, the mechanisms respond to environmental stimuli without conscious control. Therefore, setting an appropriate architecture for this simulation is a key factor.*

Resumo. *Este artigo discute sobre a arquitetura de Sistema Multiagentes adequada para avaliação do efeito de aglomeração de nanopartículas. Arquiteturas reativas ou deliberativas de sistemas multiagentes são empregadas em diversas simulações, como por exemplo, controle de tráfego, equipes de salvamento, procedimentos de evacuação, entre outros. Da mesma forma, na Nanotecnologia, que projeta, caracteriza, produz e aplica estruturas em nano escala, também se utiliza de simulações para avaliações de comportamento de nanoestruturas. Um ponto importante, ao investigar o universo em uma escala muito pequena, é a capacidade de auto-organização da matéria, muito semelhante ao que acontece com entidades biológicas. Portanto, definir a arquitetura adequada para essa simulação é um fator fundamental.*

1. Introdução

Sistemas multiagentes (SMA) são empregados em investigações com diferentes propósitos, principalmente em simulações computacionais. Esses sistemas são compostos de entidades de software autônomas, denominadas agentes, que atuam e interagem em um ambiente compartilhado, alterando o estado do ambiente [Wooldridge 2001] e [Bordini et al. 2007]. A abordagem orientada a agentes permite um grau de abstração que não é usual em metodologias tradicionais de projeto e desenvolvimento de sistemas, tornando explícitas certas funcionalidades. E esse nível de abstração apresenta vantagem à medida em que as aplicações tornam-se maiores e mais complexas (como em simulações, com maior número de variáveis, restrições, operações, tratamentos de exceções, entre

outros), pois possibilita representar problemas e soluções de forma mais natural possível [Móra et al. 1998], com característica de projeto e desenvolvimento de sistema *bottom-up* ou *top-down*, ou seja, a partir do agente ou a partir da organização [Hübner et al. 2004].

O que caracteriza um agente e a sociedade que ele está inserido são as interações com o ambiente e os processos internos que possibilitam a realização dessas interações [Wooldridge 2001]. A especificação de quais e como são esses processos internos é chamada de arquitetura. Diferentes arquiteturas têm sido propostas com o objetivo de caracterizar os agentes (e a sociedade em que estão inseridos) com um nível de inteligência e de autonomia. Dessa forma, as arquiteturas podem ser classificadas, de acordo com o mecanismo utilizado pelo agente para a seleção das ações [Bordini et al. 2007].

Na modelagem e simulação molecular, na área da Dinâmica Molecular, modelar e simular referem-se a métodos para mimetizar o comportamento de moléculas ou de sistemas moleculares. Esses processos seguem leis e princípios da Física Clássica e da Física Quântica, levando em consideração campos de força e dependência do tempo [Vilela Neto 2014]. Ao lidar com estruturas na escala nanométrica, a principal característica que distingue uma nanopartícula é sua área superficial comparada com o seu volume [Bushan 2007]. E outro ponto importante, foco desta investigação, é a habilidade de auto-organização da matéria, quando em escala muito pequena, bastante similar com o que ocorre com entidades biológicas, que respondem ao ambiente sem um controle consciente [Bushan 2007]. Por isso, a simulação é uma maneira de avaliar comportamento desses elementos.

Sendo assim, surge o objetivo desta pesquisa, discutir qual a arquitetura de sistemas multiagentes adequada para este propósito de simulação. Para tanto, o texto está dividido em seções. A Seção 2 trata da Nanotecnologia e a modelagem molecular, bem como o que se espera dessa técnica. A Seção 3 discute sobre arquiteturas básicas de SMA. Finalmente, algumas considerações sobre este estudo e as referências bibliográficas.

2. Modelagem e simulação molecular

A Nanociência é o estudo de fenômenos e manipulações de materiais em escala atômica, molecular e macromolecular, em que as propriedades diferem significativamente dos em escala maior. Já a Nanotecnologia preocupa-se com o projeto, caracterização, produção e aplicação de estruturas em escala nano [Dowling et al. 2004].

Acredita-se que a Computação aplicada à Nanotecnologia (ou Nanotecnologia Computacional) tem seu foco no projeto e construção de ferramentas computacionais para auxiliar a compreensão de fenômenos físico e químicos que ocorrem em escala nanométrica aplicadas à nanoeletrônica, lógica e computação, sensores e novos materiais com características específicas. Os sistemas e a simulação computacional são realizados no tempo e no espaço utilizando modelos analíticos e fundamentos físicos, químicos e de ciência dos materiais [Vilela Neto 2014].

A modelagem molecular envolve métodos e técnicas computacionais para imitar o comportamento de sistemas atômicos e moleculares. Conforme [Vilela Neto 2014], assume-se que modelar é o processo de extrair informações realmente relevantes de determinado sistema, como aspectos estruturais (conjunto de atributos ou características e seus possíveis valores) e aspectos funcionais (operações ou métodos e suas restrições).

Os métodos computacionais da modelagem molecular estão associados à execução de cálculos de alta complexidade computacional. Esses métodos envolvem esforço computacional, com alocação extrema de recursos de *hardware* e *software*, por exemplo, a manipulação de muitas variáveis, a enorme variação de valores nessas variáveis, bem como suas inúmeras operações e restrições [Vilela Neto 2014].

A modelagem, de uma forma geral, é a criação de um modelo que possa representar um determinado domínio por meio de uma linguagem computacional (mais natural possível) que siga um ponto de vista. Portanto, um modelo deve ser a representação do conhecimento aplicado a uma ferramenta que promova o estudo do comportamento de sistemas complexos.

Há ferramentas de simulação consolidas na área de Dinâmica Molecular, como Gromacs (http://www.gromacs.org) e Siesta (http://departments.icmab.es/leem/siesta/), em que é possível modelar e acompanhar a simulação de milhares de estruturas como moléculas, átomos e elétrons; suas restrições como tempo e campos de força; suas interações inter e intra moleculares; e os eventos relacionados à pressão, temperatura, entre outros. Entretanto, o efeito de aglomeração, objetivo desta discussão, é melhor visualizado em experimentos em laboratórios (experimentos *in vitro*). Em simulações, há o desafio de monitorar, descrever ou, até mesmo, prever o comportamento de aglomeração em nanopartículas.

Neste estudo, assume-se que aglomeração é quando as partículas estão unidas frouxamente em conjunto e que pode ser simplesmente quebrado por forças mecânicas. Já a agregação é um padrão definido de moléculas que podem estar em qualquer estado físico [Nichols et al. 2002]. A aglomeração é um processo de contato e adesão em que partículas dispersas permanecem juntas por interações físicas fracas, sendo um processo reversível [McNaught and Wilkinson 1997]. Nanopartículas aglomeram ou agregam quando colocadas em ambiente ou em fluido biológico. Mas a aglomeração frequentemente ocorre quando há elevada força iônica, que protege a repulsão devido às cargas sobre as nanopartículas. Acompanhar o efeito de aglomeração é fundamental, por exemplo para avaliação de toxicidade das partículas em escala nano, uma vez que esse efeito altera o tamanho, a área de superfície e propriedades de sedimentação das nanopartículas [Schaffazick et al. 2003].

A caracterização de nanoestruturas envolve determinar propriedades específicas, tais como morfologia, tamanho de partícula, potencial zeta, pH, taxa de associação de cinética de libertação, determinação do conteúdo, entre outros. O índice de polidispersão fornece informações sobre a homogeneidade do tamanho da distribuição e o nível de aglomeração das partículas. E, como já mencionado, essas propriedades são verificadas em experimentos reais em laboratório, sendo algumas possíveis em simulação computacional [Schaffazick et al. 2003].

3. Arquiteturas de Sistemas Multiagentes

Os SMA formam uma área de investigação na Inteligência Artificial Distribuída (IAD), tratando os aspectos referentes à computação distribuída em sistemas de inteligência artificial. A definição da arquitetura interna do agente está associada com o tipo de tarefa que o agente irá realizar e qual o seu papel na sociedade multiagente. Uma vez considerado isto, o agente pode ser classificado como [Wooldridge 2001]: reativo, em que a esco-

lha da ação (resposta) está diretamente situada na ocorrência de um conjunto de eventos (estímulos) que ele percebe no e do ambiente, captados por seus sensores ou por mensagens enviadas por outros agentes; cognitivo ou deliberativo, pois possui um processo explícito para escolher a ação a ser realizada. Essa ação pode ser escolhida, também, através de uma função de utilidade e realizada por meio de um plano e uma representação simbólica do ambiente. Um agente cognitivo é um agente racional que possui alguma representação explícita de seu conhecimento e objetivos. Um agente pode ser mais cognitivo do que outro, conforme o grau de racionalidade explícita de seu comportamento [Hübner et al. 2004] e [Wooldridge 2001].

Arquiteturas de agentes cognitivos, segundo [Oliveira 1996], podem ser classificadas em arquiteturas funcionais, em que o agente é composto por módulos que representam cada uma das funcionalidades necessárias para sua operação. O agente possui conhecimento, um conjunto de objetivos, capacidade de percepção, comunicação, decisão e raciocínio. Além dessa, há as arquiteturas baseadas em estados mentais, que adotam uma perspectiva de inspiração psicológica para definir a estrutura dos agentes, que são entidades cujo estado é constituído de componentes mentais tais como crenças, desejos, capacidades, escolhas e compromissos. Pode ocorrer que uma arquitetura baseada em estados mentais também contemple aspectos funcionais, e vice-versa, não sendo excludentes, e podendo ser complementares [Bordini et al. 2007]. As abordagens utilizadas para a modelagem de agentes são as mais diversas, e uma freqüentemente utilizada é a abordagem que considera agentes como sendo sistemas intencionais. Sistemas intencionais são aqueles a que são atribuídas atitudes intencionais (estados mentais) [Móra 2000].

De acordo em [Zamberlan 2002], as definições e propriedades que caracterizam a noção de agente têm por objetivo não meramente dividir o mundo entre entidades que são e que não são agentes, mas servirem de ferramentas para analisar sistemas, bem como especificar, projetar e implementar sistemas cujos elementos básicos sejam agentes.

4. Considerações Finais

Os ambientes de simulação de nanoestruturas podem ser considerados essencialmente reativos, ou seja, ideais para arquiteturas reativas de agentes. Muitos pesquisadores tenderiam à indicação de utilização de técnicas de Resolução Distribuída de Problemas, ao invés de Sistemas Multiagentes. Mas, com o uso de SMA é possível projetar sistemas complexos de simulação, por exemplo, de maneira naturalmente distribuída e *bottom-up*.

O paradigma de SMA baseia-se em sistemas naturais, nos quais se percebe o surgimento de um comportamento inteligente a partir da interação de seus elementos, como ocorre em um formigueiro (a colônia possui um comportamento inteligente, enquanto que a formiga não) e nos neurônios (células simples, mas sua interação e organização emerge um comportamento complexo e inteligente) [Hübner et al. 2004]. A coletividade possui características que não podem ser reduzidas aos seus elementos formadores. Neste ponto, não pretende-se afirmar que nanopartículas possam vir a ter um comportamento inteligente, mas sim complexo, uma vez que interagem, há um esquema de organização e um ambiente bastante reativo. Uma característica bastante significativa na teoria orientada a agentes é a autonomia, também existente em estruturas em escala reduzida, como átomos e moléculas, apesar da forte interação existente. Em relação à organização de um sistema multiagente reativo ou cognitivo, há eventos, restrições e interações, o mesmo que ocorre

num ambiente na escala nano.

Das duas abordagens para desenvolvimento de SMA, a abordagem *botom-up* que parte dos aspectos individuais do agente de maneira que surjam os aspectos coletivos é bastante indicada. Isso porque, os agentes são projetados independente de algum problema específico; a interação inter-agentes não é construída previamente, bastando projetar um protocolo de comunicação para situações genéricas e porque não existe um controle centralizado. Dessa forma, há vantagens como a viabilização de sistemas adaptativos e evolutivos, em que o SMA tem habilidade de adaptação de novas situações (inclusão ou exclusão de agentes ou mudanças na organização); como já mencionado, é naturalmente adequado para a modelagem de sistemas complexos e distribuídos. Assim sendo, adequado para simulações em ambientes nanoestruturados.

A área de SMA encontra-se consolidada, com metodologias e ferramentas para projeção e implementação de sistemas multiagentes, tanto reativos quanto cognitivos. Entretanto, mesmo a arquitetura reativa sendo a que melhor se enquadre no contexto desta discussão, é possível projetar e implementar a simulação de nanopartículas a partir da teoria intencional de agentes baseada em estados mentais. Uma justificativa de se utilizar arquitetura cognitiva baseada em estados mentais, segundo [Móra 2000], é que essa arquitetura baseia-se na abordagem intencional, pois possuem um forte apelo conceitual, como abstração, pois são bastante naturais para os projetistas e analistas que utilizam a abordagem orientada a agentes; fornecem descrições sucintas para sistemas complexos, além de ajudar o projetista a entender e a explicar o comportamento desses sistemas; podem ser usados pelos agentes para raciocinar sobre eles próprios e sobre outros agentes (reciprocidade, segundo nomenclatura adotada [Dennett 1987]). Outra justificativa para essa escolha, é que em *benchmark* realizados, é possível observar ferramentas, no contexto de SMA baseados em estados mentais, com bom desempenho. Por exemplo, no trabalho realizado por [Cardoso et al. 2013], procurou-se avaliar dois modelos para programação concorrente com agentes e com atores, visto que os sistemas distribuídos têm crescido com os recursos computacionais de processadores de multi núcleos. Sendo assim, um contexto adequado para aplicação de simulações de nanoestruturas.

Ressalta-se que esta pesquisa está inserida em um trabalho mais amplo, de avaliar os aspectos de aglomeração das nanopartículas por meio de sistema multiagente. Assim sendo, são necessárias algumas etapas: compreender os processos de produção e caracterização de nanopartículas; identificar e testar metodologias de análise estrutural e funcional de nanoestruturas através de ferramentas de simulação; identificar a arquitetura de sistema multiagente adequada para este contexto; projetar e implementar o ambiente de simulação baseado na teoria Multiagente. Dessa forma, neste artigo buscou-se discutir as áreas de modelagem e simulação molecular e sistemas multiagentes. As próximas ações nesta pesquisa são: i) definir a relação entre partículas e a relação das partículas com ambiente, assumindo que cada partícula será um agente no sistema, detalhando assim como se dará o uso de agentes no cenário de simulação de nanoestruturas; e ii) testar algumas métricas utilizadas em [Cardoso et al. 2013] para avaliar arquiteturas reativas e arquiteturas cognitivas implementadas em Jason.

Referências

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. Wiley.

Bushan, B. (2007). *Handbook of nanotechnology.* Springer.

Cardoso, R. C., Zatelli, M. R., Hübner, J. F., and Bordini, R. H. (2013). Towards benchmarking actor- and agent-based programming languages. In *Proceedings of the 2013 Workshop on Programming Based on Actors, Agents, and Decentralized Control*, AGERE! 2013, pages 115–126, New York, NY, USA. ACM.

Dennett, D. (1987). *The intentional stance*. MIT Press, Cambridge, MA.

Dowling, A., Clif, R., and Grobert, N. (2004). Nanoscience and nanotechnologies: opportunities and uncertainties. Technical report, The Royal Academic of Engineering.

Hübner, J. F., Bordini, R. H., and Vieira, R. (2004). Introdução ao desenvolvimento de sistemas multiagentes com Jason. In *XII Escola de Informática da SBC*, pages 51–89, Paran·. Fernando Takashi Itakura et. al.

McNaught, A. D. and Wilkinson, A. (1997). *Compendium of Chemical Terminology (The Gold Book)*. Blackwell Scientific Publications, Oxford, http://goldbook.iupac.org/.

Móra, M. (2000). *Um modelo de agente executável*. PhD thesis, Universidade Federal do Rio Grande do Sul (UFRGS), Brasil, Porto Alegre.

Móra, M., Lopes, J., Coelho, J., and Viccari, R. (1998). BDI models and systems: reducing the gap. In *Agents Theory, Architecture and Languages Workshop*, Canarias. Springer-Verlag.

Nichols, G., Byard, S., Bloxham, M. J., Botterill, J., Dawson, N. J., Dennis, A., Diart, V., North, N. C., and Sherwood, J. D. (2002). A review of the terms agglomerate and aggregate with a recommendation for nomenclature used in powder and particle characterization. *J. Pharm. Sci.*, 91(10):2103–2109.

Oliveira, F. (1996). Inteligência artificial distribuída. In *IV Escola regional de informática*, Canoas, RS. Sociedade Brasileira de Computação.

Schaffazick, S. R., Guterres, S. S., de Lucca Freitas, L., and Pohlmann, A. R. (2003). Physicochemical characterization and stability of the polymeric nanoparticle systems for drug administration. *Quim. Nova*, 26(5):726–737.

Vilela Neto, O. P. (2014). Intelligent computational nanotechnology: The role of computational intelligence in the development of nanoscience and nanotechnology. *Journal of Computational and Theoretical Nanoscience*, 11:1–17.

Wooldridge, M. (2001). *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA.

Zamberlan, A. O. (2002). Em direção a uma técnica para programação orientada a agentes BDI. Master's thesis, PUCRS.

The Evaluation of The Use of a MultiAgent System Coordination Mechanism to Support Emergency Operations *

Emmanuel D. Katende[1]

[1]Programa de Pós-graduação em Engenharia de Automação e Sistemas (PPGEAS)
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

emmanuel.katende@posgrad.ufsc.br

Abstract. *This short-paper presents an initial overview of a research work that deals with emergency operations in urban cities using MultiAgent System Coordination mechanism. We presented first the emergency scenario in the state of Santa Catarina in Brazil, citing some main challenges encountered in such cases, and thus showing how it is important to find solutions to cope with emergencies. Following, we analysed some works in the literature that present some solution for emergency cases, and show some issues in such works that need more studies. Lastly, we list the likely steps of the solution method to be implemented in future works, as a potential solution that can manage emergency operations considering the observations presented in the state of the art.*

1. Introduction

Urban areas are becoming increasingly dense and interconnected which makes them vulnerable to some types of emergencies such as vehicles accidents, building fires, gas leaks, and unexpected health crisis situations [Monares et al. 2012], requiring fast and effective responses from emergency teams such as: firefighters, medical personnel, and police officers [Borges et al. 2010]. The Brazilian Mobile Emergency Care (SAMU) that is one of emergency responder teams, registered 832,965 emergency calls from January to November 2014 in the State of Santa Catarina, from which 149,934 (18%) needed ambulance intervention. From cases registered as death, 70% occurred before the team arrives, 27% during the care, and 3% during the transportation to an healthcare point [SAM].

There are various challenges to overcome in emergency operations from which we can mention at least four: the difficulty of emergency responders to reach either the incident local (I), or healthcare points (II), due to the environment chaos, traffic congestions, as well as uncertainties and uncontrollable events that can occur. Such challenges may be seen as a classic traffic control problem, but adding to this the presence of an hostile environment and uncertainties. The cooperation between several services involved into the incident and the coordination of interdependencies between their activities in order to increase the efficiency of the rescue operation may be the main challenge in disaster management (III). Such challenges involve emergency teams interacting according to a cooperation and coordination mechanism with the aim of rescuing people and recovering infrastructures. The last challenge we may cite is to improve the security of the area and people around (IV).

*This paper is partially financed by Coordenação de Aperfeiçoamento do Pessoal de Nível Superior (Capes)

Regarding challenges I and II that may refer basically to traffic control, a considerable number of advanced solutions have been proposed in order to cope with traffic control problems such as: optimization, control and automation. Nevertheless, such approaches do not seem to be sufficient to solve the presented problem, since most of researches in classical traffic control generally focus on the task of optimizing the traffic in congestion or similar situations, due to rush hours, some event, or even traffic accidents or another incident that may be considered as an emergency case, whereas the traffic control involved during a disaster management is affected by unfamiliar and hazardous events, therefore require various services working together, that is, a kind of cooperation between such services and coordination of their activities in order to reach the overall goal established.

This work aims to present an overview of the likely steps of a method that evaluate the use of a MultyAgent System (MAS), and its cooperation and coordination mechanisms, as an instrument that manage inter-dependencies between urban services activities in order to control the traffic and improve the efficiency of emergency operations in urban cities, as well as minimize the traffic chaos that such management policies could cause. To reach such aim, we need first to create an emergency scenario as well as define and implement a MAS, endowed with a cooperation and coordination mechanism, that can achieve some objectives which consist in overcoming some of the four emergency operation challenges described in the section 1, and listed below:

1. Control the traffic in order to facilitate the displacement of emergency responders to the local of incident;
2. Control the traffic in order to facilitate the displacement of victims to healthcare points;
3. Promote and improve the emergency responders cooperation in order to take care of a maximum number of life;
4. Assess and adjust, if necessary, the traffic management policies in order to minimize the traffic chaos that such policies could cause.

2. State of The Art

The scenario being studied in this work: *Traffic Management Under Emergency Situations*, is part of studies in at least two mature topics in the literature which are: *Traffic Management* and *Emergency Responses*, which are usually studied separately. Numerous researches have been realized in both fields among which we can cite: [Papageorgiou et al. 2003], [Wu 2002], [Jin et al. 2007], [He et al. 2007], and [McKenney and White 2013] for traffic Management. Solutions proposed in these works derived from areas such as: Adaptive Control, Optimization, MultiAgent System, and Artificial Intelligence. Most of these researches focus more on the congestion problem, traffic flow optimization and similar situations. Regarding emergency responses, we can cite as examples: [Abramson et al. 2008], [Adams et al. 2008], [Stranders et al. 2010], citefilippoupolitis:etal:2008b, and [Hawe et al. 2012], where most of solutions proposed derived from AI and MAS.

Even though the traffic management under emergency situations topic have solutions from several areas, we focus our analysis on solution that use Agent and MAS theory as means, since our aim is to evaluate the use of MAS coordination mechanism in order to cope with such situations. However we also comment quickly on some works

cited at the first paragraph that propose some solutions in different areas than MAS. We used Google Scholar, IEEE Explorer, and Mendeley as reference managers and academic social networks in order to find such works, searching by key words such as: -*Traffic management*, *Emergency*, *Coordination*. Finally, we summarized the results in Table 1, separated by *title* and *reference*, *specific scenario* and *solution field* that is the domain from where the solution method or feature used in the publication derived.

Title	**Scenario**	**Solution Field**
Model Reference Adaptive Control Framework for Real-Time Traffic Management under Emergency Evacuation [Liu et al. 2007]	Traffic control under emergency evacuations	Adaptive Control Theory
Traffic organization method for emergency evacuation based on information centrality [Wu et al. 2010]	Traffic control under emergency evacuations	Optimization
(Notice of Retraction) A Decision Support System of Urban Traffic Emergency Control Based on Expert System [Shumin et al. 2010]	Traffic control under emergency operations	AI (Expert Systems)
Rapid Handling of Urban Traffic Emergencies Based on Decision and Command Support System [Lu 2009]	Handling urban traffic emergency	Software engineering
A heuristic implementation of emergency traffic evacuation in urban areas [Kang et al. 2013]	Traffic control under emergency situations	Algorithms and Graphs

Table 1. Works in Traffic Management under Emergency Operations

Analysing some of Table 1 works, we can notice that [Liu et al. 2007], by example, integrate both dynamic network modelling techniques and adaptive control theory to manage the traffic at real-time during emergency evacuations. They consider that the traffic flow during emergency is full of uncertainties, therefore needs a dynamic modelling technique as well as a real time adaptive control. Although such approach shows interesting results regarding optimization of traffic evacuation, it does not include explicitly the emergency responders management into the scenario, in the sense of several services cooperating according to a coordination mechanism in order to reach a same overall goal. Which is one of important problems to consider to cope with emergency situations. Another considerable issue in the solution proposed by [Liu et al. 2007] is that the system objectives are almost handwritten by the designer, which do not match totally with the real scenario in which team responder objectives can change at real time depending on events occurring. The same two main problems observed in [Liu et al. 2007]'s approach are encountered in [Wu et al. 2010], and [Kang et al. 2013].

[Shumin et al. 2010], by example, proposed a solution that establish a decision support system of urban traffic control for emergency based on expert system. They consider that there can have a expert system that have enough knowledge of the overall problem, therefore, can be used to solve it. Such centralized problem solving approach in which one assumes that a component or agent must have the appropriate expertise to

decompose the overall problem to specific ones, thus, must have knowledge of the task structure [Wooldridge 2009], is less encouraged to cope with complex problems. Since it is less sure to find a component that has a global view of the environment as well as events occurring. The same main problem observed in [Shumin et al. 2010]'s approach is also encountered in [Lu 2009], and [Kang et al. 2013].

[Rodríguez et al. 2009] proposed a solution using MAS. They focused on recalculation of route in real time to reduce the travel time between two points, using several emergency agents collaborating together into a hierarchical agents architecture. Still, one issue of such approach is that part of agents are not autonomous since they are used to pass the information observed to global agents and wait for their decisions. This cause a considerable quantity of explicit communications between agents that could be avoided if each agent was able to decide it self based on what it observed and its belief base. To illustrate such quantity, just consider a quadrant Q_1 in which IP_{1x}, an Information Panel got a message m_1 from a given local agent LA_1, and output $m1$. GA_1, the Q_1 Global Agent uses m_1 together with other messages in order to decide what to do. After deciding, GA_1 informs to the concerned Regional Agent R_{1x} the decision, and R_{1x} does the same recursively until to reach the concerned local agent(s) LA_{1Y}, etc. Let consider T the total need time for D_1 be done, t the unit explicit communication time. h, the agent hierarchical tree heigh. The value of T is given by: $T = t*h$;

Let suppose now that D_1 was a message that had to be output to another Information Panel IP_{ix}. This means that GA_{ix} would have to wait T time to evaluate and decide what to do, and such decision would take T' time to be done. So supposing that $T = T'$ and such event occurred n times, we have: $X = nT = n*(t*h) = t*(n*h)$; where X is the global time. So h could be avoided if each agent was able to decide itself based on what it perceives and knows. Such approach can be possible by using norms, for example. That means, when any agent perceive something that can be useful for farther decision, it can put it as a new norm so that others agents can perceive and act based on such norm. This lead us to one of mains coordination mechanism called: *coordination through norms and social laws*, presented by [Wooldridge 2009], that we intend to use farther as coordination mechanism.

3. Steps to Reach the aim

This section presents an initial method to reach the defined aim, addressing the observations highlighted at the state of the art. So it is fundamental to get more knowledge on MAS traffic control strategies as well as emergency operations management in order to combine and adjust both area methods. Furthermore, it also seems important to make some interviews with emergency responders about how their activities occur, so that we can find a solution that matches with the real world. This work will be developed in following steps:

1. Construct a urban traffic network using AIMSUN, an advanced traffic simulator;
2. Create an emergency incident scenario at one point of the network from step 1, that better represent the challenges defined in section 1, hence, the need of reaching all the specific objectives enumerated in section 1.
3. Define metrics that can better assess the MAS to be implemented, as well as its cooperation and coordination mechanisms, according to the specific objectives defined in section 1.

4. Implement a MAS as well as its cooperation and coordination mechanisms in order to reach the overall goal, achieving all the specific objectives, and meeting all the observations highlighted in the section 2.
    (a) Define an ontology that can better represent the scenario from step 2.
    (b) Define a protocol for the overall problem decomposition and task allocation;
    (c) Define a method that will synthesize the overall solution from sub problem results;
    (d) Define a adequate language for communication between agents;
    (e) Implement the MAS: agents, environment, and organization;
    (f) Implement a coordination mechanism that will manage inter-activities between the agents.
5. Apply the MAS from step 4 to the scenario from step 2, and evaluate the results based on the metrics from step 3.

4. Final Considerations

This short-paper just presents an initial overview of a research work that deals with emergency cases in urban cities. We presented first the emergency scenario and some main challenges encountered in emergency operations. We chose MAS as the technique to manage emergency operations because we believe and consider as hypotheses that MAS can better represent such scenario and also inherently considers cooperation and coordination mechanisms that are powerful instruments to cope with problems such as emergencies. After searching and analysing some works in the literature that also present solutions to cope with emergencies, we highlighted some observations that should need more studies, and lastly, we list the likely steps of the solution method to be implemented in future works, considering all the observations highlighted in the state of the art.

References

Associação paulista para o desenvolvimento da medicina -spdm, serviço de atendimento móvel de urgência (samu). `http://samu.saude.sc.gov.br/index.php/estatisticas`. accessed at 22/12/2014.

Abramson, M., Chao, W., Macker, J., and Mittu, R. (2008). Coordination in Disaster Management and Response : a Unified Approach. In *Massively Multi-Agent Technology*, pages 162–175.

Adams, N., Field, M., Gelenbe, E., Hand, D., Jennings, N., Leslie, D., Nicholson, D., Ramchurn, S., and Rogers, A. (2008). Intelligent Agents for Disaster Management. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 3 AAMAS 02*, page 1405.

Borges, M. R. S., Ochoa, S. F., Pino, J. A., and Vivacqua, A. (2010). Assigning emergency vehicles to urban incidents. In *Frontiers in Artificial Intelligence and Applications*, volume 212, pages 498–509.

Hawe, G. I., Coates, G., Wilson, D. T., and Crouch, R. S. (2012). Agent-based simulation for large-scale emergency response. *ACM Computing Surveys*, 45:1–51.

He, J., Bresler, M., Chiang, M., and Rexford, J. (2007). Towards robust multi-layer traffic engineering: Optimization of congestion control and routing. *IEEE Journal on Selected Areas in Communications*, 25:868–880.

Jin, X., Itmi, M., and Abdulrab, H. (2007). A Cooperative Multi-agent System Simulation Model for Urban Traffic Intelligent Control. In *Proceedings of the 2007 Summer Computer Simulation Conference*, pages 953–958.

Kang, W., Zhu, F., Lv, Y., Xiong, G., Xie, L., and Xi, B. (2013). A heuristic implementation of emergency traffic evacuation in urban areas. In *Proceedings of 2013 IEEE International Conference on Service Operations and Logistics, and Informatics*, pages 40–44. IEEE.

Liu, H. X., Ban, J. X., Ma, W., and Mirchandani, P. B. (2007). Model Reference Adaptive Control Framework for Real-Time Traffic Management under Emergency Evacuation.

Lu, F. (2009). Rapid Handling of Urban Traffic Emergencies Based on Decision and Command Support System. *2009 Second International Conference on Intelligent Computation Technology and Automation*, pages 677–680.

McKenney, D. and White, T. (2013). Distributed and adaptive traffic signal control within a realistic traffic simulation. *Engineering Applications of Artificial Intelligence*, 26:574–583.

Monares, A., Ochoa, S. F., Pino, J. A., and Herskovic, V. (2012). Improving the initial response process in urban emergencies. In *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2012*, pages 379–386.

Papageorgiou, M., Diakaki, C., Dinopoulou, V., Kotsialos, A., and Wang, Y. W. Y. (2003). Review of road traffic control strategies. *Proceedings of the IEEE*, 91.

Rodríguez, A., Eccius, M., Mencke, M., Fernández, J., Jiménez, E., Gómez, J. M., Alor-Hernandez, G., Posada-Gomez, R., and Cortes-Robles, G. (2009). A multi-agent system for traffic control for emergencies by quadrants. In *Proceedings of the 2009 4th International Conference on Internet and Web Applications and Services, ICIW 2009*, pages 247–253.

Shumin, S., Zhaosheng, Y., and Maolei, Z. (2010). A decision support system of urban traffic emergency control based on expert system. In *2010 IEEE International Conference on Software Engineering and Service Sciences*, pages 221–225.

Stranders, R., Delle Fave, F. M., Rogers, A., and Jennings, N. R. (2010). A decentralised coordination algorithm for mobile sensors. In *Proceedings of the 24th Conference on AI (AAAI)*, pages 874–880.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.

Wu, L. G. (2002). *Urban traffic control system using multi-agent*. PhD thesis.

Wu, Z., Jiang, G., Zhang, C., and Tang, Y. Y. (2010). Traffic organization method for emergency evacuation based on information centrality. In *Proceedings - 2nd IEEE International Conference on Advanced Computer Control, ICACC 2010*, volume 3, pages 92–96.

Definição de Personalidade em Agentes Baseada em Rede Bayesiana de Emoções

Gustavo Carneiro Fleck[1], Andressa da Cruz Freitas[1], Adriano Wherli[1], Diana F. Adamatti[1]

[1]Laboratório de Simulação Social e Ambiental – Centro de Ciências Computacionais Universidade Federal do Rio Grande (LAMSA/C3/FURG) – Rio Grande – RS - Brasil

gustavofleck@furg.br, freitas.adc@gmail.com, dianaada@gmail.com, werhli@gmail.com

Abstract*: This paper presents a case study where personalities agents are defined, based on the Bayesian networks emotions model. The results show that, depending on the personalities specified, the acting of the agent is affected, nearly as in real life.*

Resumo*: Este artigo apresenta um estudo de caso onde personalidades são definidas em agentes, baseadas no modelo de redes Bayesianas de emoções. Os resultados demonstram que, dependendo das personalidades definidas, o desempenho do agente é afetado, aproximadamente como na vida real.*

Keywords: Sistemas Multiagentes, Rede Bayesiana de Emoções, Modelo OCC.

1. Introdução

A emoção humana é alvo de inúmeros estudos em diversas áreas científicas, sendo o meio computacional uma delas. A Inteligência Artificial (IA) é uma área multidisciplinar que visa possibilitar a simulação da capacidade humana de pensar, tomar decisões, sentir e resolver problemas. Dentro da IA temos os Sistemas Multiagentes, que oferecem a possibilidade de simular várias dessas situações através da interação, de agentes e o meio, resultando em uma representação, aproximada, do comportamento humano.

Para a obtenção de resultados ainda mais aproximados da mente humana, percebeu-se a importância do uso de Redes Bayesianas. As Redes Bayesianas são uma ótima ferramenta para simular situações comuns ao dia-a-dia, pois sua estrutura utiliza um raciocínio probabilístico que nos permite colocar imprevisibilidade nos agentes.

Tratando-se de emoções, existem alguns modelos teóricos que tentam formalizar seu funcionamento, entre eles se destaca o modelo OCC (Ortony et al., 1988). O modelo é composto por 22 emoções, aceita três formas de estímulos, eventos do ambiente, ações de agentes e objetos.

Este artigo tem como objetivo mostrar como são gerados agentes com diferentes personalidades, através da criação de uma Rede Bayesiana baseada no modelo OCC de

emoções, a fim de simular um comportamento próximo ao da mente humana no meio computacional e mostrar a diferença de desempenho em cada personalidade testada.

O artigo está estruturado da seguinte maneira: Seção 1 Introdução, Seção 2 Referencial Teórico, Seção 3 Estudos de Caso, Seção 4 Resultados e Seção 5 Conclusões.

2. Referencial Teórico

Este trabalho envolve o estudo de emoções, Redes Bayesianas e sistemas multiagentes. As emoções são algo ainda incompreendido, de certa forma, na ciência, o que torna sua simulação algo muito mais complicado. A mente humana é imprevisível e é impossível prever um padrão para as emoções, já que todo o ser-humano age de forma diferente ao receber os mesmos estímulos.

O modelo OCC de emoções foi proposto no livro "The Cognitive Structure of Emotions" (Ortony et al., 1988) por Ortony, Clore e Collins. Esse modelo é capaz de identificar a partir de estímulos criados em um ambiente, as emoções que serão geradas.

Ele usa três tipos de geradores de estímulos: eventos, agentes e objetos. Toda emoção gerada é resultado de um ou mais estímulos. O modelo é composto por 22 emoções, onze positivas e onze negativas, e se baseia na diferenciação das reações de valências positivas e negativas, ou seja, a partir de um evento, variáveis são atribuídas a fim de gerar uma emoção positiva ou negativa.

As redes Bayesianas foram desenvolvidas no início dos anos 1980 para facilitar a tarefa de predição e "abdução" em sistemas de inteligência artificial (Pearl, 1988). Em resumo, são modelos gráficos para raciocínio baseado na incerteza. Podemos defini-las como sendo uma combinação da Teoria Probabilística e a Teoria dos Grafos.

De mesma forma como nas organizações humanas, as atividades, muitas vezes, são realizadas por um grupo de pessoas que trabalham de modo cooperativo, onde existem decisões individuais que afetam o grupo. Em Sistemas Multiagentes (SMA) as pessoas são representadas por agentes artificiais, os quais se relacionam em um ambiente de forma a buscar soluções para problemas de forma cooperativa, compartilhando informações, evitando conflitos e coordenando a execução de atividades (Adamatti, 2003).

2.1. Modelo de Rede Bayesiana de Emoções

Baseado no modelo OCC, Neves (2014) desenvolveu um modelo de rede Bayesiana de emoções, levando em consideração o fato de possuir uma estrutura de simples tradução computacional e por ser um modelo bastante abrangente, com isso adicionando a imprevisibilidade necessária nas emoções.

A rede Bayesiana de emoções de Neves (2014) é semelhante a rede proposta por Conati et al. (2010), porém o modelo de Neves substitui o Modelo OCC por uma rede Bayesiana, enquanto a rede de Conati processa as informações do ambiente e cria as entradas para o Modelo OCC. A rede de Neves possui 34 nós e 49 arestas e pode ser dividida em três áreas, eventos do ambiente, ações de agentes e objetos, assim como no Modelo OCC.

O modelo proposto por Neves (2014) é apresentado na Figura 1.

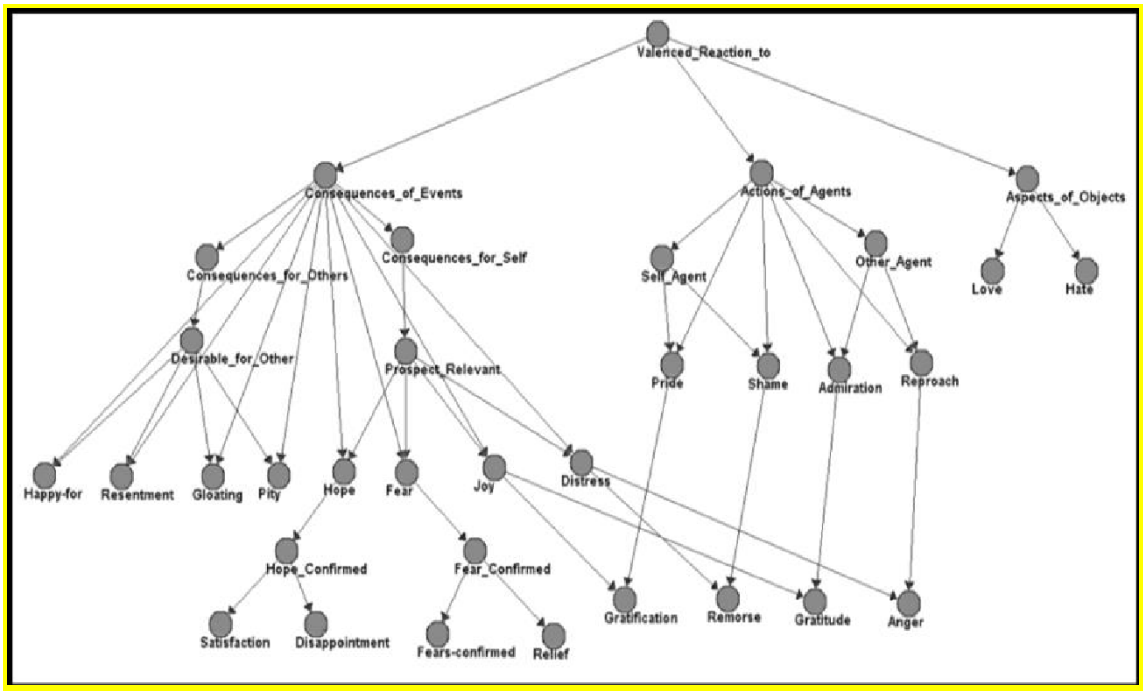


Figura 1. Rede Bayesiana de emoções (Neves, 2014).

Para modelagem da rede Bayesiana utilizada neste trabalho, fez-se uso do sistema JavaBayes (Cozman, 2001). O software permite a construção, visualização gráfica e análise de redes Bayesianas. O software possibilita a criação de redes com qualquer estrutura, com um número indefinido de nós, arestas e variáveis.

3. Estudo de Caso

Após criação da rede Bayesiana e da definição das probabilidades, é necessário adicionar a rede Bayesiana de emoções a um ambiente multiagentes de forma a validá-la e propiciar a avaliação de sua eficácia. Para isso foi escolhido o Jason, um software desenvolvido na linguagem Java, como ambiente multiagentes.

Em sua base de dados padrão, Jason oferece uma série de exemplos de modelos multiagentes que simulam diversas situações. Dentre elas se encontra o chamado *cleaning_robots,* como mostra a Figura 2, o qual foi utilizado como base para o estudo da funcionalidade e efetividade da rede Bayesiana de emoções em um ambiente multiagentes.

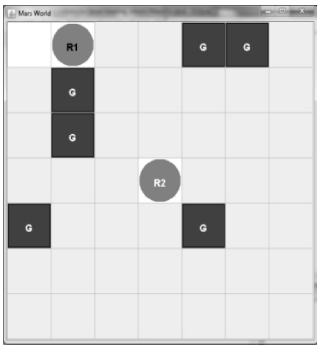


Figura 2. Interface do Cleaning Robots (Neves, 2014).

Neste exemplo, dois robôs R1 e R2 coletam e eliminam lixo no planeta Marte. O robô R1 anda sobre o solo do planeta procurando unidades de lixo. Ao encontrar uma unidade, o agente recolhe e leva até o ponto onde está R2, em seguida retornando ao ponto onde encontrou a unidade para continuar a busca. O robô R2, por sua vez, está posicionado junto a um incinerador e ao receber uma unidade de lixo imediatamente a queima. Neste trabalho, apenas o agente R1 trabalha sobre o efeito de emoções.

Quando o agente R1 não está sobre o efeito de alguma emoção ele demora dois ciclos para seguir para o próximo espaço; quando o estimulo gera uma emoção ruim, aumentam de dois ciclos para três; e quando é gerada uma boa emoção, diminui de dois para apenas um ciclo.

4. Resultados

Para definir as diferentes personalidades dos agentes, foram feitas mudanças nas probabilidades das emoções base utilizadas na rede Bayesiana proposta por Neves (2014). As emoções consideradas boas tiveram suas probabilidades de ocorrer aumentadas, dentro do código fonte da rede Bayesiana, de 95% para 99%, para a personalidade **otimista**, e as emoções consideradas ruins foram aumentadas de 95% para 99%, para a personalidade **pessimista**. A personalidade **neutra** não sofreu nenhuma alteração, pois a probabilidade de emoções positivas deve ser a mesma de emoções negativas. Com essas pequenas mudanças pode se observar uma real diferença entre as personalidades tendo a personalidade neutra como base, quando o agente é definido como pessimista, ele se mostra menos eficaz no desempenho de sua tarefa, o contrário acontece ao atribuirmos a personalidade otimista, como podemos observar nas figuras 3 e 4.

Os resultados das simulações foram armazenados em uma planilha eletrônica e importados para o software R, para que gráficos fossem criados, de forma a melhor visualizar as diferenças entre as personalidades (neutra, otimista e pessimista), resultados que foram obtidos a partir de uma média de execução.

A Figura 3 apresenta a diferença entre as personalidades por ciclos de execução. Onde o agente neutro demora 77 ciclos, o otimista demora 55 ciclos e o pessimista demora 105 ciclos para terminar a execução.

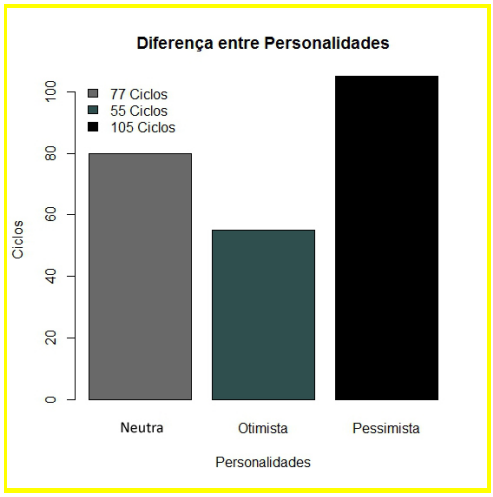


Figura 3. Ciclos por personalidades.

Já a Figura 4, apresenta o somatório de emoções ruins e boas por personalidade. Verifica-se que o agente neutro tem como soma total de suas emoções 0,587250, enquanto as outras duas personalidades, otimista e pessimista, tem como soma total 1,166836 e 0,395615 respectivamente. Lembrando que quanto maior o valor do somatório total das emoções, mais emoções boas prevaleceram durante a execução.

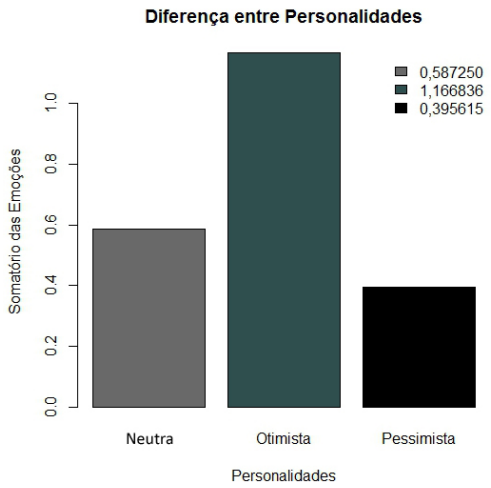


Figura 4. Somatório das emoções por personalidade.

5. Conclusões

Através da modelagem de emoções em sistemas multiagentes conseguimos simular computacionalmente situações comuns na vida real. As emoções expressas pelos agentes buscam cada vez mais uma proximidade como as dos humanos. Através da utilização de um modelo de emoções é possível criar algoritmos para tomada de decisão de agentes artificiais.

A Rede Bayesiana foi aplicada com sucesso nos exemplos do *cleaning robots,* e se mostrou uma ótima alternativa para a modelagem de emoções em softwares que utilizem o Java como principal linguagem de programação.

O Sistema Multiagentes formado por agentes cognitivos, que é o caso deste trabalho, serve para simularmos as ações de um agente conforme as mudanças que ocorrem no ambiente. Utilizando os agentes cognitivos, conseguimos ter um histórico de ações e através deste histórico conseguimos avaliar emoções positivas e negativas geradas no agente por um estímulo proveniente do ambiente.

Este artigo tem como objetivo apresentar que, a partir do modelo de rede Bayesiana de emoções proposto por Neves (2014), é possível definir personalidades para os agentes, como um meio de tornar mais verossímil a simulação de cenários. A partir dos resultados obtidos, verifica-se a possibilidade de simular agentes com diferentes personalidades dentro de um mesmo ambiente.

Com o prosseguimento da pesquisa, pretendemos realizar mais testes como personalidades em cenários mais complexos, envolvendo mais ações, como em um jogo computacional, onde os NPC (Non Player Character) podem agir de formas diferentes, dependendo da situação que se encontrarem.

6. Referências

ADAMATTI, D. F. **AFRODITE - Ambiente de Simulação Baseado em Agentes com Emoções**. Dissertação de Mestrado. Universidade Federal do Rio Grande do Sul. Porto Alegre. 2003.

ALVAREZ, L. O.; SICHMAN, J. Introdução aos Sistemas Multiagentes. **Jornada de Atualização em Informática**, Brasília, 1997. 1-38.

ARISTOTELES, **Rhetoric** (350 a.c.) – Traduzido por Manuel Alexandre Junior, 2a ed., Lisboa, 2005.

COZMAN, F. G. Bayesian Networks in Java: User manual and download. **JavaBayes**, 2001. Disponivel em: <http://www.cs.cmu.edu/~javabayes/Home/index.html>. Acesso em: 9 Abril 2013.

NEVES, F. S. **Modelagem de Emoções Usando Redes Bayesianas**. Dissertação de Mestrado. Universidade Federal do Rio Grande. Rio Grande, 2014.

FROZZA, R. **SIMULA: Ambiente para Desenvolvimento de Sistemas Multiagentes Reativos**. Dissertação de Mestrado. Universidade Federal do Rio Grande do Sul. Porto Alegre. 1997.

MORGADO, A. C. et al. **Analise Combinatória e Probabilidade**. Rio de Janeiro: [s.n.], 2001.

ORTONY, A.; CLORE, G.; COLLINS, A. **The Cognitive Structure of Emotions**. Cambridge: Cambridge University Press, 1988.

PEARL, J. **Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference**. 1a. ed: Morgan Kaufmann, 1988.

RUSSEL, S.; NORVIG, P. **Artificial Intelligence A Modern Approach**. 2a. ed. Upper Saddle River, New Jersey: Prentice Hall, 2003.

WERHLI, A. V. **Reconstruction of Gene Regulatory Networks from Postgenomic Data**. Tese de Doutorado. University of Edinburgh. Edinburgh, p. 230. 2007.

Aplicação para coleta e análise de dados de reputação de produtos em comércios eletrônicos

Eduardo Augusto Ferreira da Silva[1], Viviane Torres da Silva[2], Ricardo Choren[3]

[1] Universidade Federal Fluminense (UFF) – Niterói, RJ - Brasil

[2]IBM Research (on leave from UFF) – Rio de Janeiro, RJ – Brasil

[3]Instituto Militar de Engenharia(IME) – Rio de Janeiro, RJ – Brasil

eferreira@ic.uff.br, vivianet@br.ibm.com, choren@ime.eb.br,

Abstract. *Reputation systems applied in the e-commerce domain aims to collect and provide information on system objects, that may be products, users (costumers and providers) and organizations. A common problem of these systems is that the reputation of an object is not integrated, i.e., each system provides an different reputation for the same object. As a solution, this paper presents an application responsible for extracting data from various information sources regarding a product chosen by the user. It is able to integrate this information and present the reputation of the product on a consolidated way.*

Resumo. *Sistemas de Reputação aplicados no domínio de comércio eletrônico tem como objetivo coletar e disponibilizar informações sobre objetos do sistema, que podem vir a ser usuários do sistema (vendedores e compradores), produtos e organizações que comercializam os produtos. Um problema comum destes sistemas de reputação é a não integração das informações sobre um mesmo produto presente em diferentes bases de dados. Como solução, este trabalho apresenta um aplicativo responsável por coletar em diversas fontes de dados informações referentes a um produto escolhido pelo usuário, integrar estas informações e apresentar a reputação do produto de maneira consolidada.*

1. Introdução

A definição de comércio eletrônico (ou *e-commerce* em inglês) é dado pelo uso da internet para facilitar, executar e processar transações de negócios [Delone e Mclean 2004]. Estas transações de negócios envolvem um comprador (consumidor), um vendedor (fornecedor) e troca de produtos ou serviços por dinheiro. Toda transação, como, por exemplo, a venda de um determinado produto feita através do sistema, pode ser registrada. Portanto, é comum nos sistemas de comércios eletrônicos, como ocorre na Amazon, Buscapé e Ebay a existência de *Sistemas de Reputação* que possuem o objetivo de avaliar a reputação de usuários do sistema, dos produtos negociados e das organizações que negociam o produto.

Os sistemas de reputação têm a responsabilidade de coletar, distribuir e considerar o histórico de interações dos usuários [Mui, Mohtashemi e Halberstadt 2002]. Um problema comum nos sistemas de reputação, em especial, nos sistemas

associados às aplicações de comércios eletrônicos, é a não integração das informações sobre um mesmo produto disponíveis em diferentes sistemas de reputação (i.e, diferentes fontes de dados). Em [Khatiban 2012] é feito um experimento deste tipo integrado, porém sem ser automatizado. Como solução, este trabalho apresenta um aplicativo responsável por coletar em diversas fontes de dados informações referentes a um produto escolhido pelo usuário, integrar estas informações e apresentar a reputação do produto de maneira consolidada.

Os resultados dessas informações de reputação consolidadas pela aplicação em diversos comércios eletrônicos diferentes podem ser úteis nas seguintes problemáticas: (i) se obter reputação de um produto em que o sistema de reputação que ainda não tenha tido nenhuma experiência de interação com aquele produto; (ii) agir como uma fonte de informação complementar de reputação, considerando as interações referentes ao produto em outros comércios eletrônicos;

2. Proposta de Aplicação

Em busca de solução para resolver o problema da não integração de dados entre os sistemas de reputação para comércios eletrônicos, é proposta uma aplicação responsável por:

- Coletar em diversas fontes de dados, i.e., diferentes comércios eletrônicos, informações referentes a um produto escolhido pelo usuário;
- Integrar estas informações extraídas;
- Apresentar a reputação do produto de maneira consolidada.

Nas próximas seções, serão apresentados os detalhes da proposta da aplicação: a arquitetura; processo de coleta de dados; informações de reputação; armazenamento; e consulta de resultados.

2.1. Arquitetura

A arquitetura da aplicação é dividida em dois módulos. O módulo b*atch* é responsável pela extração e processamento de dados e o módulo *front-end* é responsável pela interface com o usuário final da aplicação. Esse módulo provê a funcionalidade de consulta da reputação do produto através das informações processadas previamente pelo módulo *batch*. A figura 1 representa a arquitetura da aplicação:

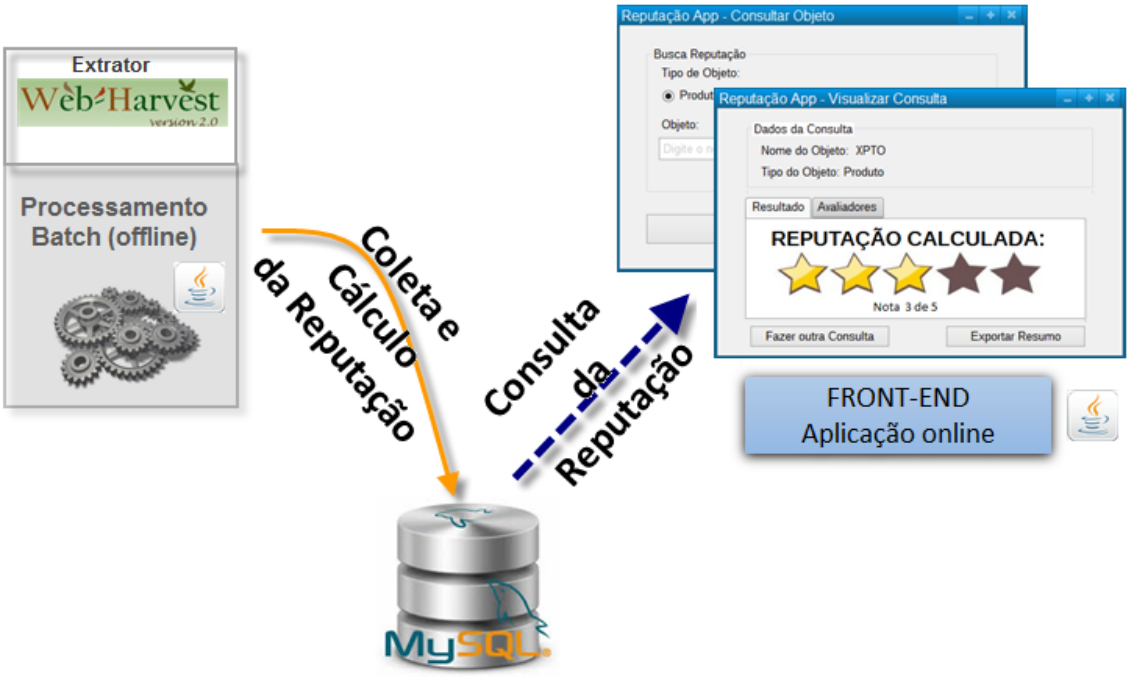


Figura 1. Arquitetura da Aplicação

Por se tratar de um trabalho acadêmico, foram adotados o uso de software livre. A solução prevê o uso da plataforma Java para ambos os módulos. Para o banco de dados relacional é utilizado o SGBD (Sistema de Gerenciador de Banco de Dados) MySql versão 5.6.23.

2.2. Módulo Batch: Coleta de dados

A ideia da aplicação é coletar em diferentes fontes de dados informações sobre as reputações de produtos. A extração da informação é feita *offline* pois o processo de extração é bastante custoso. O processo pode envolver a análise de páginas web ou a utilização de scripts disponibilizados pelos sistemas de comércio eletrônico para a extração de dados de suas páginas web. Nesta versão da ferramenta utilizamos um *plugin* feito em Java de uma ferramenta de extração (*scrapping*): Web-haverst* 2.0.

O módulo *batch* é capaz de colocar informação não somente sobre as avaliações feitas por usuários sobre um produto, mas também informações qualificando estas avaliações e informações associadas aos avaliadores do produto. As informações capturadas relativas a avaliações sobre um produto são, por exemplo: nota da avaliação, data da avaliação e características prós e contras do produto. As informações relativas a qualificação das avaliações são, por exemplo: quantos usuários avaliaram a avaliação e percentagem de avaliações positivas e negativas. Já as informações capturadas sobre o avaliador são, por exemplo: quantidades de avaliações realizadas sobre produtos similares e percentagem de avaliações positivas e negativas. Toda informação é armazenada em banco de dados.

A figura 2 ilustra a extração de dados nos comércios eletrônicos:

* Web-haverst: http://web-harvest.sourceforge.net

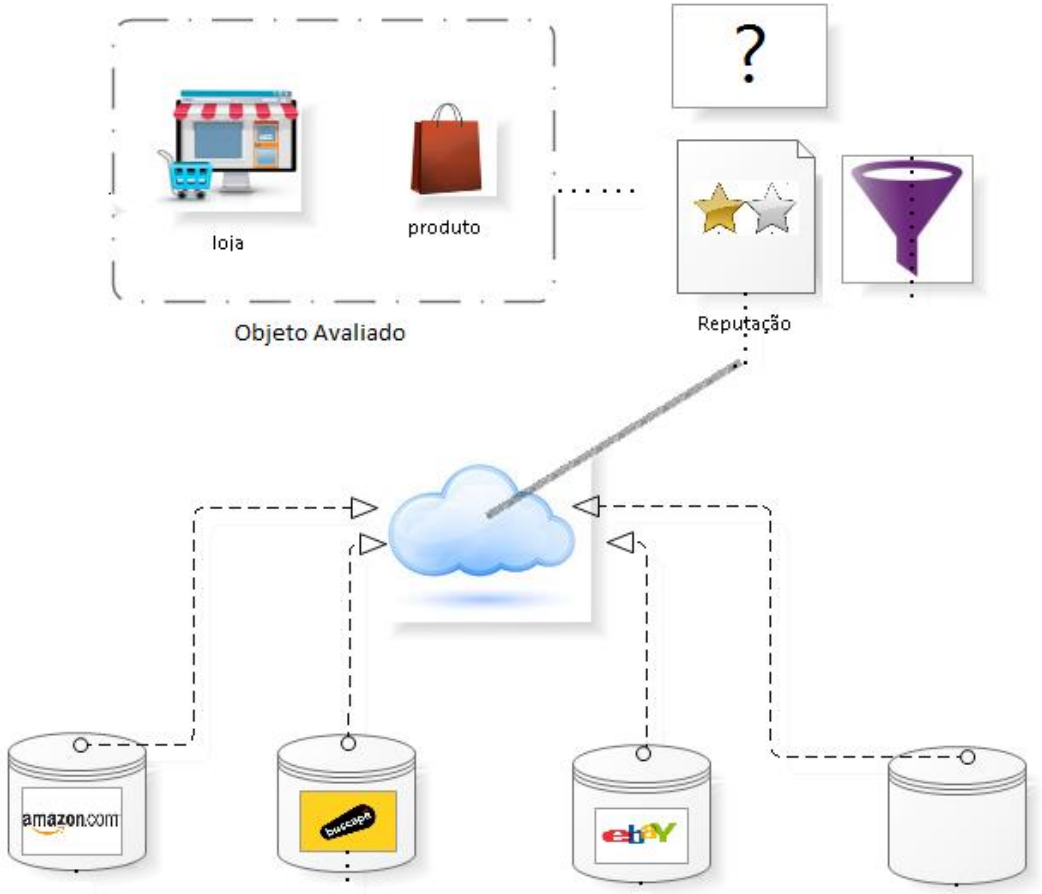


Figura 2. Coleta de dados

2.3. Informações de Reputação

Não é escopo deste trabalho detalhar o mecanismo utilizado para a consolidação das reputações de um mesmo produto. Este mecanismo ainda está em desenvolvimento. Contudo é possível afirmar que o mecanismo agrupa informações sobre as reputações dos produtos fornecida por diferentes usuários em diferentes sistemas de comércio eletrônico juntamente com informações relativas as avaliações sobre estas avaliações e informações relativas aos usuários que avaliaram o produto.

2.4. Consulta de Resultados

A consulta sobre a reputação de um determinado produto é realizada com base no nome deste produto. A figura 3 apresenta protótipos das interfaces gráficas utilizadas para consultar e para informar o resultado da consulta. O resultado da consulta informa não somente a reputação calculada mas também informações sobre os avaliadores que forneceram tais avaliações e sobre a qualificação de suas avaliações. Segue a figura 3:

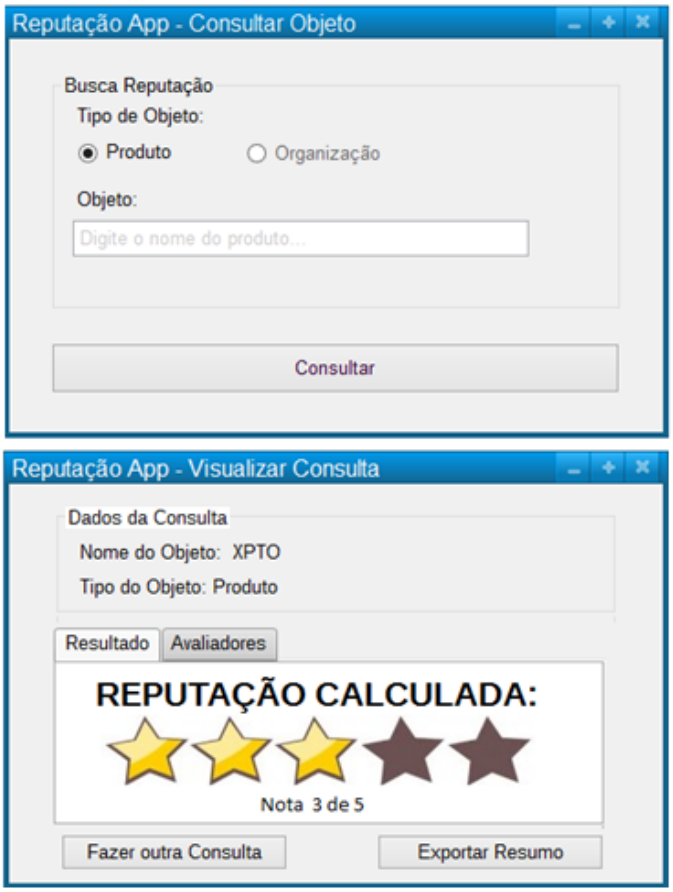


Figura 3. Interface gráfica da aplicação. Tela de Consulta e Tela de Visualização

3. Trabalhos Futuros

O módulo batch atualmente só utiliza um determinado script para a extração das informações. É nosso objetivo estender este módulo para que outros scripts sejam utilizados e outros sistemas de comércio eletrônico sejam analisados. Atualmente, conseguimos analisar os comércios eletrônicos Amazon, Buscape e Ebay.

Como comentado na seção 2.3, o mecanismo para consolidação das reputações ainda está em andamento. Atualmente estamos analisando as diferentes possibilidades de considerar as informações que capturamos com o objetivo de dar mais informação para o usuário sobre a reputação consolidada e sobre os avaliadores do produto. A medida que o mecanismo de consolidação for sendo definido acreditamos ser necessário adaptar a interface de apresentação dos dados.

4. Considerações

O objetivo deste trabalho é propor uma aplicação responsável por coletar em diversas fontes de dados informações referentes a um produto escolhido pelo usuário, integrar estas informações e apresentar a reputação do produto de maneira consolidada.

Este trabalho é inovador pois os sistemas de reputação disponíveis na literatura, como [Jacovi et al. 2014] [Sabater e Sierra 2002] [Mui, Mohtashemi e Halberstadt 2002] [Melnik e Alm 2002] [Resnick et al. 2006] [Valera et al 2001] calculam a

reputação dos produtos com base somente nas informações contidas no próprio sistema de reputação. Embora os sistemas de reputação digam que possuem uma abordagem para a Web, estes sistemas estão acoplados a somente um sistema de comércio eletrônico e utilizam somente as informações geradas por esse sistema para o cálculo da reputação dos produtos.

5. Referências

Delone, W. H., e Mclean, E. R. (2004). Measuring e-commerce success: Applying the DeLone & McLean information systems success model. International Journal of Electronic Commerce, 9(1), 31-47.

Jacovi, M. et al. (2014). The perception of others: inferring reputation from social media in the enterprise. In Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing (pp. 756-766). ACM.

Khatiban, S. (2012). Building reputation and trust using federated search and opinion mining. In Proceedings of the 21st international conference companion on World Wide Web (pp. 151-154). ACM.

Melnik, M. I., e Alm, J. (2002). Does a seller's ecommerce reputation matter? Evidence from eBay auctions. The journal of industrial economics, 50(3), 337-349.

Mui, L., Mohtashemi, M. e Halberstadt, A. (2002) "Notions of reputation in multi-agents systems: a review". In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. ACM, p. 280-287.

Resnick, P. et al. (2006). The value of reputation on eBay: A controlled experiment. Experimental Economics, 9(2), 79-101.

Sabater, J., e Sierra, C. (2002). Reputation and social network analysis in multi-agent systems. In Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1 (pp. 475-482). ACM.

Valera, F. et al. (2001). Communication management experiences in e-commerce: using a multiagent system to provide intermediation service in an e-commerce environment. Communications of the ACM, 44(4), 63-69.

Smart Parking: mecanismo de leilão de vagas de estacionamento usando reputação entre agentes

Wesley R. C. Gonçalves[1], Gleifer Vaz Alves[1]

[1]Departamento Acadêmico de Informática – Universidade Tecnológica Federal do Paraná(UTFPR)
Caixa Postal 84.016 – 210 – Ponta Grossa – PR – Brazil

wesleygoncalves@alunos.utfpr.br, gleifer@utfpr.edu.br

Abstract. *Finding a spot in a parking lot which is almost full is not an easy task for a driver. It may occur that a person drives for a considerable time until find a parking space. This paper presents the modeling of a negotiation mechanism among agents (drivers) in a parking lot. Such negotiation is done by means of an auction which is established according to driver's reputation.*

Resumo. *Encontrar uma vaga livre num estacionamento quase cheio é uma tarefa difícil para um motorista. Ele pode percorrer o estacionamento por vários minutos até conseguir uma vaga. O artigo apresenta a modelagem de um mecanismo de negociação de vagas entre agentes (motoristas) em um estacionamento. A negociação é feita através de um leilão de vagas baseadas na reputação do motorista.*

1. Introdução

Segundo [Giffinger et al. 2007], cidades inteligentes almejam a criação e aplicação de métodos inteligentes e criativos para aprimorar o uso dos recursos de uma cidade por meio do uso da tecnologias da informação e da comunicação. Possibilitando melhora na qualidade de vida para os moradores da cidade.

Uma cidade possui diversos tipos de recursos e serão melhor aproveitados com mecanismos inteligentes, pois tornam os componentes da cidade mais eficientes na utilização destes recursos. O estacionamento é uma componente da cidade, é nele que os motoristas mantém seus carros enquanto realizam suas tarefas. A quantidade de vagas existentes dentro de um estacionamento é o número limite para a quantidade de carros que poderão ser mantidos ao mesmo tempo dentro de um estacionamento, sendo desta forma um recurso de quantidade fixa. Devido a este limite, se faz necessário a criação de mecanismos inteligentes para alocação de recursos.

Existem os recursos do motorista que são o tempo e o combustível. Tais recursos são gastos ao procurar por uma vaga desconhecida num estacionamento. Em horário de pico o motorista lentamente percorre o estacionamento a procura de qualquer vaga livre, enquanto não encontrar uma vaga outros motoristas que também procuram uma vaga formam filas que dificultam o uso do estacionamento, até mesmo para os motoristas que querem sair do estacionamento e deixar uma vaga livre.

Para tentar minimizar estes problemas será criado um sistema multiagente (SMA) que utilizará a comunicação entre agentes para possibilitar o compartilhamento de informações úteis para o funcionamento mais eficiente do estacionamento.

A reputação dos motoristas é utilizada como critério para ganhar uma vaga mediante um leilão. A reputação é a credibilidade que um agente possui de outro agente ser honesto e confiável, o objetivo da reputação no sistema é minimizar erros por conta das ações dos agentes. Em sistemas multiagentes a reputação surge como uma forma de avaliação para incentivar os agentes a serem o mais honestos em suas atitudes. Uma vez que os agentes podem decidir em ser honestos ou não. Fazendo-se necessário o incentivo para que estes agentes contribuam para o sistema como um todo [Huynh et. al., 2006].

O objetivo deste trabalho é utilizar as características de um estacionamento e as informações disponíveis para criar um SMA, capaz de organizar o uso do estacionamento, tentando alocar as vagas de uma forma mais adequada e que incentive o motorista a cooperar com o sistema.

Na literatura existem abordagens relacionadas que discutem o uso de agentes inteligentes para problemas de trânsito e estacionamento. Destaca-se que em [Bazzan, 2012], a autora relaciona o conceito de cidades inteligentes com transporte, mostrando que tratar cada veículo como um agente é uma forma de criar soluções de cooperação para o trânsito. É um artigo que aprofunda em conceitos de engenharia de trânsito para relacionar com sistemas multiagentes, mostrando possibilidades de modelagem de problemas.

Em [Di Napoli et. al., 2014] é criado um modelo completo para estacionamento com apresentação do modelo e simulações, o que difere da abordagem apresentada neste artigo é que no *Smart Parking* objetiva-se resolver o problema de alocação pensando no relacionamento de agentes para distribuir vagas, enquanto a abordagem é utilizar equipamentos sofisticados que elevam a precisão de funcionamento porém aumenta muito os custos do projeto.

Trabalhos sobre estacionamentos inteligentes como [Pala et. al., 2007] e [Chinrungrueng, 2007] utilizam dispositivos físicos para criar mecanismos que possibilitam benefícios automatizados para o motorista utilizar o estacionamento porém não estimulam o motorista a melhor utilizar o estacionamento e tem como desvantagem o custo de equipamentos instalados no veículo e no estacionamento, o que dificulta sua implantação.

Este artigo é estruturado da seguinte forma. Na seção 2 descreve-se o problema da alocação de vagas em um estacionamento. Na seção 3 é apresentado o desenvolvimento do modelo e como é a adaptação dos itens do mundo real para a representação em sistema multiagentes. Na seção 4 é apresentado o conceito de reputação e como ele é calculado no modelo criado. Na seção 5 são apresentados os estados que o agente estacionamento pode permanecer e como isso afeta o modelo de leilão de vagas. Na seção 6 têm-se as considerações finais.

2. Definição do Problema

O modelo tradicional de estacionamento não disponibiliza para os motoristas informações úteis sobre o estacionamento. A dificuldade do motorista ao utilizar o estacionamento é entrar no estacionamento sem saber se existe ou não vaga para estacionar seu veículo. Há também o problema do motorista não saber se existem vagas livres próximas a elevadores e portas, necessitando estacionar na primeira vaga livre que encontrar, para assim garantir um lugar no estacionamento. Estes problemas decorrem da falta de um mecanismo que melhor aproveite as informações do estacionamento.

Se o estacionamento estiver vazio ou pouco ocupado o motorista facilmente localiza as vagas livres. Porém em horários de pico, o estacionamento está com grande parte das vagas ocupadas obrigando o motorista a desperdiçar tempo e combustível à procura da vaga e nesses momentos é o que motorista mais se preocupa em economizar tempo.

3. Modelo

No sistema *Smart Parking*, existem dois tipos de agentes: o agente motorista e o agente estacionamento. Os agentes motoristas são aqueles que representam o motorista e seu veículo no sistema para interagir com outros agentes para compartilhar informações e também irá interagir com o agente estacionamento. O agente estacionamento representa o próprio estacionamento, ou seja, existe um agente estacionamento para cada estacionamento, e é responsável pelo gerenciamento de vagas por meio de leilões.

O agente motorista possui os objetivos locais e globais. O objetivo local é aquele que o agente faz para interesse próprio, o qual neste contexto é conseguir reservar uma vaga. O objetivo global é cooperar e organizar da melhor forma o estacionamento, otimizando como um todo o funcionamento do mesmo.

O modelo tem como objetivo configurar as trocas de mensagens entre os agentes, eles vão tomar decisões importantes referentes ao uso do estacionamento. Estas informações darão o suporte para aproveitar o recurso que são as vagas e quais os critérios utilizados para escolher qual motorista é o melhor para ocupar determinada vaga.

As vagas que não estiverem em uso estarão sendo negociadas pelo agente estacionamento. As vagas estarão organizadas em uma fila de prioridade para serem leiloadas. A alocação de cada vaga para um determinado motorista é dada por um leilão. O modelo de leilão utilizado é baseado no leilão Holandês. O Leilão Holandês [Krishna, 2009] é um tipo de leilão onde o item a ser leiloado inicialmente possui um valor muito alto, o qual dificilmente será adquirido pelos compradores. Quando não há nenhum comprador interessado o valor é decrementado e o item é então novamente leiloado, até encontrar um comprador que aceite a proposta. Este modelo foi escolhido porque possibilita maior velocidade para ser realizado, este tempo otimizado reflete na alocação das vagas.

O leilão deste modelo possui a mesma estrutura do leilão Holandês, a diferença básica é que no modelo utilizado no *Smart Parking* o valor utilizado como critério de escolha é a pontuação da reputação do agente motorista.

Comparado a outras técnicas de leilão este é o mais rápido uma vez que o valor é apenas decrementado existindo uma margem limite para o valor decrementar, desta forma ao chegar no valor mínimo sem lances realizados pelos compradores o leilão é reiniciado [Krishna, 2009].

Cada vaga será leiloada separadamente e o que determinará a ordem das vagas para serem aplicadas ao leilão será a pontuação que cada vaga possui. Esta pontuação se refere a proximidade da vaga para pontos de saídas do estacionamento pelos motoristas e passageiros após saírem do carro, por exemplo: elevadores, portas de acesso, dentre outras. Esta pontuação ocorre de forma a facilitar o uso do estacionamento pelos motoristas, uma vez que veículos estacionados neste tipo de vagas possibilitarão aos motoristas andarem menos e perder menos tempo para sair do estacionamento após estacionar o carro.

O agente estacionamento por meio do leilão escolhe o melhor motorista dentre todos que estão disputando aquela vaga. O motorista escolhido é o que possuir melhor reputação entre aqueles que estão na disputa pela vaga. A vaga então é reservada para o motorista que ganhou o leilão, porém se ele não ocupar a vaga até um determinado tempo, o motorista recebe uma penalização em sua reputação e a vaga será leiloada novamente.

No momento da escolha do melhor agente motorista caso haja empate na primeira casa decimal o critério de desempate será a distância do agente para a vaga. Este critério preza pela organização do estacionamento uma vez que agentes com reputação tão parecidas têm o mesmo direito pela vaga, restando desempatar com um critério que beneficie a todos.

Se o motorista ocupar a vaga no tempo determinado, ele permanecerá o tempo necessário ocupando a vaga. No momento que deixar a vaga o motorista avisa o sistema que ele está deixando a vaga e assim a vaga estará livre. O motorista recebe uma bonificação por ter cooperado com o sistema melhorando assim sua reputação. A vaga livre para alocação será colocada novamente na fila de alocação para um novo motorista utilizar.

4. Reputação

A reputação será o valor que o agente motorista oferece ao agente estacionamento para participar do leilão por uma vaga.

A equação 1 representa o cálculo da reputação do agente M1:

Reputação(M1) = ((MédiaVagaOcupada(M1) + MédiaVagaOferecida(M1)) / 2) (1)

Reputação(M1): é a média de dois fatores: MédiaVagaOcupada e MédiaVagaOferecida. Os quais representam o quão bem o agente usa e coopera com a utilização do estacionamento.

A reputação varia entre zero e um, zero é o pior valor para a reputação e um é o melhor valor para a reputação. A equação 2 representa o cálculo da variável MédiaVagaOcupada:

MédiaVagaOcupada(M1) = NumVagaOcupada(M1)/NumVagaReservada(M1) (2)

MédiaVagaOcupada: é uma variável que representa o comportamento do agente motorista ao utilizar uma vaga. Por exemplo: para o agente motorista obter pontuação mínima que é zero, ele deve ocupar nenhuma vaga que é reservada para ele. Para obter pontuação máxima o agente motorista deve ocupar toda vaga que conseguir reservar. A equação 3 representa o cálculo de MédiaVagaOferecida:

MédiaVagaOferecida(M1) = NumVagaOferecida(M1)/ NumVagaOcupada(M1) (3)

MédiaVagaOferecida: é a média de pontuação do motorista M1 referente a quantidade de vagas oferecidas ao sistema pelo agente motorista em relação ao número de vagas ocupadas pelo agente. Se o agente motorista avisa o agente estacionamento antes de sair da vaga ele irá otimizar o uso da vaga, uma vez que o leilão poderá acontecer antecipadamente, ocasionando bonificação para o mesmo devido a contribuição ao uso do sistema. O sistema pode ter a informação de que o motorista avisou que a sua vaga está livre, por meio de um sensor na cancela de saída do estacionamento.

O cálculo das variáveis Reputação, MédiaVagaOcupada e MediaVagaOferecida é realizado pela média da somatória das n últimas pontuações de cada variável, sendo n um número estabelecido ao inicializar o sistema. Utilizar o histórico da reputação será útil para incentivar o agente motorista utilizar o estacionamento da melhor maneira sempre, e não apenas por um tempo.

Se houver empate o agente mais próximo ganhará a vaga, pois o sistema visa melhor utilizar o estacionamento como um todo. O agente próximo a vaga tende a alcançar mais rápido a vaga e será um veículo a menos procurando por vaga.

5. Mensagens Trocadas e seu significado

Os agentes necessitam de uma forma para indicar a outros agentes sua decisão sobre as negociações de vagas. A comunicação será baseada na troca de mensagens entre os agentes utilizando a linguagem e protocolo KQML (*Knowledge Query and Manipulation Language*). A linguagem utiliza performativas para que os agentes identifiquem o tipo de mensagem e seu conteúdo. Será utilizada como meio de realizar os leilões de vagas no estacionamento. A seguir as mensagens que serão desenvolvidas no sistema para comunicação entre agentes:

Leilão de vaga**:** se houver pelo menos uma vaga livre, o agente estacionamento envia esta mensagem para os agentes motoristas interessados em estacionar o carro. Este tipo de mensagem é continuamente enviada até obter resposta positiva de pelo menos um agente motorista.

Recusa de participação em leilão: o agente motorista que procura vaga responde ao agente estacionamento recusando aquela vaga ofertada.

Aceite de participação em leilão: o agente motorista responde ao agente estacionamento aceitando a participação no leilão, lhe enviando a sua reputação e também sua posição geográfica no conteúdo da mensagem.

Vencedor da Vaga: O agente estacionamento envia esta mensagem para o agente motorista vencedor indicando a posição da vaga.

Oferecer Vaga: O agente estacionamento envia esta mensagem para o agente estacionamento avisando que oferece a vaga, indicando a posição da vaga.

6. Considerações Finais

O presente trabalho tem como meta principal iniciar a modelagem do problema de alocação de vagas em um estacionamento com sistema multiagente, além de incentivar o motorista a cooperar afim de melhorar a eficiência do sistema. A reputação foi desenvolvida pensando no modo como o agente estacionamento negociaria o recurso vaga com os agentes motoristas, um tipo de reputação na forma como o motorista utiliza o sistema e o tempo que ele leva para realizar as próprias tarefas, uma vez que o tempo é um recurso utilizado pelo motorista para retirar o carro da vaga ou estacionando o carro na vaga.

Em relação a continuidade do trabalho destaca-se: (i) implementar o mecanismo de comunicação entre agentes; (ii) implementar o SMA fazendo uso do mecanismo de reputação; (iii) fazer alterações no mecanismo de reputação e comparar resultados. Dentre as alterações do mecanismo, é possível considerar o SMA tendo o agente motorista capaz de negociar diretamente a vaga com outro agente motorista.

Referências

Bazzan, Ana LC (2012). "Lessons learned from one decade of developing agent-based tools for traffic modeling, simulation, and control: how to make cities smarter." *VIII Simpósio Brasileiro de Sistemas de Informação.*

Chinrungrueng, Jatuporn, Udomporn Sunantachaikul, and Satien Triamlumlerd (2007). "Smart parking: An application of optical wireless sensor network." *Applications and the Internet Workshops, 2007. SAINT Workshops 2007.* International Symposium on. IEEE. Di Napoli, Claudia, Dario Di Nocera, and Silvia Rossi (2004). "Negotiating parking spaces in smart cities." *Proceeding of the 8th International Workshop on Agents in Traffic and Transportation, in conjunction with AAMAS.*

Di Napoli, Claudia, Dario Di Nocera, and Silvia Rossi. "Negotiating parking spaces in smart cities." *Proceeding of the 8th International Workshop on Agents in Traffic and Transportation, in conjunction with AAMAS.* 2014.

Giffinger, Rudolf, et al. (2007). Smart cities-Ranking of European medium-sized cities. Vienna University of Technology.

Huynh, Trung Dong, Nicholas R. Jennings, and Nigel R. Shadbolt. (2006). "An integrated trust and reputation model for open multi-agent systems." *Autonomous Agents and Multi-Agent Systems* 13.2, pages: 119-154.

Krishna, Vijay. *Auction theory.* Academic press, 2009.

Pala, Zeydin, and Nihat Inanc (2007). "Smart parking applications using RFID technology." RFID Eurasia, 2007 1st Annual. IEEE.

Transferência de Confiança durante Trocas Sociais em Tríades de Agentes utilizando Relações de Dependência e Reputação

Yunevda E. León Rojas[1], **Diana F. Adamatti**[1], **Graçaliz P. Dimuro**[1]

[1]Mestrado em Engenharia de Computação
Centro de Ciências Computacionais
Universidade Federal do Rio Grande (FURG)
Av. Itália Km 8, Campus Carreiros – 96.201-900 – Rio Grande – RS – Brazil

{yuniekita,dianaada,gracaliz}@gmail.com

Abstract. *The process of social exchanges defines values of exchanges generated by interactions between two agents. In this article, it is added a third agent, named intermediate agent, which has the possibility of outsourcing a service on behalf to other agent. The generated exchange values become a focus to analysis of the confidence transfer process between agents that never interact directly and they do not know among them, but they will do an exchange by mean an intermediary agent. Also, it is considered the dependency relation and reputation of each agent concepts. In this way, the confidence transfer allows to orient the agents decision for its partner choosing and in the achievement of future exchanges.*

Resumo. *O processo de troca social define valores de troca gerados em interações entre dois agentes. Nesse artigo, adiciona-se um terceiro, denominado agente intermediário, quem tem a possibilidade de terceirizar um serviço em favor de outro agente. Os valores de troca gerados se tornam foco para análise do processo de transferência de confiança entre agentes que nunca interagiram diretamente e não se conhecem, mas que realizaram uma troca por meio do agente intermediário. Alem disso, são considerados os conceitos de relações de dependência e reputação de cada agente. Dessa forma, a transferência de confiança permite orientar a decisão dos agentes na escolha de seu parceiro e na realização das trocas futuras.*

1. Introdução

A análise de modelos e teorias de interatividade da sociedade humana permite observar que existem diversas formas de interação. Cada uma dessas interações é considerada como um processo de troca social entre pares de agentes [Piaget 1995]. Diversos trabalhos discutem as interações sociais com maior detalhe [Pereira et al. 2008, Dimuro et al. 2011, Farias et al. 2013].

Troca social é o nome dado ao processo de interações entre indivíduos, os quais realizam serviços uns para os outros, gerando valores de trocas materiais (investimento e satisfação) e virtuais(crédito e débito). Outros conceitos importantes, que estão relacionados com interações entre agentes, são relações de dependência, reputação e confiança.

Uma relação de dependência, segundo [Sichman and Demazeau 1994], se dá quando um agente quer alcançar um estado, que é o seu objetivo, mas não tem a possibil-

idade de alcança-lo. Portanto, procura um segundo agente que tenha todas as condições necessárias para ajudá-lo.

A reputação é vista como uma ferramenta social que tem o objetivo de reduzir a incerteza de se interagir com indivíduos de atributos desconhecidos, encontrando-se com mais detalhe nos estudos de [Castelfranchi et al. 2000, Castelfranchi and Falcone 1998, Bromley 1993, Marsh 1994].

A proposta deste trabalho é relacionar esses conceitos sob a luz da teoria das trocas sociais, visando analisar em que condições pode se realizar a transferência de confiança em um sistema multiagente.

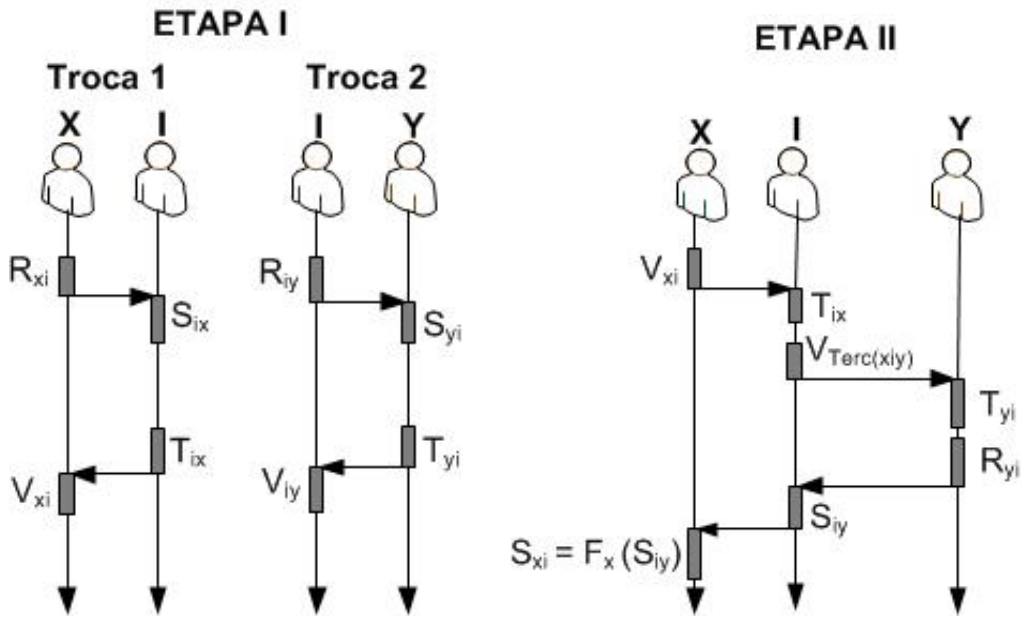
2. Modelos Propostos

Quatro modelos serão apresentados, trocas sociais com agente intermediário, relações dependência, reputação e transferência da confiança são descritos por meio de equações que se integram sequencialmente.

2.1. Modelo de Trocas Sociais com Agentes Intermediários (TSAI)

O modelo proposto, denominado Troca Social com Agente Intermediário $(TSAI)$, considera três agentes, X, I e Y, com características independentes. Tanto o agente X quanto o agente Y conhecem e mantêm comunicação com o agente I, mas não conhecem um ao outro, enquanto o agente I conhece e mantém contato com ambos, X e Y. Esses agentes interagem em duas etapas de trocas.

A Etapa I é composta por dois processos de troca independentes. Na *Troca 1*, o agente X realiza um serviço para o agente I, e na *Troca 2*, o agente I realiza um serviço para o agente Y. Gerando em cada interação valores de trocas conforme descrito na Figura 1a.



(a) Etapa I (b) Etapa II

Figure 1. Trocas Social com Agente Intermediário - TSAI

- **Etapa I - Troca 01**:
    1. R_{xi}: Valor do *investimento* do agente X na realização de um serviço para o agente I.
    2. S_{ix}: Valor de *satisfação* do agente I pelo serviço recebido do agente X.
    3. T_{ix}: Valor do *débito* do agente I pela satisfação do serviço recebido de X.
    4. V_{xi}: Valor do *crédito* que o agente X adquiriu com I pela realização de um serviço.

- **Etapa I - Troca 02**:
    1. R_{iy}: Valor do *investimento* do agente I na realização de um serviço para o agente Y.
    2. S_{yi}: Valor de *satisfação* do agente Y pelo serviço recebido do agente I.
    3. T_{yi}: Valor do *débito* do agente Y pela satisfação do serviço recebido de I.
    4. V_{iy}: Valor do *crédito* que o agente I adquiriu com Y pela realização de um serviço.

Como é apresentado na Figura 1, na etapa II existe uma possível cobrança do crédito do agente X ao agente I. Essa cobrança envolve um serviço relativo ao valor virtual (V_{xi}) gerado pelo serviço realizado em favor de I na Etapa I. Em vista disso, o agente I impedido da execução do serviço solicitado, e, reconhecendo o valor virtual (T_{ix}) de débito gerado pelo serviço recebido de X na Etapa I, realiza um processo de terceirização, que consiste na busca de um terceiro agente que possa realizar o serviço que foi solicitado para ele.

- **Etapa II**:
    1. V_{xi}: Valor do *crédito* do agente X pela realização de um serviço para agente I.
    2. T_{ix}: Valor de *débito* do agente I pelo serviço recebido do agente X.
    3. $V_{terc(i|y)}$: Valor de *crédito de terceirização* do agente I para com X na procura de um terceiro agente. É buscado algum agente Y, que tenha com I um valor de crédito V_{iy} equivalente a T_{ix}.
    4. T_{yi}: Valor do *débito* do agente Y para com I pela satisfação com o serviço recebido.
    5. R_{yi}: Valor de *investimento* do agente Y na realização de um serviço para agente I.
    6. S_{iy}: Valor de *satisfação* que o agente I adquiriu com Y pela realização de um serviço.
    7. S_{xi}: Valor do *satisfação* que o agente X adquiriu com I pela realização de um serviço, gerado pelo investimento de Y para I. O valor de satisfação vai estar em função da avaliação do agente X, e vai ser representado por: $S_{xi} = F_x(S_{iy})$.

3. Dependência em TSAI

Os agentes X, I e Y, que se encontram interagindo em trocas de serviços, tem a cada iteração valores de trocas, (r, s, t, v), nomeadas como crenças de histórico de valores de trocas. Além disso, cada agente possui lista de serviços que pode oferecer. Isso gera entre eles dependências explícitas e dependências implícitas, como apresenta a Figura 2. As dependências explícitas, são criadas entre agentes que se conhecem e tiveram algum

tipo de interação de troca na prestação dos serviços, X - I e I - Y. As dependências implícitas, são relações geradas entre agentes que não se conhecem, X - Y, mas que realizaram alguma interação na troca de serviços por meio de um agente intermediário, comum a ambos.

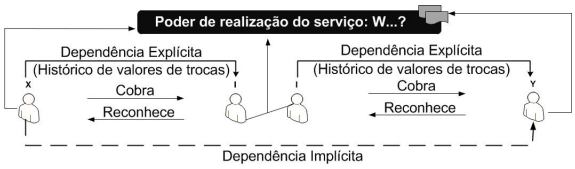


Figure 2. Relações de Dependência em TSAI

Além disso, podem existir entre os agentes diferentes graus de dependências: **Dependências muito fracas**, caracterizadas por um agente que não tem crédito a cobrar. Além disso, este precisa de um agente para realizar um serviço que ele mesmo poderia efetuar, já que sabe fazer o serviço. **Dependências fracas**, caracterizadas por um agente ter um crédito a cobrar e precisar realizar um serviço que ele mesmo poderia efetuar, já que sabe fazer o serviço. **Dependências fortes**, caracterizadas por um agente não ter um valor de crédito a cobrar e também ter que realizar um serviço que dependa de um outro agente, pois ele não tem o conhecimento para efetuar o serviço. Finalmente. **dependências muito fortes**, aquelas em que o agente tem um crédito a cobrar e precisa de um serviço que não sabe efetuar.

4. Reputação em TSAI

O modelo da reputação em TSAI, como mostra a Figura 3, é composto por três agentes X, I e Y, e é caracterizado por que cada um dos agentes estão efeituando avaliações direta ou indiretas dos serviços recebidos em cada umas das TSAI. Deste modo, faz parte de agente o histórico de valores de trocas e também o conhecimento do grau de dependência existente.

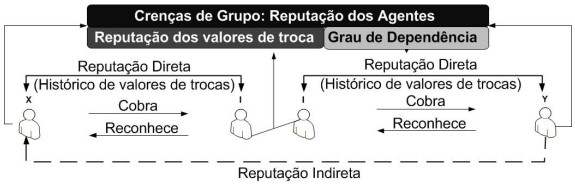


Figure 3. Relações de Reputação

4.1. Construção das crenças da reputação em TSAI

A construção das crenças sobre o valor de reputação dos valores de troca, $Vrep$, é produto da comparação do valor de satisfação observado e o valor de satisfação desejado, na prestação e contraprestação dos serviços na primeira e segunda etapa, respetivamente. Detalha-se a seguir uma interação:

- I avalia o serviço entregado pelo agente X na etapa I - troca 1:
  - I gera um valor de satisfação na recepção do serviço produto do investimento feito pelo agente X.

– I analisa X por comparação na tricotomia do valor de satisfação observada na recepção do serviço e satisfação desejada como se mostra a seguir: $S_{ix} > vd_{satisf}$; $S_{ix} = vd_{satisf}$ e $S_{ix} < vd_{satisf}$, para obter um valor de reputação dos valores de troca $V_{Rep} \in \{1; 0,5; 0\}$, respetivamente.
– I faz o registro sobre o valor de reputação dos valores de troca de X obtido no item anterior.

As avaliações direta e indireta são representadas como reputações diretas, na interação com um agente que se conhece, e reputações indiretas, criada sobre um agente que não conhecem-se.

5. Transferência de Confiança em TSAI

O processo da transferência de confiança em TSAI, conforme o modelo da Figura 4, esta composto por três agentes X, I e Y. O agente X que não conhece ao agente Y, avaliará a possibilidade de transferir sua confiança para um agente com quem não se teve iterações diretas mas existe informação sobre ele.

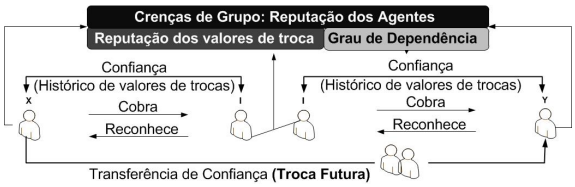


Figure 4. Relações de Transferência da Confiança

Para isso é necessário desenvolver uma análise das crenças do grupo existentes. Estas crenças são compostas pela reputação de cada agente, considerando que, o valor de reputação utilizado é calculado levando em conta o histórico de valores de trocas e as relações de dependência. Detalha-se a seguir uma interação:

Para uma iteração em tríades de agentes: X, I, Y:

1. X quer interagir com um agente desconhecido Y para alcançar seu objetivo sob determinadas circunstâncias:
   - X consulta o valor de reputação fornecido por I sobre Y: Rep_i^y
2. X **transfere a confiança** para Y, **confiando** em I, para alcançar seu objetivo quando:
   - X para confiar em I, verifica e compara se a reputação que tem sobre ele Rep_x^i é maior ou igual com o valor de reputação que ele considera aceitável, assim: $Rep_x^i \geq R_{acep}^x$
3. X, **não transfere a confiança** para Y, **não confiando** em I, quando:
   - X verifica e compara se, a reputação que ele tem sobre I, Rep_x^i é menor do que o valor de reputação que ele considera aceitável, $Acep$: $Rep_x^i < R_{acep}^x$

Dessa forma, em uma troca futura, X poderá transferir sua confiança e se relacionar diretamente com Y.

6. Considerações Finais

No mundo real é muito comum a existência de intermediários no processo de trocas. Portanto, é importante considerar a extensão da teoria das Trocas Sociais de Piaget para três agentes.

Os modelos apresentados são o início para a análise dos aspectos não econômicos dos processos de trocas sociais, e, para entender o comportamento dos agentes em iterações maiores a três agentes, tendo em consideração os conceitos de dependência, reputação e confiança, de forma interdependente. Permitindo orientar a decisão dos agentes sobre a escolha de parceiros com quem nunca tiveram interação direta e se convertem no foco para as análises do processo de transferência de confiança entre os agentes.

Os modelos propostos teriam a possibilidade de ser submetidos na realização de simulações e avaliações por meio de estudos de caso para validar-lhas quantitativamente.

References

Bromley, D. B. (1993). *Reputation, image, and impression management*. Wiley, New York.

Castelfranchi, C. and Falcone, R. (1998). Principles of trust for mas: Cognitive anatomy, social importance and quantification. In *International Conference of Multi-agent Systems (ICMAS)*, pages 72–79.

Castelfranchi, C., Falcone, R., Firozabadi, B., and Tan, Y. (2000). Special issue on trust, deception and fraud in agent societies. *Applied Artificial Intelligence Journal*, pages 763–768.

Dimuro, G. P., Costa, A. R. C., Gonçalves, L. V., and Pereira, D. (2011). Recognizing and learning models of social exchange strategies for the regulation of social interactions in open agent societies. pages 143–161.

Farias, G. P., Dimuro, G., Dimuro, G., and Jerez, E. D. M. (2013). Exchanges of services based on Piaget's theory of social exchanges using a BDI-fuzzy agent model. In *Proceedings of 1st BRICS Countries Congress (BRICS-CCI) and 11th Brazilian Congress (CBIC) on Computational Intelligence*, pages 653–658. IEEE.

Marsh, S. (1994). *Formalising Trust as a Computational Concept*. Tese (doutorado), Department of Computing Science - University of Stirling. 184p.

Pereira, D., Gonçalves, L., Dimuro, G. P., and Costa, A. R. C. (2008). Towards the self-regulation of personality-based social exchange processes in multiagent systems. In Zaverucha, G. and Costa, A., editors, *Advances in Artificial Intelligence - SBIA 2008*, pages 113–123. Springer.

Piaget, J. (1995). *Sociological Studies*. Routlege, London.

Sichman, J. S. and Demazeau, Y. (1994). A first attempt to use dependence situations as a decision criterion for choosing partners in multi-agent systems. In *Workshop on Decision Theory for DAI Aplicartions*.

Uma abordagem baseada em Sistemas Multiagentes para suporte a Telemedicina

Ariel E. Endara[1], Marx Viana[1], Francisco J. P. Cunha[1], Carlos J. P. de Lucena[1]

[1]Departamento de Informática – Pontifícia Universidade Católica (PUC-RJ)
Rua Marquês de São Vicente, 255 – 22.451-900 – Rio de Janeiro – RJ – Brasil

{aendara, mleles, fcunha, lucena}@inf.puc-rio.br

Abstract. *Telemedicine is the set of technologies and applications that enable the performing of medical actions at a distance. In this context, to assist the improvement of health services, software development has become a promising area of research, however, the health area has been suffering from the dissatisfactions and deficiencies in itself, which aggravate and defy the interaction between doctors and patients. To address such issue, this paper proposes an architecture based on multi-agent systems, trying to provide mechanisms to build applications able to monitor and interact in real time, getting a better doctor-patient relationship through mobile devices.*

Resumo. *Telemedicina é o conjunto de tecnologias e aplicações que permitem a realização de ações médicas a distância. Neste contexto, o desenvolvimento de software para auxiliar o melhoramento dos serviços de saúde tornou-se uma área promissora de pesquisa. No entanto, a área de saúde vem sofrendo diante das insatisfações e deficiências encontradas neste setor, que agravam e desafiam a interação entre médicos e pacientes. Com base neste aspecto, o presente trabalho propõem uma arquitetura fundamentada em sistemas multiagentes, visando fornecer mecanismos necessários para a construção de aplicações capazes de monitorar e interagir, em tempo integral, a relação médico-paciente através de dispositivos móveis.*

1. Introdução

A necessidade de prover serviços de qualidade é um dos grandes desafios na área de saúde pública. Nesse sentido, as insatisfações e deficiências encontradas neste setor como ausência de infraestrutura, corpo técnico deficitário e despreparado, não podem ser ignoradas uma vez que agravam e desafiam as políticas públicas. Por este motivo o desenvolvimento de software para este setor, se torna um grande desafio. Portanto, viu-se uma oportunidade de auxiliar a área medica através da criação de uma aplicação para dispositivos móveis, onde a relação médico-paciente pudesse ser melhorada. Estes sistemas são caracterizados por serem, muitas vezes, distribuídos e compostos de entidades autônomas que interagem umas com as outras. Um agente de software pode ser definido como um sistema de computador situado em um ambiente e capaz de agir de forma autônoma para atingir um objetivo, onde a autonomia refere-se à capacidade de agir de acordo com sua própria linha de controle [Weiss 1999]. Para a construção de sistemas complexos, geralmente são utilizados vários agentes. Um sistema que possui

vários agentes atuando em um ambiente é denominado sistema multiagentes ou SMA [Weiss 1999].

Telemedicina pode ser definida como a prestação de serviços médicos aos pacientes com uma combinação entre telecomunicações e tecnologias multimídia, provendo o serviço de saúde sem importar onde eles se encontrem [Al-Taei 2005]. Esta tecnologia oferece potenciais benefícios para a relação médico-paciente, pois dessa forma os médicos podem se concentrar em tarefas mais importantes, como atender pacientes com doenças crônicas. E os pacientes, por sua vez, poderão se comunicar com o médico e ter informação sem ter que fazer longas viagens [Chan et al. 2008].

Procurando utilizar-se dos benefícios que a tecnologia de sistemas multiagentes e a telemedicina podem oferecer, este trabalho propõe uma arquitetura de software capaz de estabelecer uma comunicação direta entre médicos e pacientes através de dispositivos móveis. No entanto, ampliar a supervisão da saúde da clínica ou hospital para o entorno do lar não pode ser feito como uma repetição dos mesmos protocolos e procedimentos clínicos, já que esse enfoque acrescentaria drasticamente o tempo e os recursos humanos, além de que pode ser inadmissível pelos pacientes devido a sua natureza perturbadora [Koutkias et al. 2010].

O artigo é organizado como se segue. Na secção 2 é apresentado o cenário de uso onde pretende-se aplicar a arquitetura, a secção 3 tem o foco na explicação da arquitetura proposta e cada um de seus componentes, por ultimo na secção 4 é apresentada a conclusão e trabalhos futuros.

2. Cenário de Uso

Vários pesquisadores têm explorado o uso de agentes de software para fins de saúde tendo como principal motivação o crescente interesse em sistemas distribuídos para auxiliar médicos e pacientes, são exemplos os seguintes trabalhos [Tianfield 2003] [Nageba et al. 2007] [Chan et al. 2008]. Muitas propostas para a gestão de sistemas distribuídos estão sendo pesquisadas, mas uma parte promissora da tecnologia é a utilização de agentes [Miller e Mansingh 2013].

Muitos dos pacientes com doenças que não trazem risco de morte, mas precisam de vigilância de saúde não requerem ser hospitalizados, simplesmente requerem um acompanhamento através de um sistema móvel que tenha a capacidade de detectar anomalias, assessorar ao paciente temporalmente e mandar alertas para o médico em caso de uma emergência [Chan et al. 2008]. Com base neste contexto, será analisado o ciclo de cuidados médicos, e em seguida explicar a arquitetura proposta neste trabalho.

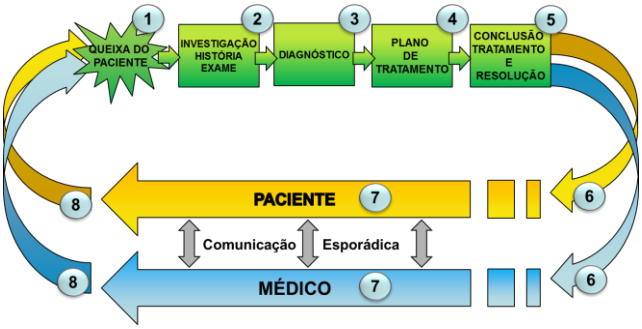


Figura 1 - Ciclo de atendimento ao paciente [Mazzi et al. 2001].

Na Figura 1, é descrito as etapas de um ciclo normal de tratamento médico: (1) paciente e médico reúnem-se, como resultado de uma queixa do paciente; (2) o médico faz uma pesquisa sobre as condições do paciente; (3) o médico diagnostica a doença com a colaboração de especialistas, se necessário; (4) o médico projeta um plano de tratamento; (5) caso o tratamento seja levado até o fim, o paciente volta para a normalidade; (6) a interação entre o paciente e o médico termina; (7) existe ocasional interação médico-paciente quando o paciente procura informações sobre o médico e o médico faz controles regulares do paciente; e (8) a interação ocasional continua até uma nova queixa do paciente e o ciclo se repete. Como é mostrado no ponto 7 (sete) da Figura 1, existe um intervalo de tempo onde o paciente e o médico deixam de ter uma relação direta e a interação entre eles é esporádica, já que só precisarão ter contato caso ocorra alguma emergência. Porém, é de vital importância o médico seguir monitorando o paciente, para saber como está sendo feito o tratamento e intervir, caso necessário. É nesta etapa do ciclo que este trabalho foi desenvolvido, com o objetivo melhorar a relação médico-paciente.

3. Arquitetura proposta

A arquitetura proposta tem como principais características ser distribuída e baseada em sistemas multiagentes, para permitir ao paciente e ao médico ter um canal de comunicação aberto em tempo integral. Para criar este canal, a arquitetura possui um ambiente com agentes que são executados em dispositivos móveis e em um servidor principal. Os agentes desse ambiente constituem uma arquitetura híbrida, pois alguns têm características reativas e outros proativas. Ou seja, alguns agentes são reativos porque suas ações dependem das ações de outros agentes e dos eventos que são gerados no contexto onde são executados. Já os agentes proativos são aqueles que suas ações são orientadas a um objetivo geral [Jennings 1999], neste caso, o de prover ao paciente um serviço de monitoramento e acompanhamento à saúde de forma integral.

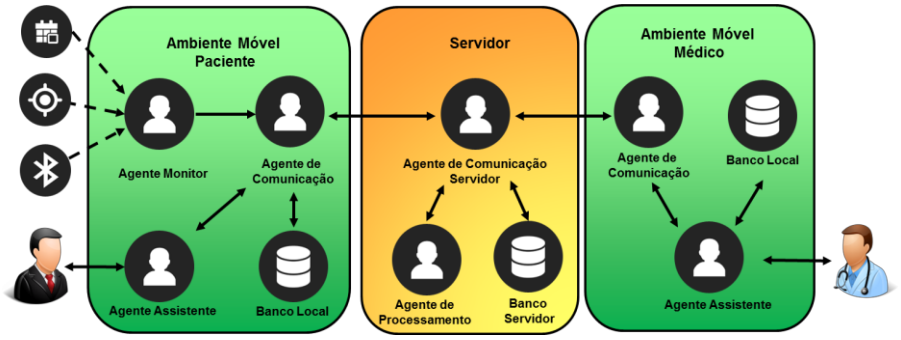


Figura 2. Ecossistema de Agentes da Arquitetura Proposta

O processo de interação entre os agentes de software e os usuários, se dá a partir de interfaces executadas em dispositivos móveis, os quais tem a capacidade de receber e prover informação para médicos e pacientes. Neste trabalho focou-se na arquitetura multiagentes e não toda a arquitetura do sistema, o qual precisa contar com a respectiva infraestrutura para obter dados gerais tanto do paciente como do médico. É assumido que este sistema irá contar com toda informação gerada em um hospital: (i) como históricos de saúde dos pacientes, (ii) lista de pacientes por médico, dentre outras informações relevantes para o tratamento. Já na Figura 2, é apresentado um diagrama no

qual é possível ver o ambiente onde os agentes interagem, para poderem cumprir com os objetivos estabelecidos no sistema. A seguir foi descrito cada um dos componentes apresentados na Figura 2.

3.1 Ambiente móvel do paciente

O ambiente móvel do paciente conta com três agentes de software e um banco de dados local. Abaixo a descrição de cada um dos componentes.

Agente Monitor. É um agente reativo que tem a função de obter a informação do contexto do paciente, e passar essa informação para o Agente de Comunicação. As tarefas do Agente Monitor são: receber informação dos sinais vitais do paciente via *bluetooth* usando dispositivos como "*Heart monitor*" [Alive Technologies 2015]; obter as informações geográficas do paciente em caso de uma emergência, e enviar alertas para o médico.

Agente Assistente. É um agente proativo, já que tem a funcionalidade de assistir ao paciente, indicando ações que se deve realizar. Para que isso seja possível, o agente fica revisando os antecedentes do paciente continuamente. Suas tarefas são: notificar os horários para tomar um remédio de acordo com as prescrições médicas, notificar datas de reuniões com o médico automaticamente e notificar instruções do médico.

Agente de Comunicação. É um agente proativo, seu papel é gerenciar a comunicação dos agentes do ambiente com os agentes do servidor. Ele monitora e analisa as informações de outros agentes, para saber se elas são relevantes para o médico. Se a informação não é relevante, estas serão salvas somente no banco local. Já se a informação é relevante, o agente além de salvar no banco local, estabelece comunicação e as envia para o servidor. Caso a comunicação não possa ser efetuada, o agente se adapta para não perder a informação e procura outro meio de comunicação até conseguir enviar as informações desejadas [Guimarães 2012]. Quando acontece do servidor enviar informações ao paciente, este agente pega a informação e caso considere relevante, passa imediatamente a informação para o agente de interface que comunicará ao paciente. Entretanto, se a informação não for relevante, o agente salva a informação no banco local, para ser aproveitada no futuro.

Banco local Paciente. É a base de conhecimento local, onde é armazenada toda a informação do perfil do paciente, além de conter toda a informação gerada tanto pelos agentes de software, como pelos humanos. Por exemplo, quando uma prescrição médica é realizada, a informação é enviada ao dispositivo móvel do paciente, e salva no banco local pelo Agente de Comunicação, em seguida o Agente Assistente pode usar essa informação.

3.2 Servidor

No servidor, a arquitetura apresenta dois agentes, além de um banco de dados geral. Agora são descritos todos os componentes do servidor.

Agente de Comunicação Servidor. É um agente proativo, seu objetivo é gerenciar a comunicação dos usuários do sistema. Este agente recebe informação do comportamento dos pacientes e instruções do médico, em tempo integral. Depois de receber e analisar a informação, o agente envia a informação direto para o destinatário, mas se a informação

precisar de algum refinamento, passa a informação para o Agente de Processamento. Caso não seja possível se comunicar com os agentes que estão nos dispositivos móveis, o agente mantém a informação no banco do servidor e espera até que a comunicação seja reestabelecida, para tentar retransmitir.

Agente de Processamento. É um agente híbrido, tanto proativo como reativo, já que pode fazer um processamento por pedido específico de outro agente, por exemplo, verificar se um paciente pode tomar um remédio ou se existe alguma restrição ao medicamento. Por outro lado é reativo porque baseado na informação obtida no banco de dados, pode enviar notificações para os agentes indicando mudanças abruptas nos sinais vitais do paciente, ou mesmo, o cadastramento de um novo membro da equipe médica. É preciso dizer que o agente de processamento poderia estar funcionando como agente de comunicação, semelhante aos dos dispositivos móveis. No entanto, se a quantidade de usuários do sistema for alta, o processamento é feito por este agente especializado, conseguindo manter uma comunicação assíncrona e sem retardo pela execução de processamentos complexos.

Banco de Dados Servidor. Onde é armazenada toda a informação dos usuários do sistema, permitindo que todos os agentes tenham acesso à informação quando necessário.

3.3 Ambiente móvel do médico

Semelhante à arquitetura do paciente, diferencia-se apenas pela ausência do Agente Monitor. Deve-se deixar claro que as instruções disponíveis pelo médico são de livre acesso por qualquer pessoa da equipe médica, uma vez que a proposta consiste em prover um serviço médico integral ao paciente, enfermeiros e outros médicos podem ajudar nesse propósito.

Agente Assistente. É um agente proativo e tem como objetivo auxiliar o médico nos cuidados aos pacientes. Para que isso seja possível, o agente cria notificações quando o paciente apresenta algum tipo de problema, por exemplo, uma mudança inesperada nos sinais vitais. O agente irá exibir um formulário para o médico enviar prescrições básicas dos passos que ele deve seguir até ser normalizado o problema.

Agente de Comunicação. É um agente proativo e seu objetivo é gerenciar a comunicação entre dispositivo móvel e servidor, já descrita anteriormente no Agente de Comunicação do paciente (Ver seção 3.1). Uma funcionalidade particular, considerada proativa, é a de verificar se um paciente está sendo atendido por um médico. Se o paciente está sendo atendido, sincroniza automaticamente a informação do banco local com a do servidor para ter disponíveis todas às informações do paciente. Entretanto, se o médico não tem relação com o paciente, o agente remove toda a informação para não ter sobrecarga dispositivo móvel.

Banco local do Médico. Tem-se uma base de conhecimento local onde é armazenada toda a informação necessária para o médico atender seus pacientes.

4. Conclusões e trabalhos futuros

Neste trabalho foi apresentada uma arquitetura baseada em sistemas multiagentes, para auxiliar a relação médico-paciente, podendo ser implementada em diferentes

dispositivos móveis. As funcionalidades da arquitetura proposta são conduzidas a ser o mais automatizado possível e tentam realizar as tarefas selecionadas de uma forma mais dinâmica e eficiente pudendo ser aplicada em diversos cenários da telemedicina como, por exemplo, o monitoramento de doenças crônicas como a diabetes. Além disso, pode-se usá-la no resgate de pessoas em áreas de risco, caso aconteça um deslizamentos de terra [Cerqueira et al. 2009], já que o agente consegue enviar a localização geográfica do dispositivo móvel, caso solicitada.

Para trabalhos futuros, será discutido a segurança em sistemas multiagentes, uma vez que as informações tanto de médicos como de pacientes é confidencial. Para isso, estabelecer mais papéis dentro no sistema, como a criação de um agente enfermeiro para ajudar nos cuidados do paciente, mas restringindo determinadas informações que são pertinentes apenas para o médico.

References

Al-Taei, M.H.A. (2005) "Telemedicine needs for multimedia and integrated services digital network (ISDN)". Computational Intelligence Methods and Applications, ICSC Congress.

Cerqueira, S. L. R. et al. Plataforma GeoRisc Engenharia da Computação Aplicada à Análise de Riscos Geo-ambientais. PUC-Rio. Rio de Janeiro. 2009

Chan, V., Ray, P., and Parameswaran, N., (2008), "Mobile and Health monitoring: an agent-based approach". Communications, IET Volume: 2, Issue: 2. p. 223-230.

Guimarães, L.F. (2012) "Um Framework para Desenvolvimento de Agentes Auto-adaptativos em Dispositivos Móveis". Departamento de Informática da PUC-Rio.

Jennings, N.R. and Wooldridge, M.J. (1999) "Agent-Oriented Software Engineering". Int. Conf. on Industrial and Engineering Applications of AI, Cairo Egypt. p. 4-10.

Koutkias, V.G., Chouvarda, I., Triantafyllidis, A., Malousi, A., Giaglis, G.D, Maglaveras, N. (2010) "A Personalized Framework for Medication Treatment Management in Chronic Care". Information Technology in Biomedicine, IEEE Transactions on Volume: 14, Issue: 2. p. 464-472.

Mazzi, C., Ganguly, P., and Ray, P. (2001) "Healthcare applications based on networked agents". EEE/IEC Int. Conf. Enterprise Networking, Applications, and Services, Atlanta, USA.

Miller, K. and Mansingh, G. (2013) "Towards a distributed mobile agent decision support system for optimal patient drug prescription". Innovative Computing Technology INTECH, Third International Conference, London.

Nageba, E., Fayn, J. and Rubel, P. (2007) "A Generic TaskDriven Multi-Agent Telemedicine System". Engineering in Medicine and Biology Society. EMBS. 29th Annual International Conference of the IEEE. Lyon, France.

Tianfield, H. (2003) "Multi-agent autonomic architecture and its application in e-medicine". Intelligent Agent Technology, IAT. IEEE/WIC International Conference.

Weiss, G. (1999). "Multiagent systems: a modern approach to distributed artificial intelligence". Cambridge: The MIT Press, 1999, 619p.

Análise sobre testes automatizados para sistemas multiagentes BDI

Martin Fabichak[1], **Jomi F. Hübner**[1]

[1]Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina – (UFSC)
Florianópolis, SC, Brasil

mfabichak@gmail.com, jomi.hubner@ufsc.br

Abstract. *Many researches have being conducted related to automated testing in multiagent systems. Some are related to testing methodologies, other regarding to automatic test generation but only a few are related to automated test tools. However, none of those involve muti-agent systems written in BDI languages. This article analyses past projects and proposes a future work of developing an automated testing framework for BDI languages.*

Resumo. *Diversos tipos de estudos já foram feitos relacionados a testes automatizados em sistemas multiagentes. Alguns relacionados a metodologias de testes, outros relacionados a geração automática de testes e poucos relacionados a ferramentas de testes. Porém, nenhum destes foram pensando em sistemas de agentes escritos em linguagens BDI. Este artigo tem como propósito analisar o que já foi feito e propor um trabalho futuro visando o desenvolvimento de um framework para testes automatizados em linguagens BDI.*

1. Introdução

Sistemas multiagentes (SMA) estão sendo explorados em diversas vertentes nas últimas duas décadas. Seja em definição de metologias [Brinkkemper 1996, Bernon et al. 2003], exploração de sua organização ou na definição mais complexa de seu funcionamento com o paradigma BDI [Bordini and Hübner 2006, Bratman 1999]. Porém, pouca atenção tem sido dada em como eles podem ser testados [Cernuzzi et al. 2005].

Testes são difíceis e consomem tempo. Frequentemente mais de 50% do custo de desenvolvimento é gasto em testes [Kit and Finzi 1995]. Ao mesmo tempo, testar é crítico para assegurar a qualidade e reduzir gastos de problemas de software. Segundo a NIST [Tassey 2002] cerca de $ 59.5 bilhões de dólares são gastos anualmente para corrigir problemas de software e mais de um terço poderia ser economizado se melhores testes fossem feitos. Utilizar testes usuais em sistemas multiagentes é uma tarefa desafiadora porque estes sistemas são distribuídos, autônomos e deliberativos [Nguyen et al. 2011]. Há problemas relacionados a interoperabilidade, comunicação e coordenação, que são funcionalidades difíceis de criar e programar [Omicini et al. 2004], e também de testar.

Dentre os poucos projetos que exploraram este tema, alguns autores como Zhang [Zhang et al. 2011], Coelho [Coelho et al. 2006] e Nguyen [Nguyen et al. 2008] publicaram trabalhos expressivos na área de testes automatizados para SMA. Porém, todos esses projetos foram desenvolvidos com a utilização de linguagens orientadas a objetos com *frameworks* orientados a agente (como o JADE) [Bellifemine et al. 1999]

e não linguagens específicas para agentes BDI (como o Jason, 2APL, entre outros) [Bordini and Hübner 2006].

Enquanto a seção 2 deste artigo detalha o estado da arte de testes em *software*, a seção 3 analisa diversos artigos e publicações sobre o tópico em sistemas multiagentes. Finalmente, a seção 4 apresentará uma proposta de um projeto nesta área.

2. Testes Automatizados

Testar um software é o processo que revela inconsistências entre o comportamento esperado e o apresentado de um sistema [Eytani et al. 2008]. Testes automatizados consistem em criar programas que testam outros programas a fim de descobrir tais inconsistências. Testes automatizados estão sendo cada vez mais utilizados como um método de teste em sistemas de software para aumentar a eficiência e eficácia do processo de teste [Runeson 2006].

Teste não são mais vistos como uma extensa fase do projeto e sim algo que permeia todo processo de desenvolvimento. A figura 1 ilustra a relação entre fases do desenvolvimento e testes, que, segundo [Myers et al. 2011], são:

1. Testes de aceitação focam em achar defeitos em requerimentos;
2. Testes de sistema focam em achar defeitos na especificação do sistema;
3. Testes de integração focam em achar inconsistências entre todas interfaces;
4. Testes unitários focam se um pedaço de código (métodos, classes ou até mesmo partes de um agente) funcionam da maneira correta.

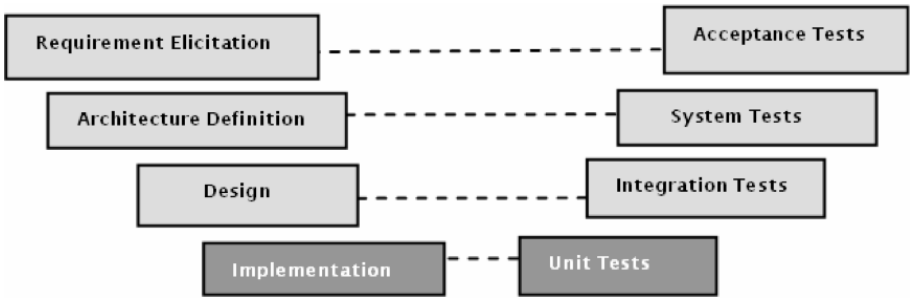


Figura 1. Relação entre diversas fases de desenvolvimento e testes em *software*. [Myers et al. 2011]

[Kaner 1988] define dois termos sempre utilizados:

- Suites de teste, que consiste em um conjunto de casos de teste e de operações que são realizadas antes e depois dos testes;
- Casos de teste, que são efetivamente um teste que será feito e faz um afirmação (*assert*) de como esse teste deveria terminar.

Muitas metodologias utilizam testes unitários como parte importante do desenvolvimento de todo projeto. Beck [Beck 2002], criador do *Extreme Programming*, diz que testes unitários é uma técnica importante para verificar a acurácia e confiança de sistemas.

Muito da literatura e divulgação de testes automatizados vieram do paradigma de orientação a objetos, das quais o paradigma de agentes pode se inspirar, mas, como resultado da sua natureza específica, novos métodos são necessários [Houhamdi 2011].

Trabalho	Nível de Teste	Principal Contribuição	Tecnologias envolvidas
[Zhang et al. 2011]	Unitário	Teste de objetivos relacionados ao eventos que o disparam; identificação da melhor ordem de execução dos testes	não especificado
[Ekinci et al. 2009]	Unitário	Teste de objetivos; criação de um *framework* que leva em conta a criação, a execução e a verificação do resultado dos testes	SEAGENT (Java)
[Coelho et al. 2006]	Agente	Criação de um *framework* baseado em JADE e JUNIT para se criar um agente *mock* [Mackinnon et al. 2001]	JADE (Java)
[Lam and Barber 2005]	Agente	Proposta de um processo semi-automatizado para compreender o comportamento de um agente, utilizando o que foi aprendido para encontrar comportamentos estranhos	múltiplos
[Serrano and Botia 2009, Botia et al. 2004]	Integração	Proposta de uma análise utilizando *data mining* de todas as comunicações feitas entre agentes, criando um grafo de comunicações e verificando posteriormente se todas foram satisfeitas. Ferramenta criada para JADE	Jade (Java)
[Padgham et al. 2005]	Integração	Criação de testes através de artefatos de design (planos e protocolos de interação) para identificação de erros em tempo real.	Jack (Java)

Tabela 1. Análise entre trabalhos relacionados

3. Testes automatizados em sistemas multiagentes

Existem alguns trabalhos relacionados a testes automatizados em SMA, nos diversos níveis apresentados na figura 1. [Nguyen et al. 2011] define um novo nível de testes relacionados a agentes e categoriza diversos trabalhos relacionados dentre os níveis e também em dois aspectos: passivo e ativo. Sendo testes passivos relativos a trabalhos que observam o *output* do sistema, geralmente tendo os dados de entrada pré-programados; Testes ativos são aqueles que analisam o *output* para gerar novos e melhores *inputs* para que os testes sejam melhorados. Já [Houhamdi 2011] analisou diversos trabalhos verificando o quão usáveis são os produtos das pesquisas, também dividindo pelos níveis de teste. Dos trabalhos estudados pelos dois autores, muitos são relacionados a metodologia de desenvolvimento (ou seja, identificar quais testes precisam ser feitos, geralmente sendo parte de metologias usuais de SMA como MASE, PROMETHEUS ou Tropos) enquanto poucos geraram ferramentas ou métodos de se testar um SMA ou partes dele. Nenhum teste compreendeu ambientes e organização [Boissier et al. 2013, Omicini et al. 2008], que, segundo [Weiss 1999], aumentam o impacto na performance de curto e longo prazo de um sistema multiagente.

Na tabela 1 estão listados alguns dos diversos trabalhos relacionados ao tema, mais próximos do projeto de mestrado apresentado na seção 4. Após a análise de tais trabalhos, é fácil de notar que não houve estudos relacionados diretamente testes automatizados em linguagens de agentes *BDI*. Alguns trabalhos são direcionados a *BDI*, e, por exemplo, a base de crenças até é utilizada para se testar os agentes, porém nunca para uma linguagem *BDI*.

4. Trabalho futuro

Trabalhos futuros serão feitos no programa de mestrado e espera-se que a contribuição acadêmica seja na criação de uma ferramenta e de métodos para testes automatizados utilizando linguagens *BDI*.

Partiremos da hipótese de que é possível definir um *script* (como no teatro) na qual todo o sistema precisa seguir para passar nos testes. Um mecanismo de testes receberá todas as comunicações de todos agentes e a percepção de todos artefatos do ambiente. Para o desenvolvimento deste *script* de teste, algumas perguntas precisam ser respondidas: como seriam tais testes? Como criar diferentes casos de testes? Como os agentes seriam testados? Como forçar em tempo real as situações descritas no *script*? Como deixar a criação e manutenção do *script* fácil o suficiente para que seja realmente produtivo utilizar o método em aplicações? Essa hipótese foi criada se pensando na facilidade de se criar testes e de executá-los. Utilizar uma maneira mais convencional como na de engenharia de *software* é a melhor opção para manter a simplicidade para o desenvolvedor.

Por exemplo, suponho uma situação onde, se um agente *a1* com papel *p1* passa a acreditar na crença *c* graças ao evento *e*. outro agente *a2* de papel *p2* também precisa ter em sua base de crenças *c*. Uma possível abordagem seria:

```
1.      agent a1(p1), a2(p2);
2.      define_event(e):
3.              message(a1, tell, c);
4.      when(e):
```

```
5.              assert ? a1 believes c;
6.              assert ? a2 believes c;
7.      trigger(e);
```

Na linha 1 ocorre a definição dos agentes com seus determinados papeis; Nas linha 2 e 3, define-se o evento *e* caracterizado pelo envio de uma mensagem para *a1* com o comando *tell* [Bellifemine et al. 1999] a crença *c*; Na linha 4 define-se o que deve acontecer quando o evento *e* ocorrer; Nas linhas 5 e 6, há o teste de fato através do comando *assert*: verifica-se se os agentes *a1* e *a2* acreditam em *c*; Na linha 8, o teste é iniciado com a execução do evento *e*.

Se a hipótese for correta e este pequeno trecho de código puder ser feito, será possível descrever diversos tipos de testes que se iniciam de diversos pontos da aplicação.

Referências

Beck (2002). *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Bellifemine, F., Poggi, A., and Rimassa, G. (1999). JADE: A FIPA-compliant agent framework. In *Proceedings of the fourth conference on the practical application of intelligent agents and multi-agent technology*, pages 97–108, London, UK.

Bernon, C., Gleizes, M.-P., Peyruqueou, S., and Picard, G. (2003). Adelfe: A methodology for adaptive multi-agent systems engineering. In *Proceedings of the 3rd International Conference on Engineering Societies in the Agents World III*, ESAW'02, pages 156–169, Berlin, Heidelberg. Springer-Verlag.

Boissier, O., Bordini, R. H., Hübner, J., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761. Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.

Bordini, R. H. and Hübner, J. F. (2006). Bdi agent programming in agentspeak using jason. In *Proceedings of the 6th International Conference on Computational Logic in Multi-Agent Systems*, CLIMA'05, pages 143–164, Berlin, Heidelberg. Springer-Verlag.

Botia, J. A., Lopez-Acosta, A., and Gomez-Skarmeta, A. F. (2004). Aclanalyser: A tool for debugging multi-agent systems. In de MÃ¡ntaras, R. L. and Saitta, L., editors, *ECAI*, pages 967–968. IOS Press.

Bratman, M. (1999). *Intention, plans, and practical reason*. The David Hume series of philosophy and cognitive sciences reissues. Center for the Study of Language and Information.

Brinkkemper, S. (1996). Method engineering: engineering of information systems development methods and tools. *Information and Software Technology*, 38(4):275–280.

Cernuzzi, L., Cossentino, M., and Zambonelli, F. (2005). Process models for agent-based development. *Eng. Appl. Artif. Intell.*, 18(2):205–222.

Coelho, R., Kulesza, U., von Staa, A., and Lucena, C. (2006). Unit testing in multi-agent systems using mock agents and aspects. In *Proc. SELMAS '06*, pages 83–90, New York, NY, USA. ACM.

Ekinci, E. E., Tiryaki, A. M., Çetin, O., and Dikenelli, O. (2009). Agent-oriented software engineering ix. pages 173–186. Springer.

Eytani, Y., Tzoref, R., and Ur, S. (2008). Experience with a concurrency bugs benchmark, validation workshop. ICSTW '08, pages 379–384. IEEE Computer Society.

Houhamdi, Z. (2011). Multi-Agent System Testing: A Survey. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 2(6).

Kaner, C. (1988). *Testing Computer Software*. TAB Books, USA.

Kit, E. and Finzi, S. (1995). *Software Testing in the Real World: Improving the Process*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Lam, D. and Barber, K. (2005). Debugging agent behavior in an implemented agent system. In *Programming Multi-Agent Systems*, volume 3346 of *Lecture Notes in Computer Science*, pages 104–125. Springer Berlin Heidelberg.

Mackinnon, T., Freeman, S., and Craig, P. (2001). Extreme programming examined. chapter Endo-testing: Unit Testing with Mock Objects, pages 287–301. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Myers, G. J., Sandler, C., and Badgett, T. (2011). *The Art of Software Testing*.

Nguyen, C. D., Perini, A., Bernon, C., Pavón, J., and Thangarajah, J. (2011). Testing in multi-agent systems. In Gleizes, M. P. and Gómez-Sanz, J. J., editors, *Agent-Oriented Software Engineering X*, pages 180–190. Springer.

Nguyen, D. C., Perini, A., and Tonella, P. (2008). A goal-oriented software testing methodology. *Agent-Oriented Software Engineering VIII*, pages 58–72.

Omicini, A., Ossowski, S., and Ricci, A. (2004). Coordination infrastructures in the engineering of multiagent systems. In *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11, chapter 14.

Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456.

Padgham, L., Winikoff, M., and Poutakidis, D. (2005). Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, 18(2):173 – 190. Agent-oriented Software Development.

Runeson, P. (2006). A survey of unit testing practices. *IEEE Softw.*, 23(4):22–29.

Serrano, E. and Botia, J. (2009). Infrastructure for forensic analysis of multi-agent systems. In Hindriks, K., Pokahr, A., and Sardina, S., editors, *Programming Multi-Agent Systems*, volume 5442 of *Lecture Notes in Computer Science*, pages 168–183. Springer.

Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology.

Weiss, G., editor (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

Zhang, Z., Thangarajah, J., and Padgham, L. (2011). Automated testing for intelligent agent systems. In *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 66–79. Springer.

Uma Proposta Híbrida baseada em Agentes e Algoritmos Genéticos para a determinação dos tempos de semáforo visando a redução da Poluição: Estudo de caso do Centro de Rio grande/RS

Míriam Blank Born[1], Diana F. Adamatti[1], Marilton Sanchotene de Aguiar[2], Weslen Schiavon de Souza[2]

[1]Centro de Ciências Computacionais (C3) – Universidade Federal do Rio Grande (FURG)
Campus Carreiros, Av. Itália km 8 – Rio Grande – RS – Brazil

[2]Programa de Pós-Graduação em Computação (PPGC) – Universidade Federal de Pelotas (UFPEL)

Campus Porto, Rua Gomes Carneiro 1, 96010-610 – Pelotas – RS - Brazil

{miriamborn,dianaada}@gmail.com, {marilton,wsdsouza}@inf.ufpel.edu.br

Abstract. *This work presents a proposal based on agentes to apply Genetic Algorithms (GA) for the optimal control of traffic lights in vehicular traffic in the city of Rio Grande / RS, as well as an analysis of the dispersion of pollutants in this scenario using the urban mobility simulator, called SUMO (Simulation of Urban Mobility).*

Resumo. *Este trabalho apresenta uma proposta, com abordagem baseada em agentes, de aplicação dos Algoritmos Genéticos (AG) para o controle otimizado de semáforos no tráfego de veículos na cidade de Rio Grande/RS, bem como uma análise da dispersão de poluentes, utilizando o simulador de mobilidade urbana SUMO (Simulation of Urban Mobility).*

1. Introdução

Atualmente, o aumento da frota de veículos nos grandes centros urbanos cresce a cada ano, causando problemas de tráfego para motoristas, pedestres e para o meio ambiente.

Segundo o Departamento Nacional de Trânsito (DENATRAN) estima-se um crescimento de aproximadamente 140% em 20 anos desta frota [DENATRAN 2014]. Neste contexto, o gerenciamento de semáforos torna-se imprescindível, sendo que os controladores de semáforos convencionais mudam de forma constante, o sistema calcula o tempo de ciclo com base na carga média de tráfego, desconsiderando a dinâmica natural do mesmo, agravando problemas de congestionamento e contribuindo para a dispersão de poluentes na atmosfera. Desta forma, o monitoramento e controle de tráfego e da dispersão de poluentes torna-se um desafio para as autoridades de transporte no mundo todo [AHMAD, *et al.* 2009].

Neste trabalho, foi utilizado como base o artigo de Ahmad, *et al.* (2009), no qual é proposto um controle otimizado de semáforo utilizando Algoritmos Genéticos (AGs) em quatro vias, uma junção e uma travessia de pedestre. Os Algoritmos Genéticos foram introduzidos neste contexto de controle de trânsito com intuito de fornecer uma

resposta inteligente de intervalo verde (tempo em que os semáforos permanecem no estado verde) baseado em dados de carga de tráfego dinâmico, buscando superar os diversos problemas dos controles convencionais de tráfego de veículos [AHMAD, *et al.* 2009].

A proposta deste artigo será desenvolver um algoritmo genético a partir da ideia dos autores citados anteriormente que simule primeiramente duas junções hipotéticas de trânsito, requisito mínimo para os cenários de testes; e, logo após as junções do centro da cidade de Rio Grande/RS onde o tráfego de veículos é intenso, buscando com este conjunto de semáforos uma aproximação da realidade de tráfego nesta região. A ferramenta SUMO será utilizada para o desenvolvimento destas simulações, e também para análise da dispersão de poluentes nesta área específica. Assim, visa a redução destes agentes poluidores através da organização dos semáforos.

O trabalho encontra-se em andamento e no presente momento está sendo desenvolvido o algoritmo genético e a integração dos dados de simulação do SUMO, para os possíveis cenários de simulações e análises.

O presente artigo foi estruturado nas seguintes seções: A seção 2 descreve as vantagens da aplicação dos AGs; posteriormente, na seção 3 são apresentadas algumas características do simulador SUMO e da Dispersão de Poluentes utilizada pela mesmo; e na seção 4 é descrita a proposta desenvolvida neste artigo. Na seção 5 estão as considerações finais do artigo.

2. Algoritmos Genéticos

Os Algoritmos Genéticos foram fundamentados por John Henry Holland e inspirados na Teoria da Evolução de Charles Darwin (1809-1882) os quais representam técnicas de busca baseadas nesta teoria. Nos AGs as variáveis de determinado problema são representadas como genes em um cromossomo, também pode ser denominado indivíduo [GUIMARÃES, RAMALHO 2012].

Dentre algumas características dos AGs, enquanto paradigmas da Computação Evolutiva pode-se destacar:

- Apresentam resultados satisfatórios com relação à precisão e recursos empregados, podem ser implantados em computadores domésticos, para diversos problemas de difícil solução;
- São bastante flexíveis e permitem fácil hibridização com diversas técnicas não relacionadas à Computação Evolutiva, sendo o paradigma mais utilizado e mais completo nesta área;
- Exigem menor conhecimento do problema específico para seu funcionamento, se comparados com outros paradigmas da Computação Evolutiva, estas características os tornam versáteis.

3. SUMO e Dispersão de Poluentes

O SUMO (*Simulation of Urban MObility*) é um simulador microscópico de trânsito, que foi desenvolvido em 2001 pelo Centro Aeroespacial Alemão (DLR), com o intuito de auxiliar a comunidade de pesquisa de tráfego com uma ferramenta onde algoritmos pudessem ser implementados e avaliados, sem a necessidade de obter um tráfego completo de simulação.

Possui código aberto, é portável e projetado para simular modelagens de redes rodoviárias de grande e pequeno porte, onde cada veículo pode ser considerado um agente, aos quais buscam minimizar a dispersão de poluentes a partir da organização dos semáforos. Por estas razões o simulador foi escolhido para o desenvolvimento deste trabalho [SUMO 2012].

Dentre as principais características de SUMO destacam-se: possui todas as aplicações necessárias para simular uma rede de tráfego; diferentes tipos de veículos com movimentos em espaço contínuo e tempo discreto; ruas com múltiplas faixas e mudança de faixa; interface gráfica para os usuários; e interoperabilidade com demais aplicações em tempo de execução.

O simulador possui em sua configuração definições tais como: fundo de ambiente, ruas, nodos, veículos, entre outros. A partir das definições de veículos com relação às cores destes, é possível selecionar como definição para a simulação, por exemplo, emissões de CO2, CO, NOx e outros [BEHRISCH, *et. al.* 2011]. Desta forma, os veículos mudam as cores, nas quais pode-se caracterizar como emissão de poluentes para simular uma determinada rede de estradas, no caso deste trabalho uma parte do centro da cidade de Rio Grande e medir qual a emissão de poluentes de determinado número de veículos em um tempo específico.

Por padrão, o SUMO é capaz de simular as emissões de poluentes veiculares com base no banco de dados da HBEFA (*Handbook Emission Factors for Road Transport*). Este órgão fornece fatores de emissão de poluentes para todas as categorias de veículos. Os poluentes escolhidos para serem implementados neste simulador são: CO2, CO, HC, NOx, PMX e Consumo de combustível [HBEFA 2008]. A Figura 1 representa a tela inicial do simulador SUMO e os poluentes inseridos na ferramenta.

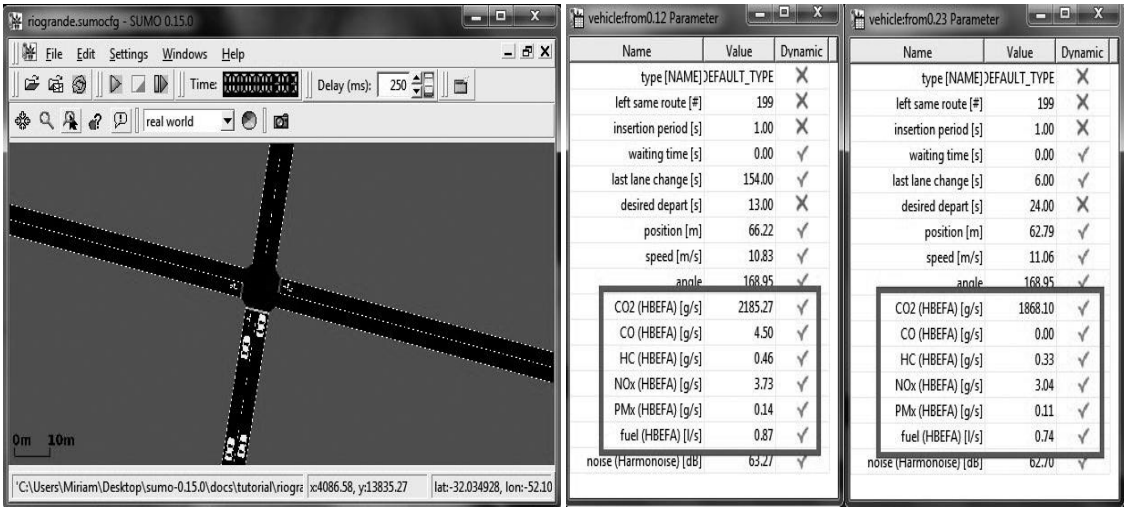


Figura 1. Representação de junção com semáforo e poluentes no SUMO

4. Proposta

A proposta do presente artigo, tomando como base o trabalho Ahmad et al. (2009), propõe estender a aplicação do algoritmo, em um primeiro momento, para duas junções de tráfego, como apresentado na Figura 2, e posteriormente para os demais semáforos do centro da cidade de Rio Grande/RS, nos modos estático e dinâmico de um AG. Para o desenvolvimento deste trabalho utilizou-se o modelo do algoritmo do trabalho referenciado, sendo que a linguagem definida para a implementação foi *Java*, visto que dentre as suas diversas características, a independência de plataforma e a portabilidade são requisitos interessantes no auxílio do problema em questão.

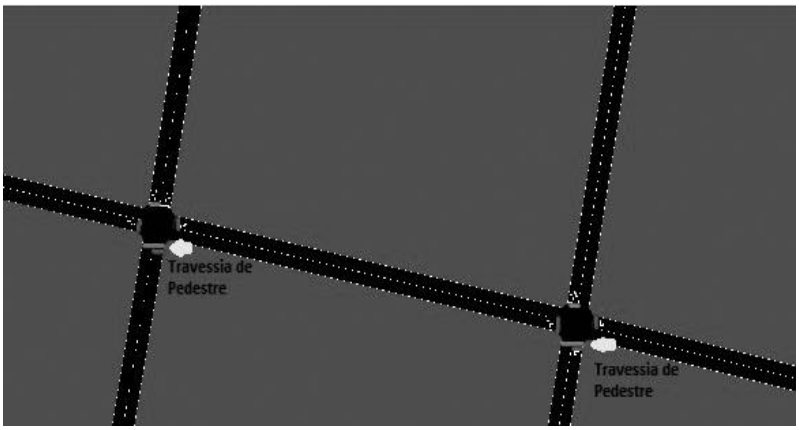


Figura 2. Tráfego com duas junções

A Figura 3 apresenta o fluxograma da proposta deste trabalho. Todo o fluxo é dividido em três etapas principais: i) geração do dados no SUMO e implementação do Algoritmo Genético; ii) simulações; e, iii) análise de resultados.

A primeira etapa, *geração de dados no SUMO e implementação do Algoritmo Genético*, subdivide-se em quatro blocos:

- **Gerar simulação no SUMO**: a partir do mapa da cidade de Rio Grande/RS, gerada através da ferramenta *OpenStreetMap* na linguagem XML (*Extensible Markup Language*), com ambiente previamente configurado de tempos dos semáforos é realizada a execução das simulações.
- **Capturar os tempos de semáforos no SUMO**: os tempos de semáforos no simulador SUMO são capturados através da TraCI (*Traffic Control Interface*) a qual fornece métodos que permitem aos desenvolvedores capturar dados e intervir na simulação em tempo de execução. A comunicação é realizada pelo protocolo TCP/IP em uma porta configurada no ambiente de simulação.
- **Configurar os tempos de semáforos no AG**: com a leitura do arquivo contendo os dados dos tempos de semáforos da simulação gerada, configura-se no AG estes tempos.
- **Realizar operação do AG e escolher os indivíduos com melhor *fitness***: a partir do arquivo de dados, com os tempos de semáforos, realiza-se as operações de seleção e mutação e assim escolhe-se os melhores indivíduos, ou seja, aqueles

que têm o valor de *fitness* (função de avaliação do AG) considerado aceitável para este caso.

A etapa de Simulações, consiste em três blocos descritos como seguem:

- **Arquivo de entrada com os melhores indivíduos**: arquivo gerado a partir das simulações realizadas com o SUMO, considerando os indivíduos que obtiveram melhor valor de *fitness*, neste caso menor índice de poluição.
- **Gerar novas simulações no SUMO**: as novas simulações são geradas considerando os *n* ciclos a serem executados conforme especificação do Algoritmo Genético.
- **Comparar os níveis de poluição (*fitness*)**: a partir dos cenários simulados será realizada uma comparação, buscando os melhores resultados nos níveis de poluição, neste caso os menores índices serão considerados os melhores, visto que o principal objetivo deste trabalho é gerenciar os tempos de semáforos de maneira a minimizar a dispersão de poluentes emitidos pelos veículos.

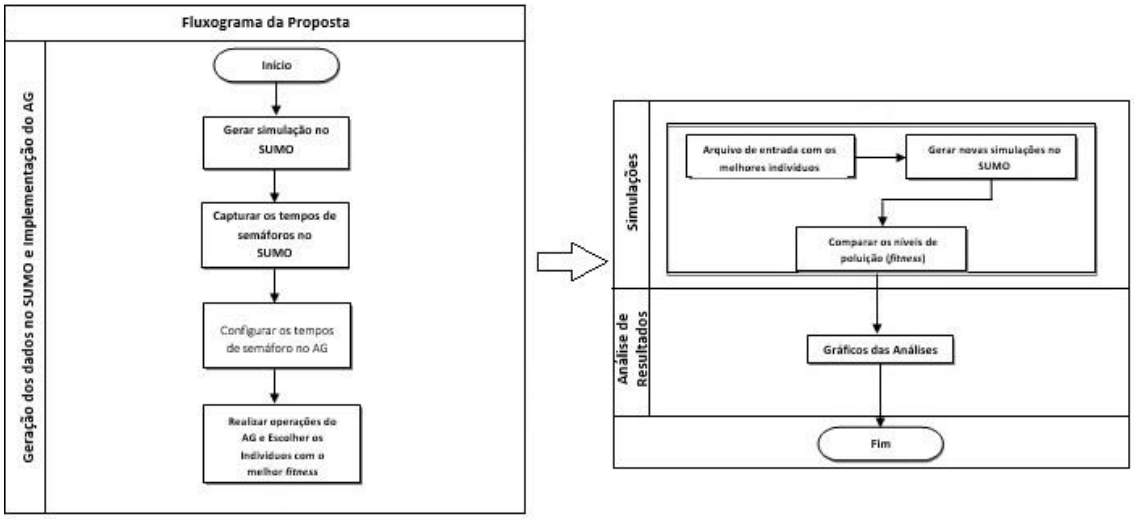


Figura 3. Fluxograma da proposta

Após a conclusão das duas primeiras etapas, Geração de dados no SUMO e implementação do Algoritmo Genético e Simulações, será realizada a etapa de Análise dos Resultados obtidos nas simulações desenvolvidas no trabalho.

A geração de relatórios e gráficos a partir dos resultados obtidos ao final das simulações propostas neste artigo, serão importantes para que possamos avaliar o grau de satisfação alcançado com a utilização dos Algoritmos Genéticos. Neste caso, a função de avaliação (*fitness*) consiste em representar o nível de poluição gerada pelos veículos na área especificada com determinada configuração dos semáforos.

5. Considerações Finais

O artigo [AHMAD, *et al.* 2009] utilizado como referência para este trabalho mostrou a comparação dos resultados das simulações realizadas com o Algoritmo Genético (modo dinâmico) e modo estático, como representado no mesmo pode-se concluir que a utilização de um AG melhora consideravelmente o tempo de espera em semáforos.

Desta forma, busca-se para este trabalho melhorar o tempo de espera, visando minimizar os efeitos da dispersão de poluentes produzidos pelos veículos.

A proposta deste trabalho visa contribuir de forma que, pesquisadores e estudantes, a partir de abordagens baseada em agentes desenvolvam modelos de tráfego próximos do mundo real. Ou seja, estender a modelagem proposta à realidade do país ou de cada região, contribuindo para solução de controladores de tráfego e a emissão de poluentes no meio ambiente.

Referências

Ahmad, M.S., Turky, A.M. and Yusoff, M.Z. (2009) "The Use of Genetic Algorithm for Traffic Light and Pedestrian Crossing Control", Malaysia.

Behrisch M.; Bieker L.; Erdmann J.; Krajzewicz D. (2011) "SUMO – Simulation of Urban MObility", http://www.sumo.sourceforge.net/pdf/simul_2011_3_40_50150.pdf, April.

DENATRAN – "Departamento Nacional de Trânsito" (2014), hhtp://denatran.gov.br, January.

Guimarães, F.G. and Ramalho, M.C. "Implementação de um Algoritmo Genético" (2012), http://www.cpdee.ufmg.br/~lusoba/disciplinas/ele037/t1.pdf, December.

HBEFA – "Handbook Emission Factors for Road Transport" (2008), http://www.hbefa.net/e/index.html, December.

SUMO – "Simulation of Urban Mobility" (2012), http://www.sumo.sourceforge.net, December.

Verifying the behavior of agents in BDI4JADE with AspectJ

Francisco J. P. Cunha[1], Marx Leles Viana[1], Márcio R. Rosemberg[1], Carlos J. P. de Lucena[1]

[1]Departamento de Informática – Pontifícia Universidade Católica (PUC-RJ)
Rua Marquês de São Vicente, 255 – 22.451-900 – Rio de Janeiro – RJ – Brasil

{fcunha,mleles,mrosemberg,lucena}@inf.puc-rio.br

Abstract. *The use of multi-agent systems (MAS) for software construction is a promising approach and has been applied in different research areas. So the validation of these systems is crucial to build robust software. However, the methods proposed focused their efforts mainly in how to analyze, to design and to implement a MAS and little attention has been given to how such systems can be tested. Furthermore, some issues related to the controllability and observability makes difficult the verification of behavior. This paper presents an approach to verify the behavior of agents based on the combination and adaptation of ideas already supported by the literature.*

Resumo. *A utilização de sistemas multiagentes (SMA) para construção de software é uma abordagem promissora e tem sido aplicada em diferentes áreas de pesquisa. Assim, a validação destes sistemas é crucial para construção de softwares robustos. No entanto, os métodos propostos concentraram seus esforços para analisar, projetar e implementar SMA, não dando atenção necessária para a forma como tais sistemas podem ser testados. Além disso, questões relacionadas à controlabilidade e observabilidade dificultam a verificação do comportamento. Este artigo apresenta uma abordagem para verificação do comportamento de agentes baseada na combinação e adaptação de ideias já suportadas pela literatura.*

1. Introdução

Com o crescimento da web, sistemas complexos se tornaram uma realidade. Estes são caracterizados por serem distribuídos e composto de entidades autônomas que interagem entre si. Sistemas multiagentes (SMA) são sociedades nas quais entidades autônomas (agentes), heterogêneas e projetadas individualmente, trabalham em função de objetivos que podem ser comuns ou diferentes [López 2003]. Assim, a utilização de agentes para construção de tais sistemas complexos é considerada uma abordagem promissora [Zambonelli et al. 2001]. Com base nestes aspectos, o uso de SMA vem sendo cada vez mais aplicado em diferentes áreas, com diferentes níveis de autonomia, indo desde sistemas quase totalmente controlados com a ajuda de intervenção humana até aqueles quase totalmente automatizados, ou seja, com o mínimo de intervenção humana, o que significa que analisar as escolhas que este tipo de software pode fazer torna-se crucial [Fisher et al. 2013].

No que se refere ao teste de sistemas de agentes de software, algumas questões relacionadas à controlabildade e à observabilidade precisam ser cuidadosamente consideradas, pois: (i) um agente é uma entidade autônoma e, consequentemente, pode ser difícil controlar seu comportamento; (ii) as crenças e objetivos do agente estão embutidos no próprio agente podendo não ser facilmente observados e controlados; (iii) sem a adoção de estratégias eficientes para cobertura dos testes, o teste certamente se torna não escalável, dado o número de possibilidades a serem testadas [Binder 1999] e [Voas e Miller 1995].

Este artigo apresenta uma abordagem que permite a verificação do comportamento de agentes BDI [Rao e Georgeff 1995] desenvolvidos em BDI4JADE [Nunes et al. 2011]. Tal abordagem se baseada na combinação e adaptação de ideias já suportadas pela literatura de testes em agentes, em especial, do JAT *Framework* [Coelho et al. 2007] e no modelo de faltas proposto por Zhang [Zhang 2009].

2. Motivação

Apesar do uso crescente de sistemas multiagentes em cenários críticos, as metodologias propostas até o momento pela Engenharia de Software Orientada a Agentes (AOSE) concentraram seus esforços, principalmente no desenvolvimento de abordagens disciplinadas para analisar, projetar e codificar tais sistemas. No entanto, pouca atenção tem sido empregada na forma como tais sistemas poderiam ser testados [Caire et al. 2004].

Sendo assim, diante da necessidade de verificar e compreender o comportamento complexo executado pelo agente, avaliar sua eficácia e do desafio e implicações referentes à testabilidade dos agentes de software, este trabalho se concentra na tarefa de apresentar uma abordagem que permita apoiar o desenvolvedor de agentes na construção de casos de teste para agentes BDI.

As seguintes questões surgem como motivação para este trabalho: "*Como podemos apoiar o desenvolvimento de sistemas multiagentes através da construção e manutenção de casos de testes para agentes BDI?*" e "*Como podemos controlar e observar as ações executadas durante o ciclo de raciocínio de agentes BDI?*".

3. Trabalhos Relacionados

Embora existam trabalhos que abordam e apresentam estratégias para o teste de agentes BDI nenhum deles se preocupa em fornecer mecanismos que auxilie na identificação de falhas de implementação e na observação do estado interno dos elementos do agente (*beliefs*, *goals*, *plans*, *events*, etc.) durante o processo de desenvolvimento do agente.

[Winikoff e Cranefield 2010] apresentam uma análise da flexibilidade e das características adaptativas dos agentes BDI observando o espaço comportamental do agente, ou seja, o número de caminhos possíveis para alcançar um objetivo. O trabalho buscou entender a viabilidade de assegurar a eficácia dos sistemas multiagentes através dos testes. Para isso, relacionaram a viabilidade do teste de um sistema multiagente à proporção de caminhos que podem ser percorridos do espaço comportamental, considerando ainda que, executar um teste consiste em observar um caminho de execução e determinar se este está correto ou não. Em última análise, os autores concluem que testar o espaço comportamental de um sistema multiagente como um

todo é inviável dado o seu comportamento assintótico. A preocupação apresentada neste trabalho não está em verificar se um determinado caminho está correto, nem em fornecer mecanismos para identificação de falhas, mas em determinar se é possível garantir a eficácia do sistema por meio de testes. As conclusões de Winikoff e Cranefield sobre a viabilidade do teste de sistemas multiagentes corroboraram para a decisão de propor uma abordagem que considere o teste unitário de agentes.

No trabalho [Coelho et al. 2007] apresenta o JAT (Jade Agent Testing Framework), um *framework*, capaz de criar testes para sistemas multiagentes escritos em JADE baseado na utilização de "agentes mock" [Coelho et al. 2006], ou seja, na implementação "falsa" de um agente real com o propósito exclusivo de testar a comunicação entre os agentes. Através do monitoramento do estado interno dos agentes é possível controlar e observar a interação entre os agentes mock e o agente em teste. Apesar da boa contribuição no que tange ao teste de agentes de software, este trabalho se limita a testar agentes de comportamentos reativos onde são verificadas, basicamente, falhas no protocolo de comunicação entre os agentes.

Em [Zhang et al. 2009] é apresentado um *framework* para geração automática de casos de testes para sistemas multiagentes. Este trabalho considera a construção de sistemas multiagentes baseados em modelos [Apfelbaum e Doyle 1997] [El-Far e Whittaker 2001]. É apresentado, ainda, um modelo de faltas para identificar possíveis falhas e as condições em que as mesmas podem ocorrer. Este modelo de faltas foi utilizado pela abordagem descrita neste artigo.

4. Uma abordagem para testes de agentes BDI4JADE

Esta seção apresenta uma visão geral da abordagem proposta, apoiada nos trabalhos de [Coelho et al. 2007] e [Zhang et al. 2009].

4.1. Visão geral da abordagem

Cada agente possui sua própria "*thread*" de execução e seu comportamento é determinado por um conjunto de inferências feitas mediante suas crenças, objetivos e planos [Garcia et al. 2004]. Com o objetivo de testar efetivamente o agente – considerando que os testes são baseados em alguma forma de comparação do resultado esperado com o resultado produzido – é necessário saber como cada um dos elementos do agente se comportou durante sua execução. Portanto, a informação sobre o estado das crenças, os planos que foram selecionados e abandonados, os objetivos, as mensagens trocadas e os eventos disparados são cruciais para verificar se o agente executou conforme esperado.

A ideia de criar estruturas capazes de manter as informações sobre as transições dos estados do agente [Coelho et al. 2007], motivou, de forma análoga, a criação de um conjunto de estruturas capazes de armazenar as informações dos elementos dos agentes, ocorridas durante a execução. Sendo assim, é possível consultar posteriormente as estruturas preenchidas, e verificar se se estas se comportaram como esperado e identificar a ocorrência de possíveis falhas de acordo com [Zhang et al. 2009].

4.2. Monitorando as informações do ciclo de raciocínio com ASPECTJ

Para obter esse resultado, deve-se adicionar código (nos agentes e plataformas envolvidas, neste caso, o BDI4JADE) nos pontos onde ocorrem alterações nas crenças,

planos, troca de mensagens e eventos assim como nos locais onde são tomadas decisões importantes do mecanismo deliberativo.

Fazendo assim, no entanto, o código adicionado estaria espalhado em muitos pontos e por muitos módulos da plataforma. Logo, monitorar o ciclo de raciocínio do agente é, naturalmente, um interesse transversal. Nestes casos, uma solução amplamente adotada é definir um aspecto para apontar diretamente os locais de execução no agente e na plataforma que registram as transições no estado dos componentes [Briand et al. 2005]. O monitoramento do ciclo de raciocínio do agente e da plataforma BDI4JADE é realizado através de um aspecto, implementado na linguagem ASPECTJ, conforme pode ser visto na Figura 1.

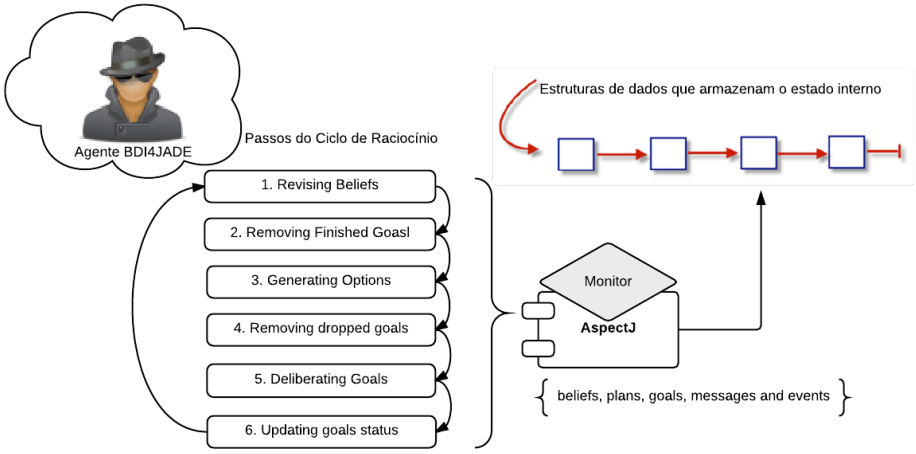


Figura 1. Armazenando as informações dos agentes durante a execução.

Cada passo do ciclo do raciocínio é responsável por uma etapa do processo deliberativo. Sendo assim, durante sua execução são criadas estruturas associadas a cada etapa do ciclo de raciocínio, capazes de armazenar as informações e transições ocorridas na execução da etapa. Por exemplo, durante a etapa de revisão das crenças, executada no início do ciclo de raciocínio do agente são criadas estruturas capazes de armazenar as crenças iniciadas com a criação do agente, as crenças que foram removidas durante a execução do agente, as crenças que tiveram seu valor alterado e assim por diante. Após a execução do agente as estruturas estão preenchidas e podem ser consultadas. Dessa forma, o testador pode consultar o conteúdo dessas estruturas e verificar se um comportamento ocorreu conforme esperado (se o valor de uma crença foi alterado, por exemplo).

O conteúdo das estruturas pode ser consultado através de métodos assertivos facilitadores no estilo JUNIT. Um exemplo de um método assertivo que permite ao desenvolvedor verificar a alteração no valor de uma crença é: *assertWasUpdatedBelief*. Assim, através de um conjunto de métodos assertivos capazes de consultar as informações e transições dos elementos do agente ocorridas durante o ciclo de raciocínio é possível, por meio de casos de testes definidos pelo desenvolvedor do agente, verificar o comportamento do agente e identificar possíveis condições de falhas.

5. Conclusão

Este artigo propôs uma abordagem para verificação do comportamento de agentes BDI desenvolvidos em BDI4JADE capaz de apoiar o desenvolvimento de agentes de software através da construção e manutenção de casos de testes. Tal abordagem apoiou-se nas ideias suportadas por [Coelho et al. 2007] e no modelo de faltas proposto por [Zhang 2009].

Como trabalhos futuros, vemos a necessidade de implementar uma ferramenta adotando a abordagem proposta e também executar um conjunto de cenários de uso como prova de conceito da abordagem. Em seguida, um experimento controlado pode ser realizado para obter junto a comunidade de desenvolvedores de agentes um *feedback* do uso da ferramenta. Ainda como trabalho futuro, uma análise desta abordagem pode ser aplicada em outras plataformas de desenvolvimento de agentes tais como: JASON [Bordini et al. 2007], JADEX [Braubach et al. 2003]e JACK [Howden et al. 2001], por exemplo. Procurando entender suas implicações e caso seja necessário, adaptações podem ser realizadas.

References

Apfelbaum, L. and Doyle, J. (1997) "Model Based Testing. International Software Quality Week Conference". CA – USA.

Binder, R (1999) "Testing Object-Oriented Systems: Models, Patterns, and Tools", Addison-Wesley, 1999.

Bordini, R. H., Hübner, J. F. and Wooldridge, M. (2007) "Programming Multi-Agent Systems in AgentSpeak using Jason", In Wiley Series in Agent Technology.

Braubach, L., Lamersdorf, W., and Pokahr, A. (2003) "Jadex: Implementing a BDI-infrastructure for JADE agents".

Briand, L., Labiche, Y. and Leduc, J. (2005) "Tracing Distributed Systems Executions Using AspectJ", In: Proceedings of ICSM.

Caire, G., Cossentino, M., Negri, A., Poggi, A. and Turci, P. (2004) "Multi-agent systems implementation and testing", In Proceedings of 4th International Symposium - From Agent Theory to Agent Implementation.

Coelho, R, Cirilo, E., Kulesza, U., Staa, A., Rashid A. and Lucena, C. (2007) "JAT: A Test Automation Framework for MultiAgent Systems", In International Conference on Software Maintenance. ICSM.

Coelho, R., Kulesza, U., Staa, A. and Lucena, C. (2006) "Unit Testing in Multi-agent Systems using Mock Agents and Aspects", In International Workshop on Software Engineering for Large-Scale Multi-Agent Systems. ICSE.

El-Far, I. and Whittaker, J. (2001) "Model-Based Software Testing", In Encyclopedia of Software Engineering, pages 825-837. Wiley, Chichester.

Fisher, M., Dennis, L. and Webster, M. (2013) "Verifying Autonomous Systems", In Communications of the ACM, Vol. 56 No. 9, Pages 84-93.

Garcia, A., Lucena, C. and Cowan, D. (2004) "Agents in Object-Oriented Software Engineering. Software Practice & Experience", Elsevier, 34(5), pages 489-521.

Howden, N., Rönnquist, R., Hodgson, A., & Lucas, A. (2001, May). “JACK intelligent agents-summary of an agent infrastructure”. In 5th International conference on autonomous agents.

López, F. L. (2003) “Social Power and Norms”.

Nunes, I., Lucena, C. and Luck, M. (2011) “BDI4JADE: a BDI layer on top of JADE”, In: International Workshop on Programming Multi-Agent Systems - ProMAS.

Pokahr, A., Braubach L. and Lamersdorf W. (2005) “Jadex: A BDI Reasoning Engine”, Multi-Agent Programming - Springer US.

Rao, A. and Georgeff, M. (1995) “BDI-agents: from theory to practice”, In: Proceedings of the First International Conference on Multiagent Systems.

Voas, J. and Miller, K. (1995) “Software Testability: The New Verication”, In: IEEE Software, 1995.

Winikoff, M. (2005) “Jack™ Intelligent Agents: An Industrial Strength Platform”, In: Multi-Agent Programming - Springer US.

Winikoff, M. and Cranefield, S. (2010) “On the testability of BDI agents”, In: European Workshop on Multi-Agent Systems.

Zambonelli, F., Jennings, N., Omicini, A. and Wooldridge, M. (2001) “Agent-oriented software engineering for internet applications”, Coordination of Internet Agents, p. 326–346. Springer Verlag.

Zhang, Z., Thangarajah, J. and Padgham, L. (2009) “Model based testing for agent systems”, In: International Conference on Autonomous Agents and Multiagent Systems.