Avaliação de Escalabilidade e Desempenho da Camada de Transporte de Mensagens em Plataformas Multiagente

Henrique Donâncio Nunes Rodrigues

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Programa: Ciências da Computação
Orientadora: Profa. Dra. Anarosa Alves Franco Brandão

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro do programa P&D da
ANEEL

São Paulo, agosto de 2019

Avaliação de Escalabilidade e Desempenho da Camada de Transporte de Mensagens em Plataformas Multiagente

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 12/08/2019. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profa. Dra. Anarosa Alves Franco Brandão (orientadora) - EP-USP
- Prof. Dr. Jomi Fred Hübner - DAS-UFSC
- Profa. Dra. Diana Francisca Adamatti - C3-FURG

Resumo

RODRIGUES, H. D. N. **Avaliação de Escalabilidade e Desempenho da Camada de Transporte de Mensagens em Plataformas Multiagente**. 2019. 120 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2019.

Este trabalho reside no campo de sistemas multiagente (MAS) compostos por agentes inteligentes que são capazes de usar protocolos de comunicação da Internet. Uma plataforma multiagente é um *software* ou *framework* capaz de gerenciar múltiplos aspectos da execução de agentes e suas interações. Muitas plataformas MAS foram desenvolvidas nos últimos anos, todas elas compatíveis com padrões de desenvolvimento de sistemas interoperáveis em diferentes níveis. Nos últimos anos, novas linguagens de programação foram definidas e novos protocolos foram adotados para comunicação em sistemas distribuídos. Esses fatos também influenciaram a comunidade multiagente, com a proposição de novas plataformas para apoiar o desenvolvimento de sistemas multiagente. Além disso, a adoção de agentes como paradigma para o desenvolvimento de sistemas distribuídos complexos em larga escala é vista como uma solução interessante na era do grande volume de dados. Portanto, uma comparação entre as plataformas existentes e seu suporte para desenvolver e implantar com eficiência sistemas multiagente de grande escala pode beneficiar a comunidade de desenvolvedores interessada em escolher qual plataforma melhor se adapta a seus projetos. O objetivo deste trabalho é avaliar plataformas multiagente em relação à escalabilidade, desempenho e compatibilidade com outras tecnologias com o objetivo de facilitar a escolha do desenvolvedor que queira projetar Sistemas Multiagente de grande porte. A fim de escolher as plataformas MAS para a comparação proposta, são consideradas plataformas de código aberto que são ativamente utilizadas pela comunidade multiagente. Além disso, tais plataformas MAS devem ser capazes de oferecer uma implantação de forma distribuída, característica essencial de sistemas escaláveis. Depois de restringir a lista de plataformas MAS de acordo com esses critérios, são analisados os sistemas de transporte de mensagens utilizando *benchmarks* para análise de escalabilidade e desempenho, considerando diferentes cenários de comunicação. Por fim, é apresentado um cenário realístico onde um MAS escalável pode ser adotado como solução.

Palavras-chave: Escalabilidade, Desempenho, Sistemas Multiagente.

Abstract

RODRIGUES, H. D. N. **Scalability and Performance Comparison Between Message Transport Systems of Multiagent Platforms**. 2019. 120 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2019.

This work resides in the field of multiagent systems (MAS) composed of intelligent agents that are able to use Internet communication protocols. A multiagent platform is a software or framework capable of managing multiple aspects of the agent execution and their interactions. In the recent years, many MAS platforms have been developed, all of them compliant with interoperable system development standards at different levels. Also, new programming languages have been defined and new protocols have been adopted for communication in distributed systems. These facts also influenced the multiagent community with the proposition of new platforms to support the development of multiagent systems. In addition, the adoption of agents as a paradigm for the development of large scale complex distributed systems is seen as an interesting solution in the era of big data. Therefore, a comparison between existing platforms and their support for efficiently developing and deploying large scale multiagent systems can benefit the developer community interested in choosing which platform best fits their projects. The purpose of this work is evaluate multiagent platforms for scalability, performance and compatibility with other technologies in order to facilitate the choice of the developer that wants design large scale multiagent systems. In order to choose MAS platforms for the proposed comparison, are considered open source platforms that are actively used by the multiagent community. Moreover, these MAS platforms should be able to provide a deployment in a distributed manner, essential characteristic of scalable systems. After narrowing the list of MAS platforms according to these criteria, message transport systems are analyzed using benchmarks for scalability and performance comparison, considering different communication scenarios. Finally, a realistic scenario is presented where a scalable MAS can be adopted as a solution.

Keywords: Scalability, Performance, Multiagent Systems.

Sumário

Lista de Abreviaturas	vii
Lista de Figuras	ix
Lista de Tabelas	xi
1 Introdução	1
1.1 Objetivo	2
1.2 Contribuições	3
1.3 Organização do Trabalho	3
2 Agentes e Sistemas Multiagente	5
2.1 Agentes Inteligentes e Sistemas Multiagente	5
2.2 Linguagens de Comunicação de Agentes	7
2.3 Plataformas Multiagente	8
2.3.1 JADE	9
2.3.2 SPADE	10
2.3.3 JaCaMo	11
2.3.4 ASTRA	12
2.3.5 JIAC V	13
2.4 Escalabilidade em MAS	14
3 Trabalhos Relacionados	17
3.1 Benchmarks e critérios para análise de escalabilidade e desempenho em MAS	17
3.2 Desempenho e transporte de mensagens em MAS	18
3.3 Escalabilidade e desempenho em MAS	19
3.4 Exemplos de MAS de larga escala	21
4 Desempenho e Escalabilidade da Camada de Transporte de Mensagens	25
4.1 Descrição dos experimentos	25
4.1.1 <i>Hardware</i> e <i>software</i> usados nos experimentos	28
4.1.2 Questões de medição, configuração e execução	28
4.2 Resultados dos Experimentos	30
4.2.1 <i>Benchmark 1</i>	30
4.2.2 <i>Benchmark 2</i>	31
4.2.3 <i>Benchmark 3</i>	32

4.2.4	<i>Benchmark 4</i>	35
4.3	Considerações sobre o capítulo	39
5	Aplicando o resultado da análise de escalabilidade na construção de um MAS escalável	41
5.1	Uma proposta de MAS escalável	42
5.2	Projeto e Implementação	43
5.3	Testes de escalabilidade da solução proposta	45
6	Conclusões	47
6.1	Sugestões para Pesquisas Futuras	48
	Referências Bibliográficas	49

Lista de Abreviaturas

A&A	Agents & Artifacts
ACL	Agent Communication Language
AID	FIPA Agent Identifier
AMS	Agent Management System
avgRTT	Average Round Trip Time
API	Application Programming Interface
BDI	Believe, Desires, Intentions
CARTAgO	Common ARTifact infrastructure for AGents Open environments
CPU	Central Process Unit
DF	Directory Facilitator
DNS	Domain Name System
FIPA	Foundation for Intelligent Physical Agents
GPU	Graphics Processing Unit
IMTP	Internal Message Transport Protocol
IoT	Internet of Things
JaCaMo	Jason, CARTAgO, MOISE+
JADE	Java Agent DEvelopment Framework
JDK	Java Development Kit
JIAC	Java-based Intelligent Agent Componentware
JVM	Java Virtual Machine
KQML	Knowledge Query and Manipulation Language
LAN	Local Area Network
MAOP	Multi-agent Oriented Programming
MAS	Multi-agent System
MASaaS	Multi-agent System as <i>a</i> Service
MOM	Message Oriented Middleware
NLP	Natural Language Processing
NLU	Natural Language Understanding
UDP	User Datagram Protocol
QoS	Quality of Service
RMI	Remote Method Invocation
RPA	Robotic Process Automation
SCADA	Supervisory Control and Data Acquisition
SOA	Service-Oriented Architecture
SPADE	Smart Python Agent Development Environment
TCP	Transmission Control Protocol
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Lista de Figuras

2.1	Processo de raciocínio prático na arquitetura BDI - adaptada de [UFR19]	7
2.2	Arquitetura do JADE distribuída - adaptada de [Ita19]	10
2.3	Distribuição de clientes e servidores Jabber/XMPP	11
2.4	Componentes básicos de JIAC e sua relação estrutural - adaptada de [Tea18]	13
4.1	Comunicação no cenário intraplataforma com o servidor Jabber	27
4.2	Comunicação no cenário interplataforma com o servidor Jabber	27
4.3	<i>Benchmark</i> 1: Cada <i>host</i> possui um par de agentes Destinatário de Remetente. Um agente Remetente envia 1000 mensagens para cada agente Destinatário	30
4.4	<i>avgRTTs</i> em LAN para o <i>Benchmark</i> 1 (em milisegundos)	31
4.5	<i>Benchmark</i> 2: N agentes Remetentes enviam cada um 1000 mensagens para o agente Destinatário. Remetentes estão distribuídos em dois <i>hosts</i> diferentes do <i>host</i> do Destinatário	32
4.6	<i>avgRTT</i> para o <i>Host</i> 2 em LAN	33
4.7	<i>avgRTT</i> para o <i>Host</i> 3 em LAN	33
4.8	<i>Benchmark</i> 3: N agentes Remetentes enviam cada um 1000 mensagens para cada um dos três agentes Destinatários. Remetentes estão distribuídos em dois <i>hosts</i> diferentes do <i>host</i> dos Destinatários	34
4.9	<i>avgRTT</i> no cenário intraplataforma	34
4.10	<i>avgRTT</i> no cenário interplataforma	35
4.11	<i>Benchmark</i> 4 em localhost: Pares de agentes Destinatários e Remetentes. Cada agente Remetente envia 1000 mensagens para o seu respectivo agente Destinatário	36
4.12	<i>Benchmark</i> 4 em LAN: Pares de agentes Destinatários e Remetentes. Cada agente Remetente envia 1000 mensagens para o seu respectivo agente Destinatário	36
4.13	<i>avgRTT</i> no cenário intraplataforma	37
4.14	<i>avgRTT</i> no cenário interplataforma	38
5.1	Etapas de distribuição de eletricidade - especificação funcional [EY18]	42
5.2	Arquitetura do protótipo da solução	44

Lista de Tabelas

2.1	Comparação de Plataformas MAS	14
3.1	Síntese dos Trabalhos Relacionados	23
4.1	Plataformas MAS avaliadas	26
4.2	<i>Hardware</i> e <i>software</i> usados nos experimentos (Setup 1).	28
4.3	<i>Hardware</i> e <i>software</i> usados nos experimentos (Setup 2 - Host 1).	28
4.4	<i>Hardware</i> e <i>software</i> usados nos experimentos (Setup 2 - Host 2).	28
4.5	<i>Hardware</i> e <i>software</i> usados nos experimentos (Setup 2 - Host 3).	29
4.6	Módulos do servidor Prosody IM Jabber/XMPP [IM19]	30
4.7	<i>avgRTTs</i> em LAN para o <i>Benchmark 1</i> (em milisegundos).	31
4.8	<i>avgRTTs</i> em LAN para o <i>Benchmark 2</i> (em milisegundos).	32
4.9	<i>avgRTTs</i> em LAN para o <i>Benchmark 2</i> (em milisegundos).	32
4.10	<i>avgRTTs</i> em LAN para o <i>Benchmark 3</i> (em milisegundos).	33
4.11	<i>avgRTTs</i> em LAN para o <i>Benchmark 3</i> (em milisegundos).	35
4.12	SPADE <i>avgRTT</i> em Localhost e LAN (em milisegundos).	37
4.13	SPADE(L) <i>avgRTT</i> em Localhost e LAN (em milisegundos).	37
4.14	JADE <i>avgRTT</i> em Localhost e LAN (em milisegundos).	38
4.15	ASTRA <i>avgRTT</i> em Localhost e LAN (em milisegundos).	38
4.16	JIAC V <i>avgRTT</i> em Localhost e LAN (em milisegundos).	39
5.1	<i>Hardware</i> e <i>Software</i> usados nos testes do protótipo.	45

Capítulo 1

Introdução

Imagine um cenário onde uma empresa de distribuição de energia elétrica necessita automatizar parte de suas operações de atendimento ao cliente. Tal empresa possui um sistema de informação distribuído que apoia sua operação atual. Para melhorar seus processos e reduzir custos, a empresa espera integrar ao sistema existente a solução de serviço de atendimento ao cliente. Tal serviço possui duas vertentes: atender demandas oriundas diretamente da interação do cliente com o sistema (solicitação de 2a. via de conta, reclamação de falta de energia, solicitação de religação de energia, etc) e antecipar demandas do cliente, a partir de alguma falha no fornecimento de energia. A ideia é atuar proativamente na ocorrência de falha, contactando o cliente e informando-lhe que foi detectado o problema, quais providências foram tomadas e qual o tempo esperado até a solução. Por um lado o cliente/usuário usa uma interface *Web* para solicitar serviços e o sistema pode usar *chatbots* para contactar os clientes.

