Relatório Técnico do INE

Revisão Bibliográfica e Modelo Conceitual de Execução de Agentes BDI para Análise de Complexidade Computacional no Paradigma de Agentes

Gabriel Campos Albino e Maicon Rafael Zatelli

Relatório Técnico INE 002/2021

UFSC

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

Resumo

Analisar um algoritmo envolve pensar quais e quanto dos recursos computacionais serão necessários para a sua execução (ex: a quantidade de tempo (CPU) e espaço (memória)), os quais variam de acordo com o tamanho e dados da entrada. Com a introdução de arquiteturas de *hardware*, como arquiteturas multi-core, e o advento de redes e sistemas distribuídos, avaliar a eficiência de aplicações baseadas no paradigma de agentes torna-se ainda mais importante.

Estabelecer métricas de eficiência em agentes é um tema de crescente preocupação uma vez que o uso de agentes vem tornando-se cada vez mais comum, porém, frequentemente, as métricas que são utilizadas para avaliar um agente são baseadas em características temporais (ex: tempo de execução) ou em relação à troca de mensagens. Além de não serem suficientes para avaliar a complexidade computacional de um programa de agente, elas dependem da plataforma de execução e da infraestrutura de *hardware* onde o agente é executado e avaliado. Também não é possível estimar o tempo de execução sem que o sistema seja efetivamente executado, o que pode ser problemático quando pensa-se na realização de experimentos.

O objetivo desta pesquisa é realizar um estudo e propor um modelo e técnica para cálculo da complexidade computacional de um programa de agente, considerando sempre as características e conceitos inerentes ao paradigma de agentes.

Palavras Chaves: agentes, complexidade computacional, BDI

Sumário

1 Introdução

Analisar um algoritmo envolve pensar quais e quanto dos recursos computacionais serão necessários para a sua execução (ex: a quantidade de tempo (CPU) e espaço (memória)), os quais variam de acordo com o tamanho e dados da entrada. Com a introdução de arquiteturas de *hardware*, como arquiteturas multi-core, e o advento de redes e sistemas distribuídos, avaliar a eficiência de aplicações baseadas no paradigma de agentes torna-se ainda mais importante. Para uma avaliação precisa de eficiência é necessário considerar uma métrica que seja independente de plataforma de execução[1], infraestrutura de *hardware*, e de instância de execução. Nesta direção, a complexidade computacional já é uma métrica usada para a análise de algoritmos e funciona muito bem na programação estruturada e orientada a objetos. A complexidade computacional de um algoritmo, por exemplo, pode ser comparada com a complexidade computacional de outro algoritmo a fim de facilitar a tomada de decisões nas fases de projeto e desenvolvimento de um programa [Kleinberg e Tardos, 2005]. No entanto, este assunto ainda merece uma investigação mais profunda no paradigma de agentes. Atualmente não existem abordagens que permitam avaliar a complexidade computacional de um programa de agente, considerando seus próprios conceitos e características.

Estabelecer métricas de eficiência em agentes é um tema de crescente preocupação uma vez que o uso de agentes vem tornando-se cada vez mais comum [Magary, 2003, Van Bien et al., 2010a]. Frequentemente, as métricas que são utilizadas para avaliar um agente são definidas para cada agente individualmente e baseadas em características temporais (ex: tempo de execução) ou em relação à troca de mensagens [Hmida et al., 2008]. Alguns trabalhos já permitem realizar *profiling* em agentes, assim fornecendo diversas informações em tempo de execução [Helsinger et al., 2003, Van Bien, 2008, Nagwani, 2009]. Porém, tais métricas de eficiência e *profiling* não são suficientes para avaliar a complexidade computacional de um programa de agente, visto que muitas delas dependem da plataforma de execução e da infraestrutura de *hardware* onde o agente é executado e avaliado. Isto pode ser uma limitação principalmente em um ambiente industrial, onde podem existir máquinas diferentes trabalhando com um mesmo programa de agente, e que eventualmente possuem uma configuração de *hardware* diferente de onde o agente foi desenvolvido e testado. Também não é possível estimar o tempo de execução sem que o sistema seja efetivamente executado, o que pode ser problemático quando pensa-se na realização de experimentos. Assim, faz-se necessário também um estudo para definir como a complexidade computacional de um programa de agente

[1] A plataforma de execução contempla o interpretador de uma linguagem e outros mecanismos e componentes de *software* necessários para a execução de um agente.

deve ser calculada, considerando a fase de projeto e desenvolvimento de um agente.

Muitas pesquisas recentes em agentes tem se preocupado especialmente com questões funcionais como coordenação, racionalidade e modelagem do conhecimento. Propriedades não funcionais de um agente, que incluem questões de desempenho, escalabilidade e estabilidade (em casos de um sistema multiagente (SMA)), assim como problemas básicos de computação que suportam o desenvolvimento de agentes, não tem recebido uma grande atenção da comunidade de agentes [Lee et al., 1998, Wooldridge, 2000, Helsinger et al., 2003, Camacho e Aler, 2005, Ajitha et al., 2009, Van Bien, 2008, Van Bien et al., 2010b, Van Bien et al., 2010a]. Por exemplo, é importante levar em consideração como questões de desempenho, escalabilidade e estabilidade podem implicar nas decisões de projeto e como as mesmas devem ser avaliadas para melhorar o projeto de um SMA [Lee et al., 1998].

A avaliação da complexidade computacional de um programa de agente não é uma tarefa simples, devido a algumas características naturais de um agente, onde muitas não são naturalmente suportadas pelas técnicas atuais de complexidade computacional [Wooldridge e Jennings, 1999, Di Bitonto et al., 2010, Wooldridge e Dunne, 2005].

- agentes são inerentemente imprevisíveis devido a sua natureza autônoma;
- padrões e efeitos de interação entre agentes são incertos e muitas vezes decididos em tempo de execução;
- agentes normalmente são aplicações multi-thread e possivelmente distribuídas;
- agentes podem possuir tipos diferentes de metas;
- as informações que os agentes podem ter em sua base de crenças podem ser incorretas ou não atualizadas, e os agentes podem deliberadamente ou acidentalmente trocar informações falsas;
- eventos e mudanças do ambiente podem ocorrer de forma imprevisível, bem como trocas de mensagem;
- agentes podem entrar ou sair de um SMA em tempo de execução (sistemas abertos);
- informações que um agente possui não são diretamente acessíveis por outros agentes, bem como outras estruturas e dados internos (ex: os planos para atingir uma meta).

Outras características que podem ou devem ser lembradas quando pensa-se em calcular a complexidade computacional de um agente são o ciclo de raciocínio, a arquitetura do agente e suas estruturas de dados [Lee et al., 1998].

Além disso, é necessário o conhecimento de técnicas de análise de algoritmos adotadas tanto em sistemas distribuídos e concorrentes, bem como em linguagens lógicas, visto que linguagens de agentes muitas vezes buscam inspiração em linguagens lógicas, e em técnicas de planejamento (*planning*). Assim, dadas as características mencionadas acima, pode-se verificar que a tarefa de calcular a complexidade computacional de um programa de agente não é trivial, mas possui relevância científica para a comunidade de agentes. As técnicas atuais de complexidade computacional não poderiam ser aplicadas de maneira natural (isto é, considerando seu estado atual) no paradigma de agentes, visto que não contemplam muitas das características mencionadas. O objetivo desta pesquisa é realizar um estudo e propor um modelo e técnica para cálculo da complexidade computacional de um programa de agente, considerando sempre as características e conceitos inerentes ao paradigma de agentes.

Este documento apresenta o estado atual da pesquisa e está estruturado como segue. A seção 2 apresenta alguns conceitos básicos relacionados à agentes enquanto que uma breve revisão da literatura também é realizada e apresentada na seção 3. A seguir, são apresentados a justificativa (seção 4) para o desenvolvimento da pesquisa, bem como os objetivos da pesquisa (seção 5). A seção 6 apresenta um modelo de execução de um agente BDI e, por fim, são apresentados alguns resultados já obtidos assim como trabalhos futuros (seção 7).

2 Agentes

Um agente é um sistema computacional situado em um ambiente e capaz de agir autonomamente neste ambiente para cumprir suas metas de projeto. O agente recebe dados do ambiente por meio de sensores e produz, como saída, ações que afetam o ambiente. Um agente difere de outros modelos baseados em objetos porque tem propriedades que não satisfazem as condições para um agente pertencer a classe de simples objetos [Wooldridge, 2002, Wooldridge, 1995]. Um agente deve possuir as seguintes propriedades:

- autonomia: a autonomia é a capacidade de um agente operar de forma independente, a fim de alcançar as metas que foram-lhe delegadas. No mínimo, um agente autônomo toma decisões sobre como atingir os objetivos impostos a ele;
- proatividade: significa exibir um comportamento direcionado para a meta. Se uma meta foi delegada a um agente, este tenta tomar ações que visem alcançá-la;
- reatividade: significa ser sensível às mudanças no ambiente, ou seja, é a capacidade que o agente possui de responder a estímulos do ambiente de uma forma reflexiva;

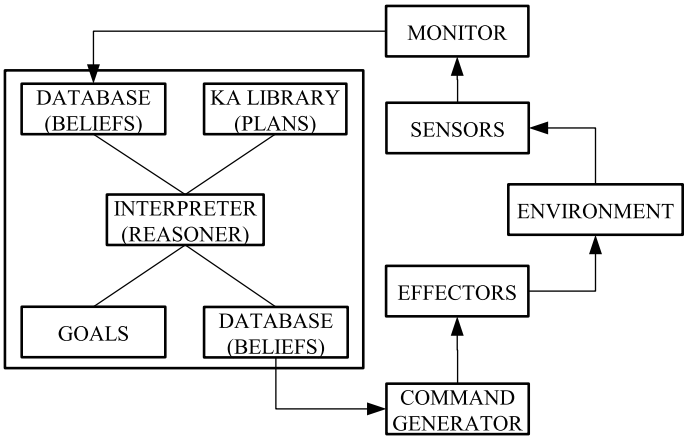

Figura 1: *Procedural Reasoning Cycle [Georgeff e Ingrand, 1989].*

- habilidade social: é a capacidade que os agentes possuem para cooperar e coordenar as atividades com outros agentes, a fim de alcançar os objetivos. Ela não visa somente a troca de *bytes* entre agentes, mas também a comunicação em nível de conhecimento, isto é, os agentes devem ser capazes de comunicar entre si suas crenças, metas e planos.

Entre os diversos modelos de agentes, esta pesquisa irá adotar o modelo *Beliefs-Desires-Intentions* (BDI) [Bratman, 1987]. A ideia central do modelo BDI é que um programa computacional tem uma espécie de estado mental. O modelo BDI usa três conceitos com origem no campo da filosofia e que são adaptados para os agentes:

- crenças (*beliefs*): crenças são informações que os agentes conhecem sobre o mundo. Estas informações podem ser imprecisas ou desatualizadas;

- desejos (*desires*): desejos são todos os estados das coisas que o agente gostaria de atingir. Possuir um desejo não implica que o agente irá de fato trabalhar em prol do desejo. Um desejo é apenas uma influência que o agente pode considerar para realizar ações e podem ser pensados como uma representação do estado motivational do agente;

- intenções (*intentions*): intenções são estados das coisas que o agente decidiu perseguir. Elas podem ser as metas que foram delegadas a um agente ou podem ser o resultado da escolha de algum desejo entre os existentes.

Uma das arquiteturas de agentes BDI mais conhecidas é o *Procedural Reasoning System* (PRS), proposta por Georgeff and Lansky [Georgeff e Lansky, 1987]. O PRS consiste em um conjunto de elementos (Fig. 1). A base de dados (*database*) contem as crenças sobre o mundo e sobre o estato interno do agente. (*Goals*) contém o conjunto de metas para serem atingidas pelo agente. A biblioteca de planos (*KA Library*) contém os planos (também conhecidos

por *Knowledge Areas (KAs)*) que são usados para atingir as metas ou reagir em situações particulares. Cada plano consiste em um corpo, que descreve as ações do plano, e uma condição de invocação, que especifica em que situação o plano é aplicável. Finalmente, a estrutura de intenções contém os planos que foram escolhidos para uma eventual execução. O agente interage com o ambiente, incluindo outros agentes, através de sua base de dados (que é atualizada como resposta ao ambiente) e através de ações que o agente realiza de acordo com suas intenções [Georgeff e Ingrand, 1989, Ingrand et al., 1992, Georgeff e Lansky, 1987].

Um interpretador (*interpreter*) (ou mecanismo de inferência) manipula esses componentes, através da seleção cíclica de planos apropriados considerando as metas e crenças, colocando os planos selecionados na estrutura de intenções, e por fim executando-os. Em um certo momento da execução, certas metas estão ativas e certas crenças estão contidas na base de crenças do agente. Dadas estas metas e crenças, um sub-conjunto de planos do agente pode ser aplicável (isto é, podem ser invocados). Um ou mais destes planos aplicáveis serão escolhidos para a execução e serão colocados na estrutura de intenções. Uma intenção é então selecionada para a execução. Como a execução completa de qualquer sequência de ações até o seu final aumenta o risco de que mudanças significativas irão ocorrer durante a execução e o agente poderia falhar para atingir o objetivo pretendido, apenas uma ação da intenção é executada [Rao e George, 1995]. Uma ação de uma intenção pode ser ou uma ação primitiva ou uma ou mais sub-metas. No primeiro caso, a ação é imediatamente inicializada; no último caso, as sub-metas são novas metas para o agente. A execução de uma ação primitiva pode afetar não apenas o mundo externo, mas também o estado interno do agente. Por exemplo, uma ação primitiva pode operar diretamente nas crenças, metas, ou intenções do agente. Além disso, a ação poderia indiretamente afetar o estado do agente como um resultado de um novo conhecimento ganho pela sua interação com o mundo externo. Neste momento, o ciclo inicia novamente: novas metas e crenças disparam novos planos, um ou mais destes planos são selecionados e colocados na estrutura de intenções, e finalmente uma intenção é selecionada da estrutura de intenções para a sua execução parcial [Georgeff e Ingrand, 1989, Ingrand et al., 1992, Georgeff e Lansky, 1987].

Como uma extensão para o modelo BDI proposto em [Bratman, 1987], em [Cohen e Levesque, 1987, Cohen e Levesque, 1990], uma intenção pode ser vista como composta por outros dois conceitos, escolha (*choice*) ou meta (*goal*), e compromisso (*commitment*). A noção de compromisso é útil para que agentes BDI possam se comprometer com decisões tomadas previamente. Um compromisso faz um balanço entre reatividade e proatividade de um agente. Com as mudanças contínuas no ambiente, o compromisso traz um certo senso de estabilidade para o processo de raciocínio do agente. Um compromisso normalmente tem

duas partes: uma é a condição que agente está comprometido a manter, chamada de condição de compromisso (*commitment condition*), e a segunda é a condição em que o agente pode desistir do compromisso, chamada de condição de término (*termination condition*). Assim, um agente pode se comprometer com uma intenção considerando que a intenção estará sendo cumprida no futuro [Rao e George, 1995].

Algumas propriedades das intenções podem ser definidas com o uso de compromissos. O agente deveria (1) adotar intenções que ele acredita serem viáveis e desistir daquelas que ele acredita serem inviáveis; (2) manter (ou se comprometer) com intenções, porém não para sempre; (3) remover as intenções que ele acredita que foram satisfeitas; (4) alterar intenções quando mudanças relevantes nas crenças ocorrem; e (5) adotar intenções subsidiárias quando necessário. Adotar uma intenção tem efeitos no estado mental do agente, pois afetam as crenças, compromissos com ações futuras, e outras intenções. Intenções também podem ser vistas como um tipo de meta persistente (*persistent goal*). Uma meta persistente é uma meta que será mantida enquanto uma condição é verdadeira. Se a condição falha, o agente deve remover o compromisso para atingir a meta. Persistência envolve um compromisso interno do agente com uma série de eventos ao longo do tempo [Cohen e Levesque, 1987, Cohen e Levesque, 1990].

Ao longo dos anos, diversas linguages inspiradas no modelo BDI foram propostas. Algumas das primeiras linguages de agentes propostas foram AGENT0 [Shoham, 1991], METATEM [Fisher e Barringer, 1991], AgentSpeak(L) [Rao, 1996], APRIL [McCabe e Clark, 1995], INTERRAP [Fischer et al., 1995], Agent Factory [O'Hare, 1996], Ciao Prolog [Hermenegildo et al., 1995], JIAC Agent Framework [Wieczorek e Albayrak, 1998], 3APL [Hindriks et al., 1999], Jinni [Tarau, 1999]. Linguagens e plataformas de agentes mais recentes incluem IndiGolog [Giacomo et al., 2009], Jason [Bordini et al., 2007], GOAL [Hindriks, 2009], 2APL [Dastani, 2008], IMPACT [Dix e Zhang, 2005], Jadex [Pokahr et al., 2005], JACK [Evertsz et al., 2003], ALOO [Ricci e Santi, 2013], CLAIM [Fallah-Seghrouchni e Suna, 2003], ViP [Kinny, 2002], ConGolog [de Giacomo et al., 2000], MINERVA [Leite et al., 2002], etc. Uma revisão mais detalhada das linguagens existentes é feita em [Sadri e Toni, 1999, Bordini et al., 2006, Mascardi et al., 2004, Bordini et al., 2009, Bordini et al., 2005].

3 Revisão da Literatura

Embora existam trabalhos relacionados à avaliação da eficiência de um agente ou SMA, tais trabalhos normalmente exploram apenas a questão de comunicação entre agentes [Jurasovic et al., 2006, Hmida et al., 2008, Timm e Schumann, 2009]. Nesta seção, citamos brevemente

alguns trabalhos que, entre outras coisas, procuram avaliar a eficiência de um agente ou SMA.

De acordo com [Hmida et al., 2008, Helsinger et al., 2003], a avaliação de um SMA deveria ser feita considerando níveis de abstração diferentes. Em [Hmida et al., 2008], os autores sugerem dois níveis de abstração. No primeiro nível estão todos os dados relativos aos efeitos que o SMA tem sobre a infraestrutura de *hardware* onde ele está sendo executado, tais como a utilização da CPU ou consumo de memória. No segundo nível estão os dados relativos às características próprias de um SMA, tais como a comunicação, que é o critério adotado pelos autores. Da mesma forma, em [Helsinger et al., 2003], a avaliação de um SMA é feita considerando diferentes níveis de abstração. Em um primeiro nível estão dados relacionados aos efeitos que a execução do SMA têm sobre o computador onde o SMA está hospedado (uso da CPU, uso da rede, consumo de memória). Em um nível acima estão métricas que representam o nível de agentes, tais como tráfego de mensagens e contador de tarefas. No nível mais alto estão as medidas específicas da aplicação que se preocupam com estruturas de dados, processos de computação e desempenho relativo aos requisitos da aplicação.

Em [Lee et al., 1998], os autores identificam outros fatores importantes introduzidos no nível de agentes e que também tem custos e portanto afetam a eficiência do SMA. Entre eles estão o custo com coordenação (que envolve raciocínio e comunicação), o modelo de conhecimento dos agentes (que são estruturas usadas e como são manipuladas) e o modelo de raciocínio. A partir disso é definido que a eficiência de um SMA é medida usando um conjunto de indicadores das saídas de um SMA e seu consumo de recursos, onde estes indicadores incluem *throughput*, tempo de resposta, número de agentes e tarefas concorrentes, tempo computacional e *overhead* de comunicação.

Outros critérios para avaliação da eficiência de um SMA são apresentados em [Nagwani, 2009, Van Bien, 2008, Van Bien et al., 2010b, Van Bien et al., 2010a]. Entre os critérios propostos em [Nagwani, 2009] estão o número de agentes (que indica o quão grande é o SMA em questão), o tempo computacional ocupado por um determinado agente no SMA (que é o tempo que algum agente em particular está em um estado ativo, ou seja, executando alguma coisa), a memória utilizada por um agente no SMA (que compreende todos os objetos criados pelo agente), o número de trocas de estados do agente (entre ativo, esperando, etc), a comunicação (tamanho médio das mensagens trocadas, número de mensagens enviadas e recebidas e o tempo de atraso são alguns parâmetros envolvidos), o custo com o gerenciamento de agentes (o número de agentes em estado ativo e o número de agentes esperando para realizar alguma tarefa são alguns dos parâmetros) e finalmente as dependências entre os agentes (ex: um agente que precisa esperar pela tarefa que outro agente deve concluir). Nagwani também propõe a avaliação individual de cada agente e por fim uma avaliação global de todo o SMA,

que nada mais seria que a combinação de todas as informações individuais dos agentes. Já, em [Van Bien, 2008, Van Bien et al., 2010b, Van Bien et al., 2010a], os autores procuram explorar critérios para a avaliação da execução de um SMA mais relacionados aos conceitos do paradigma de agentes. Um sistema de *profiling* é desenvolvido e são medidos a população de agentes, interações realizadas (mensagens enviadas e recebidas), eventos e ações realizadas pelos agentes, sendo que estas últimas podem ter um certo tempo de duração (ex: uma ação pode levar mais ou menos tempo que outra ação).

Diferentemente dos trabalhos anteriores, alguns trabalhos conduzidos por Wooldridge [Wooldridge, 2000, Wooldridge e Dunne, 2001, Dunne et al., 2002, Dunne et al., 2003, Wooldridge e Dunne, 2005] investigam a complexidade computacional de um problema fundamental em SMA: *dado um ambiente, juntamente com a especificação de uma tarefa, é possível construir um agente que irá garantir a realização da tarefa naquele ambiente?*. Os autores buscam responder se existe um agente com alguma chance de atingir uma meta (*achievement goal*) ou manter uma condição (*maintenance goal*), e até qual seria a complexidade computacional para este agente cumprir a meta. Modelos formais abstratos para agentes e ambientes são propostos e usados como base para avaliar a complexidade computacional considerando diferentes características do ambiente (ex: se determinístico ou não determinístico) e metas que os agentes precisam cumprir (se *achievement goal* ou *maintenance goal*). Assim, a linha de pesquisa conduzida por Wooldridge foca na problemática de construir um agente para resolver um problema e não na avaliação de um programa de agente propriamente dito. Por exemplo, os autores provam que o problema para agentes cumprirem uma meta em dada circunstância varia desde indecidível (*undecidable problem*), no caso de agentes que possuem *maintenance goal* em um ambiente ilimitado, até tratável (*NL-complete*), no caso de agentes possuírem *achievement goal* ou *maintenance goal* em um ambiente determinístico.

Como observado por meio dos trabalhos citados, ainda há espaço para pesquisa sobre como avaliar a execução de um agente. As formas atuais de medir o desempenho de um agente ainda não são ideais em muitos casos. Por exemplo, muitas ferramentas e técnicas realizam a avaliação da eficiência em tempo de execução, o que depende da infraestrutura de *hardware* onde o agente é executado, da plataforma de execução, e da instância de execução. Assim, os critérios de avaliação podem ter o efeito direto ou indireto da plataforma de execução e não permitem uma avaliação do que de fato está relacionado ao agente, isto é, considerando exclusivamente os programas de agentes. São necessárias métricas para que fosse possível avaliar não só a eficiência através de efetiva execução de um agente, mas também que explorem outros conceitos de agentes e permitam tornar a análise da eficiência independente de fatores como a plataforma de execução e a infraestrutura de *hardware*.

4 Justificativa

Baseando-se na leitura e estudo de artigos relacionados (Sec. 3) e também no decorrer do desenvolvimento da tese de doutorado [Zatelli et al., 2015b, Zatelli et al., 2015a, Zatelli et al., 2017], observou-se algumas questões em aberto em relação à execução de um agente:

- atualmente é possível avaliar a eficiência de um agente apenas considerando medidas temporais ou dependentes da sua execução, assim não há uma forma de avaliar a eficiência de um agente independentemente da infraestrutura de *hardware* onde ele está sendo executado e da plataforma de execução do agente;
- não é possível estimar o tempo de execução sem que o sistema seja efetivamente executado;
- não há um estudo sobre complexidade computacional no paradigma de agentes com o objetivo de calcular a complexidade computacional de um programa de agente considerando seus próprios conceitos e características.

A partir das justificativas levantadas foram levantadas algumas questões:

- Como avaliar a eficiência de um agente independentemente da infraestrutura de *hardware* onde o agente é executado?
- Como avaliar a eficiência de um agente independentemente de plataforma de execução?
- Como avaliar a complexidade computacional para atingir uma determinada meta?
- Como calcular a complexidade computacional de um programa de agente?
- Como comparar a complexidade computacional de dois planos? E de dois programas de agentes?
- Como estimar o tempo de execução de um programa de agente?

5 Objetivos

O objetivo geral desta pesquisa é elaborar uma técnica para calcular a complexidade computacional de um programa de agente a fim de avaliar a eficiência de um agente com base no uso de recursos computacionais (ex: memória, CPU), independentemente de plataforma de execução ou infraestrutura de *hardware*. Os objetivos específicos da pesquisa são:

- identificar características de execução de agentes e conceber um modelo de execução para um agente;
- identificar como os recursos computacionais (ex: CPU) são utilizados pelos agentes;
- estabelecer semelhanças e diferenças entre a análise de um algoritmo clássico e de um programa de agente;
- propor uma forma de estimar o tempo de execução de um agente com base na complexidade computacional do agente;
- conceber cenários-chave para avaliar a proposta.

O escopo desta pesquisa, além de adotar o modelo BDI, também considera apenas o programa de agente para efeitos de cálculo da complexidade computacional, e não outros elementos, como a interação entre agentes ou mesmo o ambiente em que os agentes estão situados.

6 Modelo de Execução de um Agente BDI

Nesta seção é feita uma caracterização da execução de agentes BDI, para assim propor ou adotar um modelo de execução. Neste modelo, entre outras coisas, é necessário identificar de maneira clara as etapas de execução de uma agente (por exemplo, o ciclo de raciocínio), bem como a forma em que suas intenções são executadas. O modelo conceitual de agente é apresentado na Sec. 6.1 enquanto que seu ciclo de raciocínio é apresentado na Sec 6.2.

6.1 Modelo Conceitual

Como modelo de agentes a ser utilizado neste trabalho, optou-se por um modelo BDI utilizado em várias linguagens de programação (Fig. 2), tais como 2APL [Dastani, 2008] e Jason [Bordini et al., 2007]. No modelo utilizado, são considerados os conceitos de crenças (*beliefs*), metas (*goals*), intenções (*intentions*), desejos (*desires*), eventos (*events*) e planos (*plans*). Um agente é basicamente composto por uma base de crenças (*belief base*), metas (*goals*), e uma biblioteca de planos (*plan library*).

Crenças são informações que os agentes possuem em um certo momento. Elas podem ser sobre ele próprio, sobre o ambiente ou sobre outros agentes. Metas são estados das coisas que os agentes querem alcançar (por exemplo, um estado do ambiente). Enquanto as intenções representam as metas que o agente já deliberou e se comprometeu em alcançar, os desejos representam as metas que ainda não estão sendo perseguidas pelo agente.

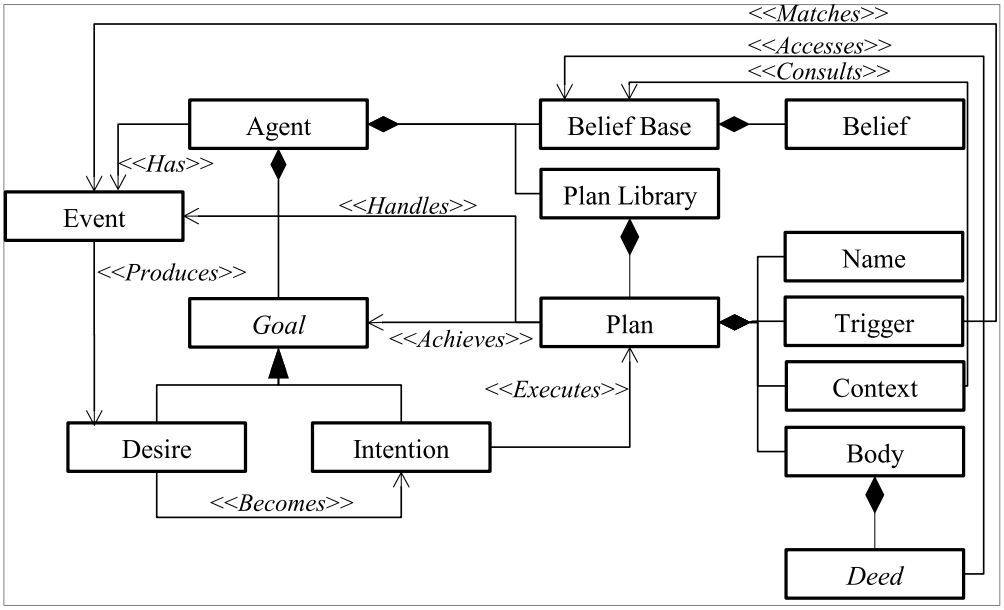

Figura 2: *Modelo conceitual de agentes.*

A biblioteca de planos é composta por planos, que são meios de tratar algum evento ou atingir alguma meta. Um plano é composto de um nome único (*name*), um gatilho (*trigger*), um contexto (*context*) e um corpo (*body*). O gatilho é um evento que o plano pode tratar (por exemplo, a adoção de alguma meta). O contexto é usado para especificar as condições para a aplicação de determinado plano e é uma fórmula lógica que deve ser avaliada de acordo com as crenças do agente. O corpo do plano é uma sequência de passos (*deeds*[2]) que devem ser executados pelo agente.

Em tempo de execução, quando o agente pretende fazer alguma coisa, ele deve iniciar a agir para satisfazer aquela intenção. As ações que o agente deve desempenhar para satisfazer determinada intenção estão definidas nos planos que o agente possui em sua biblioteca de planos. Uma intenção é assim satisfeita por meio da execução de panos. Intenções podem ser criadas, suspendidas, ou continuadas a qualquer momento, e ela é considerada terminada quando ou o plano foi executado com sucesso, ou a execução de determinado plano falhou (por exemplo, quando o agente falha para desempenhar determinada ação), ou a intenção foi descartada pelo próprio agente (por exemplo, quando o agente não tem mais a intenção de realizar algo).

Vários eventos podem ocorrer em tempo de execução. Estes eventos podem produzir desejos para o agentes (por exemplo, uma mensagem recebida por um agente pode conter uma requisição para o agente fazer algo, que produz um desejo a ser satisfeito). No modelo a

[2]O termo *deed* é usado da mesma forma que em [Dennis et al., 2012] e ele refere-se aos diversos tipos de fórmulas que podem aparecer no corpo do plano.

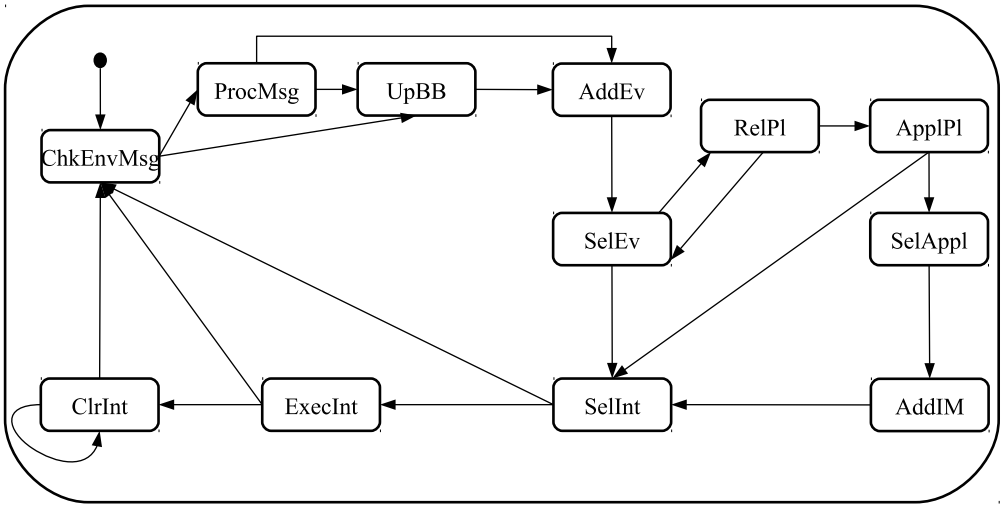

Figura 3: *Ciclo de raciocínio do agente.*

ser considerado neste trabalho, são considerados quatro tipos de eventos: (1) adição e remoção de crenças; (2) mensagens recebidas por um agente; (3) percepções que são produzidas pelo ambiente e percebidas pelo agente; (4) adoção, descarte, satisfação, falha, suspensão ou continuação de metas.

6.2 Ciclo de Raciocínio do Agente

A Fig. 3 detalha uma implementação prática do ciclo de raciocínio apresentado na Fig. 1. Este ciclo de raciocínio é adotado na plataforma Jason e o mesmo pode ser conceitualmente dividido em três principais estágios: a percepção (*sense*), deliberação (*deliberate*) (ou raciocínio) e atuação (*act*).

No estágio de percepção, o agente percebe o ambiente e recebe mensagens de outros agentes (`ChkEnvMsg`). Como resultado da percepção do ambiente, o agente obtém todas as percepções que representam o estado do ambiente. Cada percepção representa uma propriedade particular do estado atual do ambiente (por exemplo, a temperatura de alguma coisa) e a base de crenças é atualizada a fim de refletir este estado (`UpBB`). As mensagens recebidas pelo agente são processadas de acordo com suas performativas, que podem implicar na adição ou remoção de planos, metas e crenças (`ProcMsg`). Em ambos os casos (mudanças na base de crenças ou mensagens recebidas pelo agente), são produzidos eventos e os mesmos adicionados em uma fila (`AddEv`).

No estágio de deliberação, o agente trata tais eventos. Some um único evento pendente é selecionado para ser tratado em cada ciclo de raciocínio (`SelEv`). O próximo passo é recuperar todos os planos da biblioteca de planos que são relevantes para tratar o evento selecionado (`RelPl`). Por exemplo, sejam os planos da Fig. 4 serem os planos atuais que o agente possui

```
1 @p1 +pressure(P): P > 1024 <- ...
2 @p2 +temperature(T): T < 10 <- ...
3 @p3 +temperature(T): T >= 10 & T <= 20 <- ...
4 @p4 +temperature(T): T > 20 <- ...
5 @p5 +temperature(T): T > 30 <- ...
```

Figura 4: *Exemplos de planos na biblioteca de planos.*

na biblioteca de planos, onde a construção básica de um plano é dada por `@name trigger: context <- body`. Se a crença `temperature(32)` é incluída na base de crenças do agente, um evento `+temperature(32)` é produzido. Quando o evento `+temperature(32)` acontece, somente os planos `@p2`, `@p3`, `@p4`, e `@p5` podem ser inicialmente considerados para tratá-lo, isto é, apenas eles são relevantes para este evento.

Dos planos relevantes, aqueles que são aplicáveis considerando o contexto atual são selecionados (`ApplPl`), isto é, aqueles que podem ser usados para tratar o evento considerando o estado atual das crenças do agente. Continuando com o exemplo do evento `+temperature(32)`, somente dois planos são aplicáveis (`@p4` e `@p5`). Eles são aplicáveis porque depois da unificação, a variável `T` passa a possuir o valor `32`, que é maior que `20` (`@p4`) e `30` (`@p5`).

O agente pode ainda possuir vários planos aplicáveis e qualquer um deles poderia ser escolhido para (possivelmente) tratar o evento com sucesso. O próximo passo é então selecionar um desses planos aplicáveis para que o agente possa se comprometer (`SelAppl`). Por padrão, é selecionado o primeiro deles. Depois de selecionar o plano, o agente finalmente possui a intenção de executar a sequência de passos determinados pelo plano (`AddIM`).

Várias intenções podem estar ativas ao mesmo tempo e competindo pela atenção do agente. As intenções ativas são armazenadas em uma fila e em cada ciclo de raciocínio , somente uma intenção é selecionada para ser executada (`SelInt`). Por padrão, uma intenção é selecionada usando um mecanismo de escalonamento estilo `round-robin`. Isto significa que em cada ciclo exatamente uma intenção diferente é selecionada e somente um passo dela é executado (`ExecInt`). A razão para que somente um passo de uma intenção é executado em cada ciclo, além de permitir que várias intenções pudessem ser executadas concorrentemente, é que a execução de uma sequência completa de passos aumenta significativamente o risco de que mudanças possam ocorrer durante a execução e o agente possa falhar para atingir o objetivo pretendido [Rao e George, 1995].

De forma prática, uma intenção é como uma pilha de planos instanciados (isto é, *intended means*), onde o plano no topo da pilha é o que está atualmente pronto para ser executado. Qual a intenção que será de fato executada depende do tipo de passo que está no início do corpo do plano que está no topo da pilha. Se o passo causar a instanciação de um novo plano,

este plano será colocado no topo da pilha daquela intenção. A instanciação de um novo plano pode acontecer, por exemplo, quando p passo no início do plano causa a adoção de uma meta e há um plano aplicável para satisfazer aquela meta. Os planos restantes, abaixo do topo da pilha, continuação suas execuções somente quando o novo plano instanciado terminar com sucesso. Neste caso, uma analogia entre a execução de uma intenção e uma execução de um método de uma linguagem orientada a objetos pode ser feita. Se um método (por exemplo, α) chamar outro método (por exemplo β), o método α apenas continuará a execução quando a chamada do método β terminar sua execução. Se a execução de um passo falhar, a intenção também falha. Assim, se o último passo de um plano é executado com sucesso, a intenção é considerada terminada com sucesso. Uma intenção pode estar ativa ou suspensa. Intenções suspendidas não são consideradas pelo escalonador. Finalmente, o último passo do ciclo de raciocínio é remover as intenções vazias, ou seja, aquelas terminadas (`ClrInt`). Após isso, o ciclo inicia novamente pelo estágio de percepção.

7 Discussão e Considerações Finais

Considerando a revisão da literatura realizada e o modelo de execução de agentes BDI propostos, alguns resultados importantes já foram identificados, porém outros resultados esperados ainda não foram alcançados. Os próximos parágrafos destacam alguns desses resultados já identificados bem como resultados ainda esperados.

Uma análise da literatura relacionada à avaliação da eficiência de um agente ou SMA foi realizada e constatou-se que muitas das ferramentas e propostas pesquisadas realizam a avaliação de eficiência apenas considerando o tempo efetivo de execução, o que depende da infraestrutura de *hardware* onde o agente ou SMA está sendo executado, além de depender também da plataforma de execução dos agentes ou SMA e da própria instância de execução. Isso reforça ainda mais a importância de desenvolver uma técnica de análise de agentes que seja independente de plataforma de execução, infraestrutura de *hardware* ou outros elementos que poderiam afetar o resultado da avaliação.

Um modelo de execução de agente BDI foi proposto para ser seguido no decorrer do desenvolvimento deste trabalho. Este modelo é independente de plataforma de agentes e contempla os principais conceitos de agentes BDI, tais como crenças, desejos, intenções, metas, planos, e eventos. Além disso, um modelo de execução de ciclo de raciocínio foi proposto, sendo que o mesmo cobre as principais etapas de tomada de decisão de um agente BDI, sendo executadas de maneira cíclica e síncrona, que são a etapa de perceber o ambiente (e mensagens de outro agentes), deliberar, e agir. A cada execução do ciclo de raciocínio, no máximo um

passo de plano é executado, o qual chamamos de *deed*, sendo que este pode ser um envio de mensagem para outro agente, uma ação realizada pelo agente no ambiente, uma inclusão ou remoção de crença, ou alguma operação realizada sobre alguma meta.

A proposta desse modelo já nos dá indícios de que estamos no caminho para desenvolver um método de avaliação de desempenho de um agente que é independente de infraestrutura de *hardware* e também de plataforma de execução, pois podemos já perceber que ao contar o número de ciclos que o agente deve executar para desempenhar alguma tarefa (executar um plano, satisfazer uma meta), sabendo que em cada ciclo apenas um passo de plano é executado, estaremos já conseguindo obter informações do quão "complexo"é realizar a tarefa. Assim, a contagem do número de ciclos de raciocínio como uma forma de medir o desempenho de um programa de agente, nos permite também já ter indícios de respostas para algumas das questões de pesquisa (apresentadas na Sec. 4), tais como:

- Como avaliar a eficiência de um agente independentemente da infraestrutura de *hardware* onde o agente é executado?
- Como avaliar a eficiência de um agente independentemente de plataforma de execução?
- Como comparar a complexidade computacional de dois planos?

Para as duas primeiras perguntas, é possível observar que o número de ciclos de raciocínio, como mencionado acima, é independente de infraestrutura de *hardware* ou plataforma de execução onde o agente é executado - esse número de ciclos de raciocínio é possível de ser estimado analisando o próprio código fonte do agente, para cada plano individualmente. Já para a terceira pergunta, é possível fazer uma contagem de *deeds* de um determinado plano, considerando possíveis repetições/recursões e chamadas de sub-planos. Uma vez realizada a contagem de *deeds* de diferentes planos, é possível então compará-los. A notação assintótica (Big-O clássica) pode ser utilizada neste caso para uma comparação mais justa.

Além dos resultados já alcançados, ainda há uma série de outros resultados importantes esperados, por exemplo, uma análise mais profunda em relação a como calcular a complexidade computacional de um programa de agente independentemente da plataforma de execução e da infraestrutura de *hardware* onde o agente é executado, ou seja, um aprofundamento na ideia de contagem de número de ciclos de raciocínio executados e de sua viabilidade e precisão. Para isso, seria necessário estimar primeiramente a complexidade computacional de cada plano individualmente (já considerando a contagem de número de ciclos de raciocínio), depois estender essa complexidade computacional para as diversas metas que o agente possui, lembrando que cada meta pode possuir diversos planos para alcançá-la, e por fim estendendo essa

complexidade computacional para os objetivos gerais do sistema, em que o agente está participando, ou seja, para determinar aproximadamente quando o agente finalizará sua execução por completo. Em especial, quatro das perguntas de pesquisa ainda encontram-se abertas e devem ser exploradas, as quais estão bastante relacionadas a como um agente faz a escolha de planos para atingir uma determinada meta e cumprir o objetivo do sistema em que o agente se encontra, ou seja, ao seu processo de deliberação.

- Como avaliar a complexidade computacional para atingir uma determinada meta?
- Como calcular a complexidade computacional de um programa de agente?
- Como comparar a complexidade computacional de dois programas de agente?
- Como estimar o tempo de execução de um programa de agente?

Espera-se, nos futuros passos de desenvolvimento desta pesquisa, obter um modelo mais completo e uma teoria que contemple a avaliação da complexidade computacional de um programa de agente. A principal contribuição virá no enriquecimento do paradigma de agentes com técnicas mais apropriadas para a avaliação da eficiência de um agente, por meio da complexidade computacional, permitindo, entre outras coisas, estimar com mais precisão o tempo de execução de um agente nas fases de projeto e desenvolvimento.

As seções a seguir destacam direções de etapas para trabalhos futuros a serem realizados a partir da revisão bibliográfica realizada e do modelo de execução de agente BDI proposto.

7.1 Complexidade Computacional em Agentes

Nesta etapa, deve-se explorar a teoria de complexidade computacional a fim de adaptá-la para melhor calcular a complexidade computacional de um programa de agente, considerando seus próprios conceitos, características e modelo de execução. Cada nível de abstração identificado no modelo de execução de agente BDI proposto (meta, plano, *deed*) também deve ser considerado para calcular a complexidade computacional total de um programa de agente. Assim, a complexidade computacional de um agente deve considerar deste a análise específica de um programa de agente (independentemente de plataforma de execução), até incrementalmente considerar a complexidade da plataforma de execução e de outras estruturas internas de um agente. Assim, além de deixar clara a complexidade do programa de agente em si, é possível também calcular um custo computacional mais preciso e realístico quando a plataforma de execução também é considerada. A mesma tem um impacto na complexidade total de um agente quando o seu código é efetivamente executado e não deve ser negligenciada quando pretende-se estimar o tempo de execução mais precisamente.

7.2 Avaliação

O modelo e teoria de complexidade computacional propostos deve ser avaliado com base em diversos cenários e também de maneira formal. Os cenários devem ser concebidos de maneira que explorem características e demandas diferentes, que podem ser gargalos em algum agente, variações de carga de troca de mensagens, dinamicidade do ambiente, quantidade e tipos de metas, entre outros. Além disso, deve ser feita uma discussão a respeito de outras métricas utilizadas na literatura e que objetivam avaliar a eficiência de um agente. Por fim, uma comparação para realçar as semelhanças e diferenças entre a complexidade computacional utilizada na análise de algoritmos e a complexidade computacional proposta para avaliar um programa de agente também é interessante de ser realizada.

Referências

[Ajitha et al., 2009] Ajitha, S., Kumar, T., Geetha, D., e Rajanikanth, K. (2009). Predicting performance of multi-agent systems during feasibility study. Em *Proc. of IAMA*, páginas 1–5.

[Bordini et al., 2006] Bordini, R. H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A. E., Gómez-Sanz, J. J., Leite, J. a., O'Hare, G. M. P., Pokahr, A., e Ricci, A. (2006). A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, 30(1):33–44.

[Bordini et al., 2005] Bordini, R. H., Dastani, M., Dix, J., e Seghrouchni, A. E. F. (2005). *Multi-agent programming : languages, platforms and applications.*, volume 15 of *Multiagent systems, artificial societies, and simulated organizations.* Springer, New York.

[Bordini et al., 2009] Bordini, R. H., Dastani, M., Dix, J., e Seghrouchni, A. E. F. (2009). *Multi-Agent Programming: Languages, Tools and Applications.* Springer Publishing Company, Incorporated, 1st edição.

[Bordini et al., 2007] Bordini, R. H., Hübner, J. F., e Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason.* Wiley, Liverpool.

[Bratman, 1987] Bratman, M. E. (1987). *Intentions, Plans and Practical Reason.* Harvard University Press, Cambridge, MA, USA.

[Camacho e Aler, 2005] Camacho, D. e Aler, R. (2005). Software and performance measures for evaluating multi-agent frameworks. *Applied Artificial Intelligence*, 19(6):645–657.

[Cohen e Levesque, 1987] Cohen, P. R. e Levesque, H. J. (1987). Intention = choice + commitment. Em *National Conference in AI.*

[Cohen e Levesque, 1990] Cohen, P. R. e Levesque, H. J. (1990). Intention is choice with commitment. *Artif. Intell.*, 42(2-3):213–261.

[Dastani, 2008] Dastani, M. (2008). 2apl: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248.

[de Giacomo et al., 2000] de Giacomo, G., Lespérance, Y., e Levesque, H. J. (2000). Congolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 121(1-2):109–169.

[Dennis et al., 2012] Dennis, L. A., Fisher, M., Webster, M. P., e Bordini, R. H. (2012). Model checking agent programming languages. *Automated Software Engg.*, 19(1):5–63.

[Di Bitonto et al., 2010] Di Bitonto, P., Laterza, M., Roselli, T., e Rossano, V. (2010). An evaluation method for multi-agent systems. Em *Proceedings of the 4th KES international conference on Agent and multi-agent systems: technologies and applications, Part I*, KES-AMSTA'10, páginas 32–41, Berlin, Heidelberg. Springer-Verlag.

[Dix e Zhang, 2005] Dix, J. e Zhang, Y. (2005). Impact: A multi-agent framework with declarative semantics. Em Bordini, R. H., Dastani, M., Dix, J., e Fallah-Seghrouchni, A. E., editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, páginas 69–94. Springer.

[Dunne et al., 2003] Dunne, P. E., Laurence, M., e Wooldridge, M. (2003). Complexity results for agent design problems. *Annals of Mathematics, Computing & Teleinformatics*, 1:2003.

[Dunne et al., 2002] Dunne, P. E., Wooldridge, M., e Laurence, M. (2002). The computational complexity of boolean and stochastic agent design problems. Em *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, AAMAS '02, páginas 976–983, New York, NY, USA. ACM.

[Evertsz et al., 2003] Evertsz, R., Fletcher, M., Jones, R., Jarvis, J., Brusey, J., e Dance, S. (2003). Implementing industrial multi-agent systems using jack. Em Dastani, M., Dix, J., e Fallah-Seghrouchni, A. E., editors, *PROMAS*, volume 3067 of *Lecture Notes in Computer Science*, páginas 18–48. Springer.

[Fallah-Seghrouchni e Suna, 2003] Fallah-Seghrouchni, A. E. e Suna, A. (2003). Claim: A computational language for autonomous, intelligent and mobile agents. Em Dastani, M., Dix, J., e Fallah-Seghrouchni, A. E., editors, *PROMAS*, volume 3067 of *Lecture Notes in Computer Science*, páginas 90–110. Springer.

[Fischer et al., 1995] Fischer, K., Müller, J. P., e Pischel, M. (1995). A pragmatic bdi architecture. Em Wooldridge, M., Müller, J. P., e Tambe, M., editors, *ATAL*, volume 1037 of *Lecture Notes in Computer Science*, páginas 203–218. Springer.

[Fisher e Barringer, 1991] Fisher, M. e Barringer, H. (1991). Concurrent metatem processes - a language for distributed ai. Em *In Proceedings of the European Simulation Multiconference*.

[Georgeff e Ingrand, 1989] Georgeff, M. P. e Ingrand, F. F. (1989). Decision-making in an embedded reasoning system. Em *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'89, páginas 972–978, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Georgeff e Lansky, 1987] Georgeff, M. P. e Lansky, A. L. (1987). Reactive reasoning and planning. Em *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 2*, AAAI'87, páginas 677–682. AAAI Press.

[Giacomo et al., 2009] Giacomo, G., Lespérance, Y., Levesque, H., e Sardina, S. (2009). Indigolog: A high-level programming language for embedded reasoning agents. Em El Fallah Seghrouchni, A., Dix, J., Dastani, M., e Bordini, R. H., editors, *Multi-Agent Programming:*, páginas 31–72. Springer US.

[Helsinger et al., 2003] Helsinger, A., Lazarus, R., Wright, W., e Zinky, J. (2003). Tools and techniques for performance measurement of large distributed multiagent systems. Em *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, páginas 843–850, New York, NY, USA. ACM.

[Hermenegildo et al., 1995] Hermenegildo, M., Bueno, F., de la Banda, M. G., e Puebla, G. (1995). The ciao multi-dialect compiler and system: An experimentation workbench for future (c)lp systems (extended abstract). Em *IN PARALLELISM AND IMPLEMENTATION OF LOGIC AND CONSTRAINT LOGIC PROGRAMMING*, páginas 65–85. Nova Science.

[Hindriks, 2009] Hindriks, K. V. (2009). Programming rational agents in goal. Em El Fallah Seghrouchni, A., Dix, J., Dastani, M., e Bordini, R. H., editors, *Multi-Agent Programming:*, páginas 119–157. Springer US.

[Hindriks et al., 1999] Hindriks, K. V., De Boer, F. S., Van Der Hoek, W., e Ch. Meyer, J.-J. (1999). Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401.

[Hmida et al., 2008] Hmida, F. B., Chaari, W. L., e Tagina, M. (2008). Performance evaluation of multiagent systems: communication criterion. Em *Proceedings of the 2nd KES International conference on Agent and multi-agent systems: technologies and applications*, KES-AMSTA'08, páginas 773–782, Berlin, Heidelberg. Springer-Verlag.

[Ingrand et al., 1992] Ingrand, F. F., Georgeff, M. P., e Rao, A. S. (1992). An architecture for real-time reasoning and system control. *IEEE Expert: Intelligent Systems and Their Applications*, 7(6):34–44.

[Jurasovic et al., 2006] Jurasovic, K., Jezic, G., e Kusek, M. (2006). A performance analysis of multi-agent systems. *ITSSA*, 1(4):335–342.

[Kinny, 2002] Kinny, D. (2002). Vip: a visual programming language for plan execution systems. Em *AAMAS*, páginas 721–728. ACM.

[Kleinberg e Tardos, 2005] Kleinberg, J. e Tardos, E. (2005). *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Lee et al., 1998] Lee, L. C., Nwana, H. S., Ndumu, D. T., e De Wilde, P. (1998). The stability, scalability and performance of multi-agent systems. *BT Technology Journal*, 16(3):94–103.

[Leite et al., 2002] Leite, J. a. A., Alferes, J. J., e Pereira, L. M. (2002). Minerva - a dynamic logic programming agent architecture. Em *Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, ATAL '01, páginas 141–157, London, UK, UK. Springer-Verlag.

[Magary, 2003] Magary, J. (2003). Examining performance issues of multiagent systems. Relatório técnico, University of Calgary.

[Mascardi et al., 2004] Mascardi, V., Martelli, M., e Sterling, L. (2004). Logic-based specification languages for intelligent software agents. *Theory Pract. Log. Program.*, 4(4):429–494.

[McCabe e Clark, 1995] McCabe, F. G. e Clark, K. L. (1995). April—agent process interaction language. Em *Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents*, ECAI-94, páginas 324–340, New York, NY, USA. Springer-Verlag New York, Inc.

[Nagwani, 2009] Nagwani, N. K. (2009). Performance measurement analysis for multi-agent systems. Em *Proc. of IAMA*.

[O'Hare, 1996] O'Hare, G. M. P. (1996). Agent Factory: An Environment for the Fabrication of Multi-Agent Systems. Em *Distributed Artificial Intelligence*, páginas 449–484.

[Pokahr et al., 2005] Pokahr, A., Braubach, L., e Lamersdorf, W. (2005). Jadex: A bdi reasoning engine. Em Bordini, R. H., Dastani, M., Dix, J., e Fallah-Seghrouchni, A. E.,

editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, páginas 149–174. Springer.

[Rao, 1996] Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. Em *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-agent World : Agents Breaking Away: Agents Breaking Away*, MAAMAW '96, páginas 42–55, Secaucus, NJ, USA. Springer-Verlag New York, Inc.

[Rao e George, 1995] Rao, A. S. e George, M. P. (1995). Bdi agents: From theory to practice. Em *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, páginas 312–319.

[Ricci e Santi, 2013] Ricci, A. e Santi, A. (2013). Concurrent object-oriented programming with agent-oriented abstractions: The aloo approach. Em *Proceedings of the 2013 Workshop on Programming Based on Actors, Agents, and Decentralized Control*, AGERE! '13, páginas 127–138, New York, NY, USA. ACM.

[Sadri e Toni, 1999] Sadri, F. e Toni, F. (1999). Computational logic and multi-agent systems: a roadmap. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net*.

[Shoham, 1991] Shoham, Y. (1991). Agent0: A simple agent language and its interpreter. Em Dean, T. L. e McKeown, K., editors, *AAAI*, páginas 704–709. AAAI Press / The MIT Press.

[Tarau, 1999] Tarau, P. (1999). Jinni: Intelligent mobile agent programming at the intersection of java and prolog. Em *In Proceedings of The Fourth International Conference on The Practical Application of Intelligent Agents and Multi-Agents*, páginas 109–123.

[Timm e Schumann, 2009] Timm, I. J. e Schumann, R. (2009). Performance measurement of multiagent systems: towards dependable mas. Em *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, páginas 22:1–22:8, San Diego, CA, USA. Society for Computer Simulation International.

[Van Bien, 2008] Van Bien, D. D. (2008). Agentspotter: a mas profiling system for agent factory. Dissertação de Mestrado, University College Dublin.

[Van Bien et al., 2010a] Van Bien, D. D., Lillis, D., e Collier, R. W. (2010a). Call graph profiling for multi agent systems. Em *Proceedings of the Second international conference*

on Languages, Methodologies, and Development Tools for Multi-Agent Systems, LADS'09, páginas 153–167, Berlin, Heidelberg. Springer-Verlag.

[Van Bien et al., 2010b] Van Bien, D. D., Lillis, D., e Collier, R. W. (2010b). Space-time diagram generation for profiling multi agent systems. Em *Proceedings of the 7th international conference on Programming multi-agent systems*, ProMAS'09, páginas 170–184, Berlin, Heidelberg. Springer-Verlag.

[Wieczorek e Albayrak, 1998] Wieczorek, D. e Albayrak, S. (1998). Open scalable agent architecture for telecommunication applications. Em Albayrak, S. e Garijo, F., editors, *Intelligent Agents for Telecommunication Applications*, volume 1437 of *Lecture Notes in Computer Science*, páginas 233–249. Springer Berlin Heidelberg.

[Wooldridge, 1995] Wooldridge, M. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*.

[Wooldridge, 2000] Wooldridge, M. (2000). The computational complexity of agent design problems. Em *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, páginas 341–348.

[Wooldridge, 2002] Wooldridge, M. (2002). *An introduction to multiagent systems*. John Wiley & Sons Ltd, Chichester.

[Wooldridge e Dunne, 2001] Wooldridge, M. e Dunne, P. E. (2001). *Optimistic and Disjunctive Agent Design Problems*, páginas 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Wooldridge e Dunne, 2005] Wooldridge, M. e Dunne, P. E. (2005). The complexity of agent design problems: Determinism and history dependence. *Annals of Mathematics and Artificial Intelligence*, 45(3-4):343–371.

[Wooldridge e Jennings, 1999] Wooldridge, M. e Jennings, N. (1999). Software engineering with agents: Pitfalls and pratfalls. *IEEE Internet Computing*, 3(3).

[Zatelli et al., 2017] Zatelli, M. R., Hübner, J. F., e Ricci, A. (2017). *Exploiting Parallelism in the Agent Paradigm (ONGOING WORK)*. Tese de Doutorado, Federal University of Santa Catarina, Florianópolis, Santa Catarina, Brazil.

[Zatelli et al., 2015a] Zatelli, M. R., Ricci, A., e Hübner, J. F. (2015a). A concurrent architecture for agent reasoning cycle execution in jason. Em *Multi-Agent Systems and Agreement Technologies (EUMAS 2015/AT 2015)*, volume 9571 of *LNAI*. Springer.

[Zatelli et al., 2015b] Zatelli, M. R., Ricci, A., e Hübner, J. F. (2015b). Evaluating different concurrency configurations for executing multi-agent systems. Em Baldoni, M., Baresi, L., e Dastani, M., editors, *Engineering Multi-Agent Systems: Third International Workshop, EMAS 2015*, volume 9318 of *LNCS*, páginas 212–230, Cham. Springer.