Este é o nosso cenário de trabalho: uma situação real que requer uma solução científico-tecnológica adequada, visto que o mercado não dispõe de uma solução inteligente e eficiente para o problema. Abstraindo o domínio, o problema que se coloca é o de integração de serviços inteligentes a um sistema de informação distribuído existente. Tal integração deve escalar mantendo a eficiência do sistema. Várias questões relacionadas à escalabilidade e eficiência em sistemas distribuídos ainda sucintam pesquisa e permanecem em aberto [vST17].

Sistemas de informação distribuído representam uma evolução de sistemas tradicionais de forma a modularizar e distribuir seus componentes a fim de que diversas aplicações possam fazer uso de suas partes [vST17]. Essas aplicações de uma forma geral reduzem custos e melhoram a eficiência dos processos de gerenciamento de informação e produção. Essa modularidade é desejável uma vez que divide tarefas administrativas em partes menores, reduzindo a complexidade e permitindo a rápida adaptação às necessidades de mudança do negócio. Para tanto, muitas das tecnologias empregadas demandam interoperabilidade com outros sistemas, isto é, a capacidade de conversar entre si. Além disso, outra questão que sistemas de informação distribuídos precisam lidar é a integração entre os dados, especialmente considerando a consistência de suas bases.

Um dos desafios desses sistemas está relacionado a escalabilidade. Um sistema escalável possui a capacidade de ser ampliado para acomodar o seu crescimento sem perda notável de desempenho [vST17]. Existe uma relação entre crescimento do sistema e sua complexidade, uma vez que o aumento do tamanho impacta em maior complexidade. A questão é projetar sistemas que possam crescer dinamicamente sem que o desempenho seja afetado. Para tanto, um outro aspecto importante é a flexibilidade, uma vez que esse sistema dinâmico necessita de integração com outras partes de forma não estática, ou seja, são também interoperáveis. Sistemas interoperáveis têm a capacidade de se comunicar com outras partes, isto é, transferir dados ou executar aplicações entre si, através de padrões e especificações acordados.

Agentes podem ser descritos como entidades relativamente independentes e autônomas que podem ser usadas para resolver problemas de diferentes graus de complexidade [Fer99]. Essas entidades também podem assumir diferentes graus de complexidade, variando de agentes puramente reativos a agentes capazes de raciocinar de forma inteligente (agentes proativos) [RN03]. Devido às

suas capacidades autônomas, interativas e sociais, os agentes podem ser vistos como um paradigma para a engenharia de *software* [Jen00]. O projeto de um agente pode seguir padrões arquiteturais distintos, como a arquitetura de subsunção [Bro86] para agentes reativos ou arquitetura BDI (Belief-Desire-Intention) [UFR19] para agentes proativos.

Agentes podem ser organizados em comunidades e trabalhar juntos usando diferentes mecanismos de interação para resolver problemas complexos. Essas comunidades também são conhecidas como sistema multiagente (MAS) ¹: um sistema composto de vários agentes que interagem entre si em um único ambiente [RN03]. O principal propósito de um MAS é resolver autonomamente problemas complexos que de outra forma seriam impossíveis (ou muito difíceis) de se resolver usando um sistema monolítico [Fer99]. Como MAS é inerentemente distribuído, ele também pode ser usado no contexto de sistemas complexos. Os sistemas complexos são freqüentemente compostos de subsistemas inter-relacionados, orientados por uma hierarquia estruturada, às vezes não estática [Sim96]. MAS podem ser usados para reduzir a complexidade de controle do sistema e o grau de acoplamento entre os componentes. Assim, cada agente pode agir para maximizar o resultado esperado e se adaptar a contingências inesperadas [Jen00]. Para desenvolver e implantar MAS, usualmente fazemos uso de plataformas multiagente.

Uma plataforma multiagente é um *software* ou *framework* capaz de gerenciar múltiplos aspectos da execução de agentes e suas interações. Muitas plataformas MAS foram desenvolvidas nos últimos anos [KB15], todas elas compatíveis com padrões de desenvolvimento de sistemas interoperáveis em diferentes níveis. A FIPA (Foundation for Intelligent Physical Agents) ², por exemplo, é um esforço para estabelecer diretrizes para o desenvolvimento e execução de plataformas interoperáveis. Numa associação entre MAS e sistemas de informação distribuídos, plataformas multiagente exercem funções compatíveis aos *middleware*.

Considerando o cenário descrito acima e as características atribuídas a MAS, considera-se que a adoção de uma solução multiagente parece adequada para resolver o problema subjacente. Porém, antes de adotá-la é necessário analisar questões relacionadas à escalabilidade e desempenho destes sistemas, em particular no que se refere à troca de mensagens entre os agentes.

Neste trabalho, escalabilidade está relacionada a degradação do desempenho à medida em que o sistema cresce (em quantidade de agentes e/ou mensagens trocadas entre agentes). Para tanto, consideramos particularmente o mecanismo adotado na camada de transporte de mensagens em plataformas MAS para avaliação. A comunicação é parte essencial em um MAS, uma vez que agentes muitas vezes precisam se coordenar para atingir seus objetivos. Nesse sentido, adotamos como métrica de avaliação o *avgRTT* (*average round trip time*) e aplicamos os *benchmarks* propostos em [SAM⁺07].

Este trabalho, portanto, consiste na análise de plataformas multiagente com relação à implantação de MAS de larga escala. Para tanto, vamos selecionar plataformas existentes e avaliar sua escalabilidade e desempenho tendo em vista os sistemas de transporte de mensagens. O objetivo é fornecer informações tanto para desenvolvedores de plataformas, como também para desenvolvedores de sistemas que desejam utilizar plataformas MAS para apoiar o desenvolvimento de aplicações em larga escala.

1.1 Objetivo

O objetivo deste trabalho é avaliar algumas das plataformas MAS atualmente ativas [CQV02, GCB06, LKK⁺13, CRL15] para implantar sistemas em larga escala baseados em agentes [GC04, TRBC09, Huh02]. Para tanto, são avaliadas as características de cada plataforma bem como o desempenho de sua camada de transporte de mensagens.

¹Do acrônimo em inglês Multi-Agent System

²<http://www.fipa.org/>

1.2 Contribuições

Como contribuições para este trabalho, espera-se prover uma avaliação de escalabilidade e desempenho da camada de transporte de mensagens de plataformas MAS existentes.

A partir das avaliações realizadas, é esperado que desenvolvedores possam obter conhecimento sobre a capacidade atual das plataformas multiagente, bem como sua capacidade operacional para soluções em larga escala. Como as plataformas selecionadas possuem particularidades, o intuito é que com as avaliações, a escolha da plataforma mais adequada para novos projetos seja facilitada.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

O Capítulo 2 apresenta conceitos e fundamentos necessários para o entendimento da proposta. Assim, na Seção 2.1, são fornecidas informações sobre agentes inteligentes e as linguagens de comunicação de agentes. Na Seção 2.3, são apresentadas as plataformas multiagente usadas nos experimentos, bem como os critérios adotados para escolher essas plataformas para comparação. Na Seção 2.4 são brevemente discutidas algumas perspectivas sobre o conceito de escalabilidade em MAS.

O Capítulo 3 começa descrevendo artigos que definem *benchmarks* para avaliação de desempenho e escalabilidade em MAS (3.1). Em seguida, nas seções 3.2 e 3.3 descreve-se o trabalho existente relacionado à avaliação de desempenho e escalabilidade em plataformas multiagente. Finalmente, na seção 3.4 são apresentadas aplicações existentes na literatura que utilizam agentes proativos para soluções de grande porte.

O capítulo 4 se inicia com uma descrição dos experimentos realizados para avaliação de escalabilidade e desempenho na camada de comunicação de plataformas multiagente (seção 4 seguida da apresentação dos resultados obtidos (seção 4.2). O capítulo é finalizado com uma discussão sobre tais resultados, sua validade e eventuais ameaças a validade.

A partir dos resultados obtidos nos experimentos realizados, adotou-se a plataforma com melhor desempenho para o desenvolvimento de uma solução multiagente para cenário inicialmente proposto, o que é apresentado no capítulo 5. Assim, na Seção 5.2 o cenário inicialmente proposto é revisitado, agora detalhando melhor o problema concreto, bem como é descrito um protótipo da solução.

Finalmente, são apresentadas as conclusões no Capítulo 6, bem como perspectivas de trabalhos futuros.

Capítulo 2

Agentes e Sistemas Multiagente

2.1 Agentes Inteligentes e Sistemas Multiagente

Um agente é um sistema de computador com habilidades autônomas, capaz de tomar decisões sobre o que é necessário fazer para atingir seus objetivos [Woo09]. Embora essa definição não seja consensual, ela gira em torno do conceito de autonomia. Algumas das características dos agentes situados em um ambiente (físico ou virtual) [WJ95, BHW07] são:

- **Autonomia** autonomia está relacionada a capacidade do agente de construir objetivos e executar ações para que estes objetivos sejam satisfeitos
- **Proatividade** significa ser capaz de exibir o comportamento direcionado por objetivos. Se um agente tem um objetivo, então espera-se que este agente execute algumas ações para alcançá-lo. Esse comportamento é diferente, por exemplo, de objetos em Java que precisam ser invocados (comportamento passivo).
- **Reatividade** significa ser capaz de oferecer respostas imediatas decorrentes de mudanças ou estímulos no ambiente. Um agente reativo é capaz de reagir a essas mudanças e adaptar suas ações de acordo para atingir seus objetivos.
- **Habilidade Social** refere-se não apenas a comunicação ou invocação de métodos, mas também a capacidade de trocar conhecimento com outros agentes. No caso dos agentes BDI (explicado a seguir), essa troca de conhecimento é ilustrada pelo compartilhamento de crenças e objetivos.

Agentes são capazes também de interagir com outros agentes, trocando dados e participando de interações complexas, como cooperação, coordenação e negociação. Um sistema multiagente (MAS, do acrônimo em inglês) é um sistema composto por múltiplos agentes executando em um ambiente [RN03]. Tais sistemas têm como objetivo principal a resolução de problemas complexos que seriam de difícil solução (ou sem solução possível) usando um sistema monolítico. Portanto, um MAS composto por agentes proativos é intrinsecamente adaptável, considerando suas características inerentes, além de autonomia, reatividade e habilidade social.

Agentes inteligentes são capazes de raciocinar e decidir qual ação executar de acordo com as informações disponíveis. Essas informações geralmente estão relacionadas às condições ou consequências das ações a serem executadas [Woo09]. Aproveitar as capacidades inerentes aos agentes permite que o MAS se adapte às mudanças no ambiente. Além disso, o uso de múltiplos agentes permite que mecanismos de cooperação ocorram em situações em que a solução do problema em questão está além das capacidades de cada agente do sistema isoladamente.

Cooperação em MAS pode emergir a partir das interações entre os agentes que compõe o sistema, como também pode ser definida em tempo de projeto [Fer99]. A definição de mecanismos de coordenação de agentes em MAS pode ser estabelecida pelo uso de modelos organizacionais, que estabelecem restrições ao comportamento de agentes que compõe o sistema. O uso de tais modelos faz com que o sistema seja considerado uma organização, onde agentes desempenham papéis

que estão sujeitos a restrições de comportamento, seja pela definição de normas organizacionais, por relações hierárquicas entre papéis, entre outras estratégias [HSB07, FGM03, AD10]. Aplicações de MAS incluem controle de fabricação, planejamento de produção, logística, transporte, entre outros [PM08].

Agentes possuem características distintas de outras abstrações usadas para desenvolvimento de *software*. Em particular, questões relacionadas a autonomia e proatividade impulsionaram a definição de arquiteturas para guiar o desenvolvimento de agentes. Assim, algumas arquiteturas foram definidas para casos particulares de agentes, desde agentes reativos a serem embutidos em robôs (arquitetura de subsunção) até agentes deliberativos, que raciocinam para definir quais ações executar de forma a atingir seus objetivos (arquitetura BDI).

A arquitetura de subsunção [Bro86] é uma arquitetura em camadas especialmente indicada para desenvolver agentes reativos. Essa arquitetura foi originalmente desenhada para controlar o comportamento inteligente de robôs. Na arquitetura de subsunção os comportamentos são organizados em camadas paralelas e independentes. Em um robô, por exemplo, cada camada é responsável por uma de suas atividades, o que permite a realização de atividades concomitantes. As camadas são organizadas de forma hierárquica, sendo assim, camadas superiores tem prioridade sobre camadas inferiores na execução, o que pode ocasionar a supressão de um determinado comportamento.

A arquitetura BDI (Belief-Desire-Intention) [RG91] é uma das arquiteturas de *software* usadas para modelar e implementar agentes inteligentes. Baseia-se no modelo de raciocínio prático humano [Bra87], usando os conceitos de crença, desejo e intenção:

- **Crenças:** representam o que o agente acredita ser verdade sobre o ambiente em que está situado. As crenças dos agentes também podem incluir informações sobre si mesmos e sobre outros agentes em execução no mesmo ambiente, bem como regras de inferência que permitem a dedução de novas crenças a partir das crenças existentes. Em outras palavras, as crenças representam o conhecimento do agente.
- **Desejos:** representam o estado motivacional do agente. Desejos são estados ou objetivos que o agente gostaria de realizar eventualmente.
- **Intenções:** representam o estado deliberativo do agente - um desejo que o agente se comprometeu a alcançar. Em outras palavras, uma intenção pode ser definida como uma "escolha com compromisso" [CL90].

Esses conceitos são usados por agentes em um processo de raciocínio meio-fim. Nesse sentido, as ações do agente são organizadas em um plano de execução construído sobre o que ele acredita ser verdade e o que ele deseja alcançar como meta. Esse processo de raciocínio é geralmente chamado de "planejamento" [KN80]. Usando essa separação entre informações, planejamento e execução, o agente é capaz de se adaptar às diferentes condições apresentadas pelo ambiente em que está situado. O processo de raciocínio prático na arquitetura BDI (ver Figura 2.1) possui sete componentes básicos definidos em [Woo99]:

- Um conjunto de crenças atuais, que representa as informações que o agente tem sobre seu ambiente;
- Uma função de revisão de crenças chamada *frc*, que a partir de uma entrada perceptual e as crenças atuais do agente, e com base nelas, determina um novo conjunto de crenças;
- Uma função de geração de opções, (gera opções), que determina as opções disponíveis ao agente (seus desejos), com base em suas crenças atuais sobre seu ambiente e suas intenções atuais;
- Um conjunto de desejos atuais, representando possíveis cursos de ações disponíveis para o agente;

- Uma função de filtro (filtro), que representa o processo de deliberação do agente, e que determina as intenções do agente com base em suas crenças, desejos e intenções atuais;
- Um conjunto de intenções atuais, representando o foco atual do agente - os estados de coisas que ele se comprometeu a tentar fazer;
- Uma função de seleção de ação (ações), que determina uma ação a ser executada com base nas intenções atuais.

Figura 2.1: Processo de raciocínio prático na arquitetura BDI - adaptada de [UFR19]

2.2 Linguagens de Comunicação de Agentes

Uma Linguagem de Comunicação de Agente (ACL) estabelece uma estrutura para as mensagens trocadas entre agentes, tanto em tipo quanto em significado. No entanto, os agentes não trocam mensagens simplesmente; eles realmente se engajam em conversas [Lab01]. Uma conversa pode ser vista como um protocolo pré-organizado ou padrão de troca de mensagens, orientado para uma tarefa ou objetivo específico.

As duas Linguagens de Comunicação de Agentes (ACLs) mais usadas são [LK13, KPK18]: FIPA-ACL [ON98] e KQML [FFMM94]. Ambas linguagens são baseadas na teoria dos atos de fala [SS69] e são compostas de diferentes *performativas*¹. As performativas podem ser entendidas como sentenças que descrevem um ambiente e o influenciam ao mesmo tempo. Como tal, as mensagens trocadas entre os agentes podem representar ações ou atos comunicativos. Apesar de serem relativamente antigos, esses padrões ainda estão sendo usados atualmente [RABA18, BdSW18].

FIPA-ACL é composta por 22 performativas de comunicação. As performativas mais comuns são:

- **inform:** O remetente informa ao receptor que uma determinada proposição é verdadeira;

¹<https://stanford.library.sydney.edu.au/entries/speech-acts/>

- **request**: O remetente solicita que o receptor execute alguma ação;
- **agree**: O remetente concorda em executar alguma ação, possivelmente no futuro;
- **not understood**: O remetente (por exemplo, *i*) informa ao destinatário (por exemplo, *j*) que percebeu a ação executada por *j*, mas não a compreendeu;
- **refuse**: O remetente se recusa a executar uma determinada ação, explicando o motivo da recusa.

Embora FIPA-ACL estabeleça apenas protocolos de interação entre agentes, também há especificações da FIPA para o gerenciamento de agentes usados em conjunto com os padrões de comunicação. Os componentes básicos dessas especificações são (i) a *Agent Platform*, na qual cada agente possui um identificador exclusivo chamado AID (FIPA Agent Identifier); (ii) o *Agent Management System* (AMS), responsável pelo gerenciamento da *Agent Platform*; e (iii) o *Directory Facilitator* (DF), que permite aos agentes publicar serviços de forma detectável.

Da mesma forma, a linguagem KQML também é composta por performativas de comunicação. Possui três camadas diferentes:

- **Camada de conteúdo**: onde o conteúdo real da mensagem reside, na própria linguagem de representação do programa no computador;
- **Camada de comunicação**: usada para codificar parâmetros de comunicação de nível inferior, como a identidade do remetente e do destinatário da mensagem;
- **Camada de mensagem**: fornece as performativas usadas no processo de comunicação entre agentes aderentes ao KQML.

O KQML também descreve uma classe especial de agentes denominada *facilitators*. O *facilitator* é opcional, e quando presente é responsável por executar vários serviços de comunicação, como manter um registro de serviços, rotear mensagens para esses serviços com base no conteúdo e fornecer mediação entre as partes de comunicação. Diferentemente do FIPA-ACL, essas especificações estão contidas na definição da ACL e não dependem de padrões adicionais.

Apesar de serem semelhantes em vários aspectos, existem algumas características particulares de uma ou outra ACL que tornam seu uso dependente do domínio. O gerenciamento de agentes e os recursos de comunicação são incorporados na camada de mensagens no KQML, enquanto o FIPA-ACL os considera como serviços fornecidos por outros agentes ou estruturas. Essas diferenças, no entanto, podem ser cruciais na escolha de uma ou outra ACL de acordo com um conjunto específico de requisitos do sistema. Considerando o contexto do presente trabalho, por exemplo, o fato de ter um diretório de serviços como o fornecido pelo FIPA, faz com que as especificações auxiliares estejam de acordo com a Arquitetura Orientada a Serviços (SOA) atualmente usada por sistemas baseados em *Web Services* [Erl08]. Para tais aplicações, também há um benefício do fato de que a semântica da especificação FIPA-ACL é mais clara do que a do KQML [Lab01].

2.3 Plataformas Multiagente

Nesta Seção, são apresentadas cinco plataformas *ativas* para o desenvolvimento e a execução de um MAS: JADE (Java Agent DEvelopment Framework) ², SPADE (Smart Python multi-Agent Development Environment) ³, JaCaMo (Jason, CArtaGO e MOISE+) ⁴, ASTRA ⁵ e JIAC (Java-based Intelligent Agent Componentware) V ⁶.

²<http://jade.tilab.com/>

³<https://pypi.org/project/SPADE/>

⁴<http://jacamo.sourceforge.net/>

⁵<http://astralanguage.com/wordpress/>

⁶<http://www.jiac.de>

JADE e SPADE possuem múltiplos pontos em comum - especialmente considerando a aderência às especificações do padrão FIPA ⁷. JaCaMo por sua vez está mais atrelado a arquitetura BDI e a linguagem AgentSpeak(L), podendo fazer uso da arquitetura do JADE no que se refere a execução distribuída e comunicação usando FIPA-ACL, tornando-se assim aderente às especificações FIPA. ASTRA possui uma comunicação baseada em FIPA-ACL, particularmente no que se refere ao uso das performativas, enquanto JIAC V não adota nenhuma padronização tanto para comunicação quanto para organização. A especificação FIPA é um esforço para estabelecer diretrizes para a interoperabilidade de plataformas multiagente. Uma plataforma é compatível com o FIPA se possui as seguintes estruturas:

- **Agent Identifier (AID):** Identificador único de um agente na plataforma.
- **Agent Management System (AMS):** O AMS funciona como páginas brancas em um catálogo de telefonia, listando todos os agentes presentes na plataforma. O AMS também é responsável pela criação, exclusão e migração de agentes na plataforma.
- **Directory Facilitator (DF):** O DF funciona como a página amarela para serviços, catalogando e listando todos os serviços que podem ser usados pelos agentes, oferecidos por outros agentes. O DF é opcional na especificação FIPA, mas se estiver presente, deve ser implementado como um serviço DF.

Além disso, as plataformas compatíveis com FIPA devem suportar o FIPA-ACL. O JADE é totalmente compatível com todas as especificações do FIPA, assim como SPADE em sua versão 2, enquanto o SPADE 3 e ASTRA aderem apenas às especificações de comunicação (FIPA-ACL).

Outra característica que é comum a JADE, SPADE, JaCaMo e também a ASTRA é que possuem estruturas para implementar agentes com a arquitetura BDI ⁸. Essa arquitetura permite que os agentes encapsulem propósitos específicos (ações) que podem ser compostos e reutilizados por meio de interações. Outra característica importante, especialmente considerando expor agentes como um serviço é o suporte a ontologias, para transferir, reutilizar e consumir conhecimento. Nesse sentido, tanto JADE, JIAC V quanto SPADE oferecem suportes para uso de ontologias em diferentes níveis.

2.3.1 JADE

JADE (Java Agent Development Framework), atualmente na versão 4.5, é um *middleware* distribuído escrito em Java completamente aderente às especificações FIPA, ao mesmo tempo em que requer um conhecimento mínimo sobre a teoria de agentes. JADE usa as especificações FIPA para comunicação, gerenciamento e arquitetura do agente.

Cada agente no JADE (ver Figura 2.2) é hospedado por um *Container* e uma *Plataforma* pode conter vários *Containers*. O *Main Container* é responsável por fornecer as funcionalidades essenciais da especificação FIPA.

O mecanismo de entrega de mensagens no JADE depende da localização do *remetente* e do *destinatário* e é otimizado para minimizar o tempo de entrega das mensagens. Se ambos estiverem em execução no mesmo *host*, JADE utilizará seus protocolos internos de transporte de mensagens (IMTPs). Neste caso, o objeto da mensagem será clonado e a referência do clone será passada para o *destinatário*.

Caso contrário, se *remetente* e *destinatário* estiverem em contêineres diferentes, o objeto de mensagem será serializado e o Java RMI (Remote Method Invocation) ⁹ é usado.

Sendo totalmente compatível com a FIPA, JADE usa o protocolo FIPA não apenas para comunicação do agente, mas também para gerenciamento. A especificação FIPA detalha uma infraestrutura de gerenciamento de agentes (denominada *Agent Platform*) e uma plataforma de publicação de serviços (denominada *Directory Facilitator*). Uma *Agent Platform* é o ambiente em que o

⁷<http://www.fipa.org/>

⁸A arquitetura BDI para o JADE está disponível em <http://www.inf.ufrgs.br/prosoft/bdi4jade/>

⁹<http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html>

Figura 2.2: Arquitetura do JADE distribuída - adaptada de [Ita19]

agente é executado. Nesse ambiente, cada agente é identificado exclusivamente por meio do uso de um identificador exclusivo (AID). O *Directory Facilitator*, por outro lado, permite que os agentes publiquem serviços visíveis para outros agentes.

2.3.2 SPADE

O SPADE (Smart Python Agent Development Environment), atualmente na versão 3.1.3, é uma estrutura de agente escrita em Python e compatível com os padrões FIPA para comunicação do agente. O SPADE estava totalmente em conformidade com as especificações FIPA até a versão 2. A partir da versão 3, o SPADE concentra-se mais nos recursos oferecidos pela tecnologia XMPP (Extensible Messaging and Presence Protocol).

Esta plataforma é promissora do ponto de vista do desenvolvimento de agentes orientados à serviços, devido à sua própria natureza. De fato, o SPADE usa o *framework*/protocolo XMPP (Figura 2.3) como camada de comunicação, proporcionando interoperabilidade com outras entidades na Internet. O XMPP é um protocolo cliente-servidor, executado em protocolo TCP e projetado para comunicação em tempo real. Os servidores suportam um grande número de usuários e mensagens com base em XML (Extensible Markup Language) [GCB06, KCVGAZ15].

Uma das características relevantes deste protocolo empregado no SPADE é a *notificação de presença e conferência multiusuário*. A notificação de presença permite que um agente detecte a presença e a disponibilidade de outros agentes. A conferência multiusuário permite a criação de um canal protegido, permitindo a troca de mensagens entre um grupo específico de agentes.

Para executar tarefas, o SPADE possui alguns padrões de execução (chamados *behaviours*), também presentes na plataforma JADE, que são:

- **Cyclic**: Executa a tarefa repetidamente;
- **One-Shot**: Executa uma tarefa apenas uma vez;

Figura 2.3: Distribuição de clientes e servidores Jabber/XMPP

- **Periodic:** Aciona a execução da tarefa em um tempo t predeterminado;
- **Finite State Machine:** Composição de tarefas acionadas a partir de estados e transições.

2.3.3 JaCaMo

O *framework* JaCaMo (atualmente na sua versão 0.8) para programação de sistemas multiagente é constituído de três ferramentas: Jason [BHW07] que é um interprete da linguagem de programação de agente AgentSpeak(L) [Rao96], CArtAgO [RVO06] e MOISE+ [HSB07]). Usando o JaCaMo como plataforma multiagente, um MAS apresenta uma separação de preocupações relacionadas a agente, organização, ambiente e interação.

Jason e a linguagem AgentSpeak(L)

A plataforma Jason [BHW07] é um interpretador de uma extensão da linguagem AgentSpeak, conhecida como AgentSpeak(L), baseada na arquitetura BDI (*Believes, Desires, Intentions*) [Woo09]. Jason foi construído de forma a direcionar o comportamento do agente, sem que o programador tenha de se preocupar com isso.

A plataforma Jason é baseada na arquitetura BDI, portanto as crenças no Jason tem a função de representação do conhecimento. Um agente nessa arquitetura está constantemente percebendo o ambiente, alterando sua base de conhecimento, raciocinando sobre como agir de modo a atingir seus objetivos, e em seguida, agindo de modo a mudar o ambiente. A parte de raciocínio (prático) do comportamento cíclico do agente na linguagem AgentSpeak é feita de acordo com os planos que o agente tem em sua biblioteca de planos.

Um agente em AgentSpeak tem uma base de crenças, que expressam propriedades que o agente acredita serem *verdadeiras* do mundo em que está situado. Os objetivos expressam as propriedades dos estados do mundo que o agente deseja alcançar. Embora isso não seja imposto pela linguagem AgentSpeak, normalmente, ao representar um objetivo g em um programa de agente, significa que o agente está comprometido a agir de modo a mudar o mundo para um estado no qual g se torne realmente verdade [BHW07]. A terceira construção essencial de um programa de um agente seguindo

a arquitetura BDI são os planos, que ditam como um objetivo deve ser satisfeito. Alterações em crenças e em objetivos desencadeiam esses planos.

CARTAgO

O *framework* CARTAgO (Common ARTifact infrastructure for AGents Open environments) [RVO06] é baseado em agentes e artefatos (A & A) para modelar e projetar sistemas multiagente. Com esta ferramenta, pode-se criar artefatos estruturados em espaços abertos onde os agentes podem se unir para trabalhar em conjunto. Os artefatos são recursos e ferramentas, construídos dinamicamente, manipulados e usados por agentes para apoiar/realizar suas atividades individuais e coletivas.

Os principais elementos utilizados no CARTAgO são as propriedades observáveis, operações e sinais. Propriedades observáveis são elementos do artefato que o agente passa a observar de forma que qualquer operação sobre esse elemento será percebida. As operações são implementadas para fornecer uma interface para utilização dos recursos do artefato pelos agentes, com parâmetros de entrada e retorno (*OpFeedbackParam*). Finalmente, os sinais (*Signals*) permitem que seja detectado pelo agente uma alteração em sua base de crenças, alterando-a. *Signals* podem ser úteis para desencadear ações a partir de artefatos, ou mesmo adicionar algo ao conjunto de *verdades* que o agente acredita.

MOISE+

O modelo organizacional MOISE+ [HSB07] é uma ferramenta com intuito de modelar a organização de um MAS. Consiste na especificação de três dimensões: a estrutural, onde são definidos 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 deontica, que é a dimensão responsável pela definição de qual papel tem obrigação ou permissão para realizar cada missão.

2.3.4 ASTRA

ASTRA, atualmente na versão 3.6.2, é uma linguagem de programação de agentes baseada em AgentSpeak(L), que possui ambiente para sua execução. ASTRA possui múltiplos pontos em comum com Jason, especialmente considerando a linguagem adotada e ambas serem baseadas em Java. Além disso, ASTRA também oferece suporte à integração com o ambiente de desenvolvimento CARTAgO. Entre as características disponíveis em ASTRA, está a capacidade de executar em ambiente distribuído, onde a comunicação é suportada através de *Multicast* baseado no protocolo UDP (User Datagram Protocol).

Outra característica interessante é o uso dos chamados *Módulos* pelos agentes. Módulos são abstrações similares a artefatos CARTAgO, que oferecem aos agentes utilizarem suas funções tal como bibliotecas. Isso permite o desenvolvimento de *métodos* baseados em Java facilmente integráveis com agentes ASTRA. Esses métodos são invocados dependendo de um contexto, determinado por uma anotação. Atualmente ASTRA conta com 5 anotações:

- **@ACTION:** ações internas que retornam um valor booleano indicando se a ação foi executada com sucesso;
- **@TERM:** cálculos básicos que podem retornar um valor (inteiro, ponto flutuante, ...) que por sua vez podem ser usados como termos;
- **@FORMULA:** criam e retornam fórmulas lógicas no ASTRA (podem ser valores booleanos simples ou fórmulas mais complexas);
- **@SENSOR:** geram crenças que são adicionadas ao estado do agente;
- **@EVENT:** define tipos de eventos personalizados que podem ser usados no evento de acionamento de uma regra.

Figura 2.4: Componentes básicos de JIAC e sua relação estrutural - adaptada de [Tea18]

Em uma aplicação ASTRA, um agente chamado *Main* é geralmente responsável pela criação de outros agentes, ou seja, a inicialização de agentes é feita em tempo de execução. Além disso, o agente *Main* pode ser responsável por definir configurações importantes, como por exemplo, a camada de transporte de mensagens e a quantidade de *threads*¹⁰.

2.3.5 JIAC V

JIAC (Java-based Intelligent Agent Componentware), atualmente na versão 5.2.4, é uma arquitetura e um *framework* baseado em Java para desenvolvimento de agentes. Dentre as características listadas pelos desenvolvedores da plataforma estão a capacidade de ser empregada em larga escala, a distribuição, sua adaptabilidade e autonomia. Uma aplicação típica de JIAC consiste em *Agents* (agentes), *AgentNodes* e *AgentBeans*, detalhados a seguir:

- **Agents** são parte fundamental em uma aplicação em JIAC. Cada agente JIAC possui seu próprio *thread* de execução e dois componentes básicos: a *memória* onde são guardados dados em um espaço de tuplas, e *execução*, que é o controle de execução interno do agente;
- **AgentBeans** fornecem as funcionalidade básicas de comunicação e estende o agente no sentido de prover comportamentos e capacidades de forma modular;
- **AgentNode** é a infraestrutura de execução para os agentes. Nela está a descoberta de serviços e a camada de comunicação dos agentes. Normalmente existe um *AgentNode* por *host*.

Uma aplicação em JIAC pode ser composta por vários *AgentNodes*, que por sua vez podem conter vários agentes. Os *AgentBeans* são uma forma de modularizar as capacidades dos agentes, de forma que um agente é uma composição de um ou múltiplos *AgentBeans* (ver Figura 2.4).

JIAC apesar de seguir alguns princípios FIPA, não é aderente a essa especificação. A comunicação é baseada no ActiveMQ¹¹ e o *message broker* é o componente responsável por oferecer comunicação aos agentes em cenários intra e inter nodo. Um agente básico em JIAC possui dois métodos: `doStart()` onde capacidades essenciais para o agente são definidas, como exemplo o *Bean* para comunicação, e o método `execute()`, que o é o ciclo de execução do agente.

¹⁰Por padrão a quantidade de *threads* compartilhadas (*pool de threads*) pelos agentes é 2

¹¹<http://activemq.apache.org/>

A Tabela 2.1 faz uma comparação das capacidades das plataformas discutidas. Para tanto, elencamos a capacidade de oferecer suporte a arquitetura BDI, a aderência a especificação FIPA total ou parcial (sinalizada com *), a capacidade de execução múltiplas plataformas (sistemas operacionais), bem como o fornecimento de um ambiente capaz de criar e manipular ontologias.

Tabela 2.1: *Comparação de Plataformas MAS*

	JADE	SPADE	ASTRA	JIAC V	JaCaMo
Suporte BDI	✓	✓	✓		✓
Aderente a FIPA	✓	✓*	✓*		
Multiplataforma	✓	✓	✓	✓	✓
Ontologias	✓			✓	

2.4 Escalabilidade em MAS

No presente trabalho, o objetivo é explorar algumas características sobre escalabilidade em MAS. Para isso, é necessário definir o que é escalabilidade e como ela pode ser medida. Essa discussão é necessária, uma vez que os MAS são, muitas vezes, sistemas complexos que podem ser compostos por diferentes arquiteturas de agentes organizados de diferentes formas. Neste trabalho, a escalabilidade é a relação entre o desempenho quando o sistema é ampliado (em quantidade de agentes e/ou mensagens) e os recursos computacionais utilizados.

Deters em [Det01] aborda três aspectos relativos a escalabilidade em MAS: (i) número de agentes no sistema/*host*; (ii) consumo de recursos pelos agentes e (iii) número de mensagens. Em relação a número de agentes no sistema/*host*, Deters se refere especificamente a agentes proativos uma vez que sua execução intermitente consome mais recursos. Essa abordagem é similar a proposta neste estudo.

Segundo Deters, agentes reativos são invocados somente quando é necessário, por exemplo, ao receber uma mensagem. Já agentes proativos por sua vez são mais complexos do que agentes reativos, já que se exige planejamento. A proatividade tem um custo, já que agentes estão em um processo intermitente de deliberação, enquanto agentes reativos executam apenas quando são estimulados. Assim, um sistema composto por agentes proativos pode exceder o recurso de uma única máquina (ou *host*) facilmente.

Em [TJ00], Turner e Jennings abordam escalabilidade do ponto de vista do coletivo dos agentes. Os autores acreditam que um MAS é mais escalável se pode lidar com diferentes tamanhos de populações de agentes e de forma dinâmica. A métrica usada aqui para mensurar escalabilidade é a capacidade dos agentes em atingir suas metas coletivamente. Para tanto, a forma como os agentes se organizam e interagem é crucial para o bom desempenho do sistema.

Em [LNNDW98] são definidas conceitualmente três características ditas não funcionais de um MAS: estabilidade, escalabilidade e desempenho. Assim como Turner e Jennings, este trabalho relaciona a organização de um MAS com seu desempenho. A organização de um MAS está relacionada com sua coordenação, e uma organização não eficiente pode por exemplo, acarretar em uma sobrecarga de comunicação. Escalabilidade, por sua vez, assim como no presente trabalho é visto em como o desempenho do sistema degrada à medida em que a *sociedade* de agentes aumenta. Por fim, estabilidade é um conceito que pode ser definido como a capacidade do sistema em lidar com perturbações. Ou seja, um MAS é dito estável quando dada uma perturbação, seus indicadores de desempenho se mantêm.

Rana e Stout apontam brevemente em [RS00] a necessidade de diferenciar abordagens clássicas de avaliação de desempenho entre sistemas paralelos tradicionais e de MAS. Essa diferença baseia-se na granularidade e na complexidade desempenhada na execução de tarefas por agentes. Além disso, assim como os demais trabalhos, os autores consideram a organização social dos agentes e arquitetura usada com impacto no desempenho e escalabilidade de um MAS.

Este trabalho reside em avaliar a escalabilidade e desempenho dos mecanismos de comunicação de plataformas MAS. Esta abordagem é devida ao entendimento de que agentes precisam se coordenar para atingir seus objetivos e para tal, a comunicação se faz necessária. No entanto, considera-se de extrema importância avaliações que levam em consideração a organização de um MAS e a arquitetura de agentes. Porém, essas outras duas abordagens são restritas a problemas específicos, uma vez que cada solução demanda uma organização e um conjunto de regras próprios. Além disso, uma mesma solução pode adotar agentes que utilizam diferentes arquiteturas, com o uso das partes que compõe essa arquitetura em diferentes níveis. Sendo assim, este trabalho concentra-se em verificar limitações nas tecnologias para construir sistemas em larga escala utilizando agentes.

Capítulo 3

Trabalhos Relacionados

Neste capítulo são abordados trabalhos encontrados na literatura que possuem algum ponto de contato com o presente trabalho. Assim, começamos com trabalhos que apresentam *benchmarks* independentes de plataforma e de domínio e critérios para analisar a escalabilidade em MAS [SAM⁺07, BvSW⁺01], seguindo para trabalhos que analisam o desempenho de plataformas multiagentes de acordo com sua eficiência no transporte de mensagens [Det01, CQV02, Vrb03, MSA06, RCS18, CTG⁺04], passando por propostas para melhorar eficiência em MAS [WM15, ADF16, CDIPDPC15, PCGLO14, LSY09, TJ00] e fechando com MAS de larga escala [FGLO10, BCG15b, LW05, SMM⁺09]. A Tabela 3.1 faz uma síntese dos trabalhos discutidos nesse Capítulo.

3.1 Benchmarks e critérios para análise de escalabilidade e desempenho em MAS

Até onde conhecemos, [SAM⁺07] é o único trabalho que apresenta *benchmarks* independentes de plataforma e de domínio com o objetivo de analisar a escalabilidade de MAS. Em [SAM⁺07], os autores propõem quatro diferentes *benchmarks* para avaliar a eficiência e a escalabilidade de MAS em diferentes cenários. Os *benchmarks* propostos avaliam a capacidade da plataforma de gerenciar múltiplos *hosts*, sobrecarga de comunicação massiva e escalabilidade do sistema em geral. Enquanto a plataforma JADE foi utilizada como objeto de estudo, os *benchmarks* propostos não estão restritos a plataformas específicas. Foi adotado o mesmo *benchmark* para avaliação de escalabilidade no presente trabalho.

Brazier et al. em [BvSW⁺01] discutem algumas preocupações intrínsecas relacionadas à escalabilidade em MAS. Em particular, eles estabelecem algumas técnicas para lidar com MAS em grande escala, tais como: (i) técnicas de dimensionamento (ocultando latências de comunicação, técnicas de distribuição e replicação); (ii) gestão de agentes; e (iii) serviços escalonáveis. Ocultar as latências de comunicação refere-se à necessidade do sistema de abranger uma rede de área ampla. Assim, os agentes podem ser orientados para executar outras tarefas até que a resposta esperada seja entregue (se houver). Técnica de distribuição é sobre distribuir agentes entre *hosts*. A Internet é um exemplo de aplicação desta técnica, distribuindo milhões de páginas separadas por vários servidores. A replicação é uma técnica que coloca várias cópias de um agente na rede. A ideia subjacente é reduzir a latência colocando os dados perto de onde eles são usados, mas aumentando a necessidade de coordenação, porque as cópias devem ser consistentes (se necessário) na rede.

Embora a literatura acima não esteja estritamente relacionada a este trabalho, [SAM⁺07] fornece os *benchmarks* usados e [BvSW⁺01] estabelece a base para os experimentos realizados na Seção 4. Isto é verificado dado a opção de executar o MAS isoladamente para evitar a latência da comunicação e a adoção da camada de transporte de mensagens para analisar a eficiência das plataformas multiagente.

3.2 Desempenho e transporte de mensagens em MAS

Deters em [Det01] discute algumas características relacionadas a problemas na construção de MAS escaláveis. Neste trabalho, é apresentado um *middleware* chamado DICE baseado em CORBA/Java como uma camada para acessar recursos distribuídos para melhorar a escalabilidade nesses sistemas. Além disso, Deters também aponta gargalos no MAS em: (i) *host* do agente e (ii) roteamento de mensagens. *Host* do agente refere-se a executar agentes proativos na mesma máquina. Já que cada agente proativo precisa ter um *thread* para executar seus comportamentos, aumentar o número de agentes (*threads*) conseqüentemente aumenta os recursos necessários. Já o roteamento de mensagens é uma das características mais importantes de um MAS, uma vez que os agentes precisam se comunicar para se coordenar. Assim, um sistema de transporte de mensagens rápido e confiável é crucial para evitar comportamentos caóticos.

Semelhante ao discutido em [BvSW⁺01], Deters aponta alguns aspectos para melhorar a escalabilidade com uma abordagem mais focada em técnicas de programação. Esses aspectos incluem distribuição e replicação de componentes, e também o agendamento de agentes. O agendamento de agentes refere-se particularmente a otimizar a distribuição dos recursos esparsos entre um grande número de agentes. Isso pode ser alcançado com uma política para gerenciar a execução de agentes.

O *middleware* proposto para melhorar a escalabilidade oferece acesso transparente aos recursos. Para tanto, os recursos em uma máquina podem ser vistos como uma célula e uma máquina que possui mais recursos possui mais células. Esses recursos são gerenciados por um *provedor* que oferece recursos para um ou muitos *hosts*, e muitos *hosts* podem usar um ou vários provedores. Essa dependência é flexível e cada um desses componentes pode ser adicionado ou removido durante o tempo de execução. No entanto, esta técnica não é comparada com uma abordagem pura de qualquer MAS. Os resultados mostram que a estrutura cumpre seu próprio objetivo de oferecer uma alocação transparente de recursos.

Em [CQV02], o desempenho da camada de transporte de mensagens da plataforma JADE é comparado com o Java RMI. Sempre que os agentes de comunicação estavam no mesmo *Container* (comunicação intra-plataforma), o JADE apresentou melhor desempenho que o Java RMI. No entanto, quando os agentes de comunicação situados em diferentes *Containers* (comunicação interplataforma) JADE e Java RMI apresentaram desempenhos comparáveis. Além disso, assim como [CTG⁺04] os autores constataram que a sobrecarga introduzida pelo JADE em relação ao Java é aceitável, devido aos benefícios de um menor tempo de desenvolvimento de um MAS usando o JADE.

O objetivo de Mulet et al. [MSA06] era medir o *tempo médio de ida e volta* - *avgRTT* de mensagens em diferentes plataformas MAS baseadas em Java e em conformidade com o FIPA (JADE, MadKit ¹ e AgentScape [BMO⁺02]). O tempo de ida e volta (RTT) é o tempo que uma mensagem leva para ser enviada de um agente remetente para um agente destinatário, mais o tempo que leva para o agente remetente reconhecer que a mensagem foi recebida pelo agente destinatário. Neste estudo, para cada par de agentes, um *avgRTT* foi calculado a partir de uma coleção de 1000 mensagens enviadas. O tamanho dessas mensagens também foi levado em conta, variando de 1 byte a 100000 bytes e o número de pares de agentes fixados em 50 e 100. A localização dos agentes também foi um fator avaliado pela mudança do *kernel* (ou *Container* no JADE), bem como o *host* onde os agentes foram alocados. Os resultados mostram que o JADE apresenta mecanismos mais eficientes para a troca de mensagens entre agentes em diferentes cenários. Outro item avaliado nesses experimentos foi a resposta do sistema ao *Directory Facilitator* (DF) para o inscrição e cancelamento do registro de serviços, considerando agentes em diferentes *hosts*. No geral, o tempo de registro de um serviço não é influenciado pelo crescimento no número de serviços registrados. Nesse aspecto, o MadKit apresentou melhores resultados tanto no tempo de cadastro quanto na busca por um serviço cadastrado. Em uma análise geral, este trabalho mostra que a maneira como um serviço é implementado influencia seu desempenho. Oferecer um serviço através de agentes aumenta a modularidade do sistema, mas degrada seu desempenho em comparação com um serviço

¹<http://www.madkit.net/madkit/>

oferecido por meio de um *kernel*. Isso porque toda vez que um serviço é oferecido por agentes, eles precisam se comunicar por meio de um ACL, aumentando a carga do sistema.

Uma abordagem específica de domínio é mostrada em [Vrb03], onde os autores analisam a eficiência das camadas de transporte de mensagens das plataformas JADE, JACK, FIPA-OS e ZEUS em um cenário de controle de produção. Nessa análise, o *avgRTT* das mensagens e o uso de memória foram usados como métricas ao aumentar o número de agentes comunicantes no ambiente. JADE e JACK se destacaram entre as plataformas multiagentes, com JADE apresentando resultados significativamente melhores do que as outras plataformas de código aberto analisadas.

Chmiel et al. [CTG⁺04] concentraram sua atenção na escalabilidade de MAS usando a plataforma JADE. Eles realizaram experimentos focados na análise da camada de transporte de mensagens e na criação e/ou migração de agentes na plataforma, usando cenários próximos a situações de aplicações reais. Um dos cenários avaliados consistiu em agentes *spam* e agentes usuários. Os agentes *spam* e usuário foram criados em pares e os agentes *spam* transmitiam mensagens para todos os agentes usuários. O objetivo desse cenário era enfatizar o enfileiramento de mensagens recebidas pelos agentes usuários e observar a sobrecarga no canal de comunicação. Em outro cenário, os agentes precisavam passar por um número estabelecido de *hosts* diferentes, migrando de um para outro. A razão encontrada entre o número de *hosts* e o tempo decorrido para migrar entre todos eles é próximo de linear. Outro experimento de aplicação muito próximo do mundo real foi apresentado no estudo para avaliar a capacidade de instanciar novos agentes. Em um contexto modelado a partir de um estabelecimento comercial, cada agente cliente tinha seu agente vendedor associado, agrupado em diferentes *hosts*. Cada vendedor foi projetado para responder às solicitações de seu cliente associado. A relação entre o número de agentes e o tempo de processamento de solicitações mais uma vez apresentou uma proximidade linear. Com os experimentos realizados nos cenários acima, os autores esperavam encontrar um limite relativo a quantidade de agentes que o JADE poderia administrar em diferentes circunstâncias. Esse limite, é claro, também está relacionado ao *hardware* em uso, bem como o desempenho relacionado às interações entre a Java Virtual Machine e o JADE. O que foi comprovado é que o JADE não introduz uma sobrecarga substancial no sistema, e todas as limitações observadas estão vinculadas à máquina virtual Java.

Recentemente, Radhakrishnan et al. [RCS18] apresentou um relatório de desempenho comparativo entre SPADE e JADE, com foco na segurança no ambiente de nuvem. Sua razão para escolher as plataformas foi semelhante à deste trabalho, com a adição de considerar a possibilidade de transferência de conhecimento com o uso de ontologias. Eles usaram uma versão anterior do SPADE que a usada neste trabalho, que é compatível com Python 3 e superior. Portanto, como o Python 3 impacta diretamente no desempenho do SPADE, não podemos comparar seus resultados com os obtidos aqui.

No entanto, embora eles tenham mostrado que o JADE tem um uso de CPU (Central Process Unit) comparativamente menor do que o SPADE quando o número de mensagens aumenta, é preciso observar que os experimentos não informam a quantidade de agentes utilizada, e todos eles estão situados no mesmo *host*. Nesse cenário, JADE e SPADE apresentaram desempenho semelhante. A abordagem adotada nesse trabalho torna-se válida uma vez que apresentamos algumas limitações e especificidades de cada plataforma quanto ao desempenho da camada de transporte a depender da localidade do agente e a configuração dos elementos envolvidos.

3.3 Escalabilidade e desempenho em MAS

Em [WM15] é apresentada uma visão geral dos desafios para criar sistemas em larga escala, considerando especificamente o ambiente de MAS. A definição de grande escala aqui é focada em sistemas que podem ser decompostos em subsistemas, por razões práticas (design) e/ou desempenho. Existem quatro preocupações sobre sistemas de grande escala em MAS neste artigo: o primeiro é sobre a organização. O MAS tem uma natureza descentralizada e, por isso, uma organização deve ser cuidadosamente projetada para não criar um ambiente “caótico”. A segunda preocupação é sobre o gerenciamento de recursos no sistema. Às vezes, os agentes precisam usar recursos compartilhados

e, para isso, é necessário criar uma boa política para que o acesso aos recursos não seja um gargalo. O terceiro aspecto diz respeito à comunicação. Quando milhares de agentes são necessários, o ambiente deve fornecer meios para a comunicação descentralizada. O último é sobre interações e o papel do ambiente nisso. Para os autores, o ambiente deve oferecer primitivas de alto nível para a percepção dos agentes. Como conclusão, os autores propuseram que algumas responsabilidades deveriam ser endereçadas ao ambiente para reduzir a responsabilidade do agente. Um exemplo é o pré e pós processamento de dados para as necessidades dos agentes. Outra questão apontada é o *design* de MAS subjacentes a arquiteturas físicas de computação de alto desempenho como GPUs (Graphics Processing Unit).

Os autores em [ADF16] estão motivados a usar a plataforma Jason para controle de qualidade de processos de desenvolvimento de *software* em um sistema de larga escala. No cenário apresentado, os agentes realizam algumas modificações no código como refatoração e correções. Apesar das características adequadas de Jason para este cenário, de acordo com os autores, são identificadas dificuldades para ampliar essa plataforma usando centenas de milhares de agentes ou mais. A ideia de usar o MapReduce é basicamente identificar partes do ambiente simulado que são completamente independentes e, portanto, podem ser processadas em paralelo. Sendo assim, foi identificado que a deliberação em um agente Jason pode ser executada de forma independente, quando este não se comunica ou realiza uma ação no ambiente. O trabalho apresenta a comparação utilizando diferentes intérpretes do Python e outras duas abordagens (Jason 2.0a e MASERaTi), avaliando a execução em relação ao número de agentes. A execução usando o interpretador PyPy 2.4.0 apresentou melhor desempenho, enquanto o Jason 2.0a não pode ser escalável após 10 000 agentes. No entanto, o trabalho não apresenta informações sobre os recursos utilizados nos experimentos para estimar o quão bom os resultados são considerando os recursos onde os experimentos foram conduzidos.

Os autores em [CDIPDPC15] propõe uma nova infraestrutura capaz de integrar diferentes tecnologia de sensoriamento, coleta de dados de diferentes fontes e gerenciamento inteligente de informação baseada em MAS para Internet das Coisas (IoT). Essa necessidade surge dada a grande quantidade de ferramentas de propósito específico que desempenham tais atividades. Para tanto, os autores escolheram a arquitetura que combina agentes reativos e proativos, neste trabalho chamados de *lightweight* and *heavyweight* agents. Na plataforma proposta, os autores focam numa característica importante para MAS que são sistemas abertos. Em sistemas abertos, basicamente agentes podem entrar e sair do sistema por diferentes razões, criando ambientes dinâmicos e heterogêneos. Com intuito de oferecer um crescimento do sistema de forma escalonável e organizada, a plataforma adota *Cloud Computing* para oferecer recursos e uma representação virtual da organização dos agentes, assim como o MOISE+ se propõe na arquitetura JaCaMo [HSB07]. A plataforma é organizada através de camadas, onde a camada de mais alto nível encapsula serviços das camadas inferiores na forma de *Web Services*, acessíveis aos usuários. Como forma de avaliação, os autores apresentam um cenário simulado de tráfego de veículos e esperam avaliar a plataforma em um cenário real no futuro. Apesar de não ser detalhada a arquitetura de funcionamento dos agentes, a plataforma apresenta uma solução interessante uma vez que combina agentes reativos e proativos em um único sistema. Em termos de escalabilidade, um sistema aberto baseado em *Cloud Computing*, usando uma organização virtual parece uma boa escolha.

Em [PCGLO14] é abordada a utilização de Jason em computadores com multi-processadores e suas capacidades utilizando tais arquiteturas. Uma das motivações que levaram esse estudo ser feito é que trabalhos que abordam a capacidade das ferramentas multiagentes não exploram esse contexto, focando basicamente na camada de transporte ou nos serviços oferecidos pelas plataformas. Jason pode ser executada de duas maneiras: *Centralizado* e distribuído usando JADE. De forma centralizada, toda a arquitetura é executada em um mesmo *host*, enquanto que usando JADE, Jason faz uso da tecnologia implantada nessa arquitetura para distribuir componentes. O estudo foca na execução centralizada do Jason.

Na execução Centralizada, existem duas configurações de *threads*: PThe, onde um número fixo de *threads* é compartilhado entre os agentes (cada agente possui seu próprio *thread*) oferecendo simultaneidade na execução mas limitando o número de agentes e Ptha, onde um *pool de threads* é

compartilhado pelos agentes, ao custo de reduzir a simultaneidade. A comunicação na arquitetura Centralizada é feita através de passagem de eventos, isso significa que são clonadas na JVM, e não são serializadas através de uma rede. O trabalho faz dois tipos de testes na plataforma Jason, um de computação onde é medido o uso de CPU, e outro teste que analisa o desempenho na comunicação entre os agentes. Jason possui modos de execução sincronizados e assíncronos que também foram avaliados. Como resultado, o trabalho apresenta vários cenários onde é definida uma quantidade de agentes onde ocorrem saturações, baseados no *hardware* onde os experimentos foram executados e no modo de execução (síncrono ou assíncrono).

Os autores em [LSY09] propõem um esquema de tomada de decisão para replicar dinamicamente agentes e/ou dados. O esquema de monitoramento coleta informações sobre os agentes no sistema, como quantidade de carga, sua função e duração da tarefa. Em seguida, o esquema de decisão de replicação, com base nessas informações, tenta minimizar a quantidade de replicação necessária, reduzindo a sobrecarga do mecanismo de tolerância à falhas. O esquema foi implementado em um serviço de saúde que fornece MAS composto por 64 agentes distribuídos em cinco *hosts* e comparado com um esquema em que os agentes replicados são selecionados aleatoriamente, medindo o tempo de recuperação à medida que o número de agentes com falha aumenta. O tempo de recuperação com o esquema proposto é menor do que com o aleatório, onde a diferença aumenta com o número de falhas. Verificou-se também que a sobrecarga da CPU do esquema proposto é limitado a aproximadamente 3%.

Em [TJ00] os autores apostam que um MAS deve ter características que possam prover o *self-building*, uma vez que são ambientes dinâmicos e abertos (considerando a Internet e corporate intranets). Para tanto, o foco deste trabalho é como um MAS pode se adaptar para vários tamanhos de população de agentes considerando dificuldades inerentes. Os autores acreditam que um MAS é mais escalável se pode lidar com diferentes tamanhos de populações de agentes e de forma dinâmica. A métrica usada aqui para mensurar escalabilidade é a capacidade dos agentes em atingir suas metas coletivamente. Para tal, é proposto um cenário de avaliação baseado em comércio entre agentes. Há dois tipos de agentes nesse cenário: Clientes e Fornecedores. Clientes são capazes de estabelecer preferências sobre produtos, fazer recomendações e fazer compras coletivas com intuito de obter descontos. Fornecedores por sua vez, possuem um comportamento menos dinâmico, estabelecendo preço para as mercadorias. As ações aqui são baseadas em *ticks*, ou seja, quando todos os agentes atingem determinado estágio, têm-se uma próxima etapa do processo. Os *ticks* representam uma passagem de tempo, mas não limitado a uma certa quantidade dele. Assim, a avaliação não sofre influência de comportamentos individuais, como por exemplo, um agente receber informações antes que outro, obtendo assim vantagens.

Para a avaliação, os autores propuseram várias formas de organizações para os agentes. Essas formas influenciam em como os agentes interagem entre si, trocando informações, formando grupos e tomando ações. No primeiro cenário, agentes com o mesmo papel não interagem entre si. No segundo cenário, essa restrição de comunicação não existe e o terceiro é similar ao segundo, no entanto existe um novo agente que intermedia tarefas coletivas.

O trabalho de uma forma geral apresenta questões importantes sobre escalabilidade, mas fortemente ligado a um cenário específico e suas possíveis organizações. Os autores reconhecem isso e consideram diferentes cenários onde se exige um grande número de re-organizações, uma vez que o cenário apresentado é bem estável nesse sentido. Em relação ao presente trabalho, a avaliação desse tipo de abordagem sobre escalabilidade é inviável uma vez que se pretende escolher ferramentas para uma solução, e não avaliar sua implantação (que seria uma etapa posterior).

3.4 Exemplos de MAS de larga escala

Fernandez et al. [FGLO10] definem um MAS de larga escala, no contexto de simulação de multidões, implementado usando Jason. A análise de desempenho nesse trabalho busca responder se especificamente a arquitetura Jason é capaz de gerenciar uma quantidade massiva de ações concorrentes. Para tal, a avaliação é baseada no tempo de resposta da ação dos agente e a porcentagem de

utilização da CPU pelo MAS ao simular um *benchmark* de locomoção. O tempo de resposta nesse caso é definido entre a requisição de um movimento aleatório pelo agente ao ambiente e a resposta do ambiente ao agente. Foram propostos três cenários baseados nas infraestruturas capazes de executar o MAS: Centralizada, onde todo o processamento é realizado em um único *host*; SACI uma API que fornece mecanismos para comunicação e localização de agentes de forma distribuída e transparente; e a plataforma JADE, que oferece aderência aos requisitos do protocolo FIPA. Um aspecto importante levantado por esse trabalho é a limitação do Jason pelo número de *threads* da JVM. Cada agente no Jason pode corresponder respectivamente a um *thread* processando seu ciclo de raciocínio, e nesse trabalho, dada as especificações do *hardware* onde foram realizados os experimentos e a capacidade da versão da JVM utilizada, o limite máximo foi definido como 3500 *threads*. Outra possibilidade é o compartilhamento desses *threads*, levando o MAS a uma concorrência por recursos mas não restringindo o limite de agentes pelo número de threads possíveis. Em suma, o trabalho conclui que Jason é uma plataforma adequada para simulações de multidões, e aponta ajustes necessários principalmente relacionadas a Coleta de Lixo do Java e a distribuição dos agentes.

Em [BCG15b] é apresentada uma aplicação da plataforma AMUSE, uma extensão da plataforma JADE e WADE. AMUSE foi criada para apoiar o desenvolvimento de aplicações de jogos em larga escala, provendo ao desenvolvedor algumas ferramentas para tal e permitindo o foco na construção de jogos sociais *mobile*. WADE foi desenvolvida para lidar com problemas inerentes de MASs relacionados a administração e tolerância a falhas, e é responsável por gerenciar redes e serviços de milhões de usuários na Telecom Italia [BCG15a]. A solução adotada constitui basicamente de dois grupos de agentes: *front-end*, responsáveis por gerenciar a interação com o usuário, e agentes *back-end*, operando no lado do servidor. Esses agentes são designados a oferecer suporte a recursos compartilhados pelos usuários, controle de acesso a seções de jogos, coleta e armazenamento de dados dos jogadores, entre outros. Os autores acreditam que uma plataforma baseada em agentes pode oferecer benefícios na coordenação dinâmica de um grande número de grupos de usuários e reduzir o tempo de disponibilizar jogos ao mercado devido a abstração providas para lidar com sistemas distribuídos em larga escala.

Em [LW05] é apresentada a AmbieAgents, uma infraestrutura baseada em agentes para entrega de informações baseadas em contexto para usuários móveis. Essa estrutura é baseada em JADE/-LEAP e implementada, testada e demonstrada em um cenário realístico no aeroporto internacional de Oslo. O principal objetivo do projeto é fornecer ao usuário móvel informações do ambiente e de forma personalizada. Para tanto, o modelo adotado é composto por cinco partes: contexto social; contexto de tarefa; contexto pessoal; contexto de ambiente e contexto de espaço-tempo. Essas partes servem como base para a especificação da ontologia utilizada pelo AmbieAgents para comunicação e basicamente expressam preferências e informações dos usuários. Um agente central desse sistema é o Agente de Contexto. Esse agente possui ampla autonomia e toda a comunicação é canalizada através dele, pois ele administra o espaço de contexto do usuário, promovendo comunicação com os demais agentes sempre que as informações do contexto do usuário sofrem alterações. Cada usuário possui um Agente de Contexto dedicado, já que as informações são anônimas. Como forma de validar a pesquisa, foi disponibilizada a infraestrutura AmbieAgents para viajantes e colhida informações durante e depois o seu uso. O objetivo foi identificar tendências e receber comentários. Como resultado, foi identificado a satisfação dos usuários em receber informações baseadas em contexto.

Open Smart Classroom [SMM⁺09] é baseado em uma plataforma inteligente que consiste em um MAS que fornece interação e serviços entre humanos e computadores. Este trabalho concentra-se muito na interoperabilidade entre diferentes *Smart Spaces*, descrevendo sua comunicação (com base no mecanismo de publicação/assinatura) e as principais funcionalidades do agente. A interoperabilidade também concede escalabilidade de sistemas, através dos quais muitos *Smart Spaces* podem ser interconectados criando uma *Open Smart Community*. O desempenho do sistema foi avaliado estabelecendo duas salas de aula em diferentes países e analisando a escalabilidade da carga. O gargalo evidenciado estava em um aplicativo de terceiros específico para o contexto de *e-learning*, responsável pela tradução quase em tempo real.

Para facilitar a identificação da relação dos trabalhos citados neste capítulo e o nosso, resumimos os trabalhos na tabela 3.1.

Tabela 3.1: *Síntese dos Trabalhos Relacionados*

Benchmarks e critérios para análise de escalabilidade e desempenho em MAS	
[SAM ⁺ 07]	<i>Benchmarks</i> para avaliação de plataformas
[BvSW ⁺ 01]	Técnicas para lidar com escalabilidade
Desempenho e transporte de mensagens em MAS	
[Det01]	<i>Middleware</i> para acesso a recursos distribuídos
	Técnicas de programação para melhorar a escalabilidade
[CQV02]	Comparação da camada de transporte de JADE com Java RMI
[MSA06]	Análise de <i>avgRTT</i> e DF de plataformas aderentes a FIPA
[Vrb03]	Comparação de <i>avgRTT</i> e uso de memória das plataformas JADE, JACK, FIPA-OS e ZEUS
[CTG ⁺ 04]	Avaliação da escalabilidade de JADE em cenários de aplicação
[RCS18]	Comparação de <i>avgRTT</i> entre JADE e SPADE
Escalabilidade e desempenho em MAS	
[WM15]	Questões de escalabilidade envolvendo organização, comunicação, gerenciamento de recursos e interações
[ADF16]	Uso da plataforma Jason para um cenário de larga escala
[CDIPDPC15]	Infraestrutura para IoT que utiliza agentes reativos e proativos
[PCGLO14]	Utilização de Jason em computadores com multi-processadores
[LSY09]	Aborda tomada de decisão para replicar dinamicamente agentes e/ou dados
[TJ00]	Adaptabilidade na organização de um MAS para cenários de larga escala
Exemplos de MAS de larga escala	
[FGLO10]	Uso da plataforma Jason em simulação de multidões
[BCG15b]	Aplicação da plataforma AMUSE em jogos de larga escala
[LW05]	Uso da AmbieAgents no aeroporto internacional de Oslo
[SMM ⁺ 09]	Apresenta o Open Smart Classroom, um MAS que fornece interação e serviços entre humanos e computadores

Capítulo 4

Desempenho e Escalabilidade da Camada de Transporte de Mensagens

As experiências descritas neste capítulo são baseadas nos *benchmarks* estabelecidos em [SAM⁺07]. Os quatro *benchmarks* são independentes de plataforma e servem para orientar a avaliação de desempenho e escalabilidade de um MAS:

- *Benchmark 1 - Número de Hosts*: Neste *benchmark*, agentes são distribuídos em uma quantidade de *hosts* estabelecida e trocam mensagens entre si. Os papéis dos agentes são divididos entre remetentes e destinatários. Cada remetente i envia uma mensagem para um destinatário j , e ao receber resposta o remetente envia uma mensagem para o destinatário $j + 1$, e assim sucessivamente, até o limite de mensagens estabelecido.
- *Benchmark 2 - Recepção Massiva (1 destinatário)*: O objetivo desse *benchmark* é sobrecarregar um único agente destinatário. Para tanto, vários agentes enviam mensagens para o agente destinatário que por sua vez tem a função de responder cada uma, semelhante a um *ping*. Agentes destinatário e remetentes estão em *hosts* diferentes, com o número de *hosts* que hospeda agentes remetentes a ser definido.
- *Benchmark 3 - Recepção Massiva (N destinatários)*: Este *benchmark* é complementar ao Benchmark 2, uma vez que seu objetivo é avaliar se o gargalo do sistema está na camada de transporte de mensagens ou no gerenciamento da fila de mensagens do próprio agente. Para tanto, o *benchmark* consiste em alocar vários agentes destinatários em um único *host*, que recebem mensagens de agentes remetentes distribuídos em vários *hosts*.
- *Benchmark 4 - Número de Agentes por Host*: Esse *benchmark* consiste em pares de agentes destinatários e remetentes que trocam mensagens exclusivamente entre si. O objetivo é verificar a escalabilidade do sistema aumentando a quantidade de pares de agentes.

4.1 Descrição dos experimentos

Algumas plataformas foram consideradas inicialmente para o estudo, no entanto observamos que muitas já não eram utilizadas ativamente pela comunidade ou não possuíam capacidade de serem distribuídas. Sendo assim, incluímos no critério de seleção, plataformas utilizadas recentemente na competição *Multiagent Programming Context*¹ e aquelas com características apropriadas citadas em [KB15]. Outro critério estabelecido é que as plataformas deveriam ser *open source*. As platafor-

¹<https://multiagentcontest.org>

mas inicialmente selecionadas foram: SPADE², eJason³, ASTRA⁴, JaCaMo⁵, JIAC V⁶, JADE⁷ e AgentScape⁸.

Como critério intrinsecamente relacionado a escalabilidade, a plataforma deveria oferecer um mecanismo próprio para suporte a distribuição e comunicação entre *hosts*. Por essa razão, JaCaMo foi excluída uma vez que pode ser usada sobre a arquitetura do JADE para comunicação distribuída, havendo uma intersecção de tecnologias. Já eJason foi descontinuada, bem como AgentScape que recebeu sua última atualização em Outubro de 2014.

Tabela 4.1: *Plataformas MAS avaliadas*

Plataforma	Distribuída	Ativa	Padrão de Comunicação	Linguagem
SPADE	✓	✓	FIPA-ACL	Python
eJason	✓		KQML	AgentSpeak(L)
ASTRA	✓	✓	FIPA-ACL	AgentSpeak(L)
JIAC V	✓	✓	Não conhecido	Java
JaCaMo		✓	KQML	AgentSpeak(L)
JADE	✓	✓	FIPA-ACL	Java
AgentScape	✓		Não conhecido	Java

Para avaliar o desempenho e a escalabilidade durante a comunicação em plataformas multi-agente, foram criados cenários com agentes comunicativos (pares destinatário-remetente). O objetivo é congestionar o canal de comunicação, aumentando o número de agentes, a sobrecarga na comunicação ou a sua distribuição. Além disso, são exploradas diferentes configurações das plataformas⁹.

JADE usa mecanismos diferentes para enviar mensagens dependendo da localização dos agentes. Se ambos, o *Destinatário* e *Remetente* estiverem no mesmo *host*, a mensagem será clonada e a referência do novo objeto será passada. Caso contrário, se ambos estiverem em *hosts* distintos, a mensagem precisará ser serializada e enviada por meio do Java RMI.

SPADE, por sua vez, usa um cliente do protocolo XMPP para enviar e receber mensagens, que são baseadas em XML. As mensagens enviadas por agentes no SPADE são intermediadas por um servidor Jabber. Qualquer agente que deseje se comunicar precisa estar registrado em um servidor Jabber, podendo dois servidores encaminhar mensagens entre si. A configuração do servidor Jabber tem um enorme impacto no desempenho e na confiança do serviço oferecido (isso será discutido mais adiante).

A comunicação implementada em ASTRA e JIAC V é baseada em *Middlewares* Orientados à Mensagens (MOMs). MOMs como ActiveMQ¹⁰ usado em JIAC V, são desenhados para enviar mensagens entre duas aplicações ou dois componentes em uma mesma aplicação de forma transparente e assíncrona.

Devido à diferença dos mecanismos de transporte de mensagens para intra e interplataforma, foram definidos dois cenários para avaliar o desempenho do transporte de mensagens para o *benchmark 1* "Número de agentes por *host*".

Cenário 1 Todos os agentes comunicativos estão no mesmo *host*.

Cenário 2 Os agentes *Destinatário* e *Remetente* estão em *hosts* diferentes.

²<https://pypi.org/project/spade/>

³<https://github.com/avalor/eJason>

⁴<http://astralanguage.com/wordpress/>

⁵<http://jacamo.sourceforge.net>

⁶<http://www.jiac.de/agent-frameworks/jiac-v/>

⁷<https://jade.tilab.com>

⁸<http://www.agentscape.org/>

⁹O código base dos experimentos realizados está disponível em: <https://gitlab.com/hdonancio/benchmarksmas>

¹⁰<https://activemq.apache.org>

Em ambos os cenários, foi estabelecido que a resolução de nomes (DNS) para os agentes SPADE era definida em código, porque a busca por agentes em plataformas baseadas em Java é um serviço transparente fornecido pela própria plataforma. Para o Cenário 2, o servidor Jabber está no *host* dos agentes *Remetentes*.

Foi adotado o *avgRTT* como métrica para avaliar o desempenho do transporte de mensagens. O RTT (round trip time), é o período de tempo entre uma mensagem ser enviada por um agente *remetente* para um agente *destinatário*, mais o tempo que leva para o agente *remetente* confirmar que a mensagem foi recebida pelo outro agente. A Figura 4.1 ilustra o RTT para agentes SPADE no Cenário 1 e a Figura 4.2 no Cenário 2.

Figura 4.1: Comunicação no cenário intraplataforma com o servidor Jabber

Figura 4.2: Comunicação no cenário interplataforma com o servidor Jabber

Um par de agentes comunicativos é composto por um *Remetente* e um *Destinatário* que se comunicam um com o outro. O ciclo de comunicação entre um remetente e um destinatário é mostrado abaixo:

Agente Remetente

CyclicBehaviour:

Cria mensagem

Obtém o tempo inicial

Para cada mensagem a enviar:

Envia mensagem ao Destinatário

Aguarda resposta

Obtém o tempo final

Agente Destinatário

CyclicBehaviour:

Aguarda mensagem do Remetente

Envia resposta

4.1.1 Hardware e software usados nos experimentos

Para os experimentos, foram usadas duas configurações diferentes de conjuntos de *host*, chamados de *Setup 1* e *Setup 2*. As configurações de *hardware* e *software* descritas na Tabela 4.2 são para o *Setup 1* e as Tabelas 4.3, 4.4, 4.5 para o *Setup 2*. No cenário interplataforma, os computadores estão conectados por uma LAN Ethernet de 1000 Mbit/s própria, para que não haja interferência externa. Em todos os *hosts*, foram usadas as versões das plataformas: JADE 4.5, SPADE 3.0.9, ASTRA 3.6.2 e JIAC V 5.2.4.

Tabela 4.2: *Hardware e software usados nos experimentos (Setup 1).*

Modelo	Centrium Eliteline 7700
Processador	Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
Memória total	8 Gb
Sistema Operacional	Lubuntu 18.04/18.10
Java	Sun JDK 1.8_102
Python	3.6.7
Servidor Jabber	Prosody IM 0.10.2-1

Tabela 4.3: *Hardware e software usados nos experimentos (Setup 2 - Host 1).*

Modelo	Acer Nitro 5 AN515-51
Processador	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
Memória total	8 Gb
Sistema Operacional	elementary OS 5.0 Juno
Java	OpenJDK 10.0.2
Python	3.6.7
Servidor Jabber	Prosody IM 0.11.2

Tabela 4.4: *Hardware e software usados nos experimentos (Setup 2 - Host 2).*

Modelo	Asus K40IJ
Processador	Intel(R) Core(TM) 2 Duo CPU T6600 @ 2.20GHz
Memória total	4 Gb
Sistema Operacional	Ubuntu 18.04.1 LTS
Java	OpenJDK 11.0.3
Python	3.6.8

4.1.2 Questões de medição, configuração e execução

Como adotado em [CQV02], foi utilizado o método `Java long System.currentTimeMillis()` para obter o tempo corrente nos testes usando as plataformas baseadas em Java. Para testes com

Tabela 4.5: *Hardware e software usados nos experimentos (Setup 2 - Host 3).*

Modelo	LENOVO(TM) ideapad(TM) 110
Processador	Intel(R) Celeron(R) CPU N3060 @ 1.60GHz
Memória total	4 Gb
Sistema Operacional	Linux Mint 19.1 Tessa
Java	OpenJDK 11.0.3
Python	3.6.8

SPADE, foi usado `int(round(time_.time() * 1000))` onde `time_` é o módulo `time`¹¹. Esses métodos têm precisão suficiente para os experimentos.

Os clientes e servidores do Jabber usam uma resolução de Sistema de Nomes de Domínios (DNS) para obter o endereço dos serviços. Nos cenários propostos, é testado apenas as plataformas em uma rede local exclusiva (LAN). Isso é muito importante para manter as plataformas comparáveis, porque procurar um serviço usando um DNS na *Web* aumenta significativamente o RTT. Em testes realizados com um par de agentes registrados no servidor Jabber `xmpp.jp`¹², o *avgRTT* foi de 707.645 ms. Para isso, foi usado o `Dnsmasq`¹³ para apontar o serviço no sistema.

Outra questão importante é sobre o paralelismo para executar muitos arquivos Python simultaneamente. Para isso, foi utilizado o pacote *multiprocessing*¹⁴. Isso melhora um pouco a quantidade de agentes em execução em um *host* e adere paralelismo na execução. Para os cenários usando JADE, foi usada a configuração padrão, ou seja, não foi feita nenhuma alteração no número de *threads* nem aumentados os recursos para a JVM. Já na plataforma ASTRA, a quantidade de *threads* (`setSchedulePoolSize()`) é exatamente o dobro da quantidade de núcleos da máquina hospedeira no *benchmark* 4. Essa quantidade foi definida devido a uma sugestão dos desenvolvedores. Já nos demais *benchmarks* é usada a configuração padrão com 2 *threads*. ASTRA e JIAC V foram testadas na mesma configuração da JVM adotada para JADE.

É usado o servidor Prosody IM¹⁵ como servidor Jabber. A configuração deste servidor é crucial para o desempenho da camada de transporte de mensagens, uma vez que alguns dos módulos podem ser ativados ou desativados. Com base na configuração do arquivo de exemplo de Prosody¹⁶, os módulos são ativados ou desativados de acordo com o cenário de avaliação (com e sem alguns módulos). A tabela 4.6 descreve os módulos que podem estar ativos ou não. Apenas os módulos `roster`, `saslauth`, `tls`, `dialback`, `disco` são de seleção obrigatória. É chamado de SPADE(L), quando somente os módulos "(geralmente) **obrigatório**" e `register` estão habilitados. Quando todos os módulos estão habilitados ("**obrigatório**", "**recomendado**" e "**interessante**"), é chamado simplesmente de SPADE.

Por fim, a plataforma JIAC V apresentou muitas particularidades tanto no modo de execução quanto no seu funcionamento. A primeira questão é sobre o paralelismo na execução. Em todas as plataformas avaliadas, era de extrema importância que todas as instâncias de agentes iniciassem em conjunto, a fim de garantir que a sobrecarga no sistema em termos de quantidade de agentes fosse equivalente. JIAC V, como já dito, é organizada em nodos e até o presente momento deste trabalho, não havia na plataforma alguma funcionalidade que garantisse a inicialização de todos os agentes em um nodo após uma determinada condição. Essa condição foi atingida em ASTRA através do uso de artefatos `CARTAgO` e suas *propriedades observáveis*. ASTRA assim como JIAC V, cria seus agentes impreterivelmente em tempo de execução. Quando um nodo é inicializado em JIAC, os agentes vão sendo criados um a um e a partir daí ocorre a sincronização entre os nodos ao mesmo tempo que agentes já iniciados entram no método `execute()`. Segundo envolvidos no

¹¹<https://docs.python.org/3/library/time.html>

¹²Servidor disponível em: <https://www.xmpp.jp>

¹³<http://www.thekelleys.org.uk/dnsmasq/doc.html>

¹⁴<https://docs.python.org/2/library/multiprocessing.html>

¹⁵Este servidor é recomendado pelo projeto SPADE

¹⁶Prosody IM: <https://prosody.im/>

Tabela 4.6: Módulos do servidor Prosody IM Jabber/XMPP [IM19]

roster	Permitir que os usuários mantenham e gerenciem listas de contatos	Obrigatório
saslauth	Autenticação para clientes e servidores	Obrigatório
tls	Adiciona suporte para TLS em conexões c2s/s2s	Obrigatório
dialback	Suporte de discagem s2s	Obrigatório
disco	Descoberta de serviços	Obrigatório
carbons	Mantenha vários clientes em sincronia	Recomendado
pep	Permite que os usuários publiquem coisas pessoais	Recomendado
private	Armazenamento em XML privado	Recomendado
blocklist	Permite que os usuários bloqueiem a comunicações com outros usuários	Recomendado
vcard	Permitir que os usuários definam vCards	Recomendado
version	Responde às solicitações de versão do servidor	Interessante
uptime	Informa há quanto tempo o servidor está executando	Interessante
time	Deixe que os outros saibam a hora local do servidor	Interessante
ping	Respostas para pings XMPP com pongs	Interessante
register	Permitir que os usuários se registrem no servidor usando um cliente	Interessante
mam	Armazena mensagens e permite que os usuários acessem	Interessante
smacks	Gerenciador de <i>stream</i>	Interessante

projeto, é muito cedo para um agente buscar outro agente (destinatário) no método `doStart()`, além de não haver método que garanta que essa sincronia já foi realizada nessa etapa. Um fato que possivelmente ameniza essa questão é que todos agentes possuem um intervalo de execução predeterminado. Nos experimentos realizados, esse intervalo de execução foi fixado em 50 ms.

4.2 Resultados dos Experimentos

4.2.1 Benchmark 1

O *benchmark 1* (veja na Figura 4.3) consiste em alocar em cada *host*, um par de agentes *remetente* e *destinatário*. Cada agente remetente i envia uma mensagem para um agente destinatário j , e ao receber a resposta, o remetente i envia para o destinatário $j + 1$, e assim sucessivamente. Para esse experimento, foi utilizado o *Setup 2* apresentado em 4.1.1. Cada agente remetente envia 1000 mensagens para cada agente destinatário, totalizando 3000 mensagens enviadas por cada *host*. Os resultados mostrados na Tabela 4.11 são relativos ao desempenho individual de cada *host* e a média dos *avgRTT* de todos os *hosts*.

Figura 4.3: Benchmark 1: Cada *host* possui um par de agentes Destinatário de Remetente. Um agente Remetente envia 1000 mensagens para cada agente Destinatário

De maneira geral, o *avgRTT* cresce à medida que os recursos computacionais em termos de capacidade diminui. Isso é evidenciado dado que o *Host 1* possui maior capacidade em termos de recursos e consequentemente obteve menor *avgRTT*, enquanto o *Host 3* apresentou maior *avgRTT*

Tabela 4.7: *avgRTTs em LAN para o Benchmark 1 (em milisegundos).*

Plataforma	<i>avgRTT</i> (ms)			
	<i>Host 1</i>	<i>Host 2</i>	<i>Host 3</i>	Média
ASTRA	1.638	2.555	3.142	2.445
SPADE	1.085	3.624	7.419	4.043
JADE	0.156	0.210	0.472	0.280
JIAC V	8.769	9.045	10.078	9.298

uma vez que possui menor capacidade, considerando todas as plataformas avaliadas. JADE foi a plataforma que apresentou menor *avgRTT*, enquanto JIAC V o maior, como mostra a Figura 4.4. Além disso, SPADE apresentou a maior variação em termos de *avgRTT* entre os *hosts*.

Figura 4.4: *avgRTTs em LAN para o Benchmark 1 (em milisegundos)*

Os resultados desse experimento são ilustrativos para um cenário com 3 *hosts*, no entanto, reconhecemos que uma análise completa consiste em aumentar a quantidade de *hosts* e então analisar o comportamento da plataforma multiagente em gerenciá-los à medida em que estes *hosts* vão sendo adicionados.

4.2.2 Benchmark 2

O *benchmark 2* (veja na Figura 4.5) consiste em definir 1 agente *destinatário* em um *host* e N *remetentes* em *host(s)* diferente(s) do *destinatário*. O objetivo é avaliar a capacidade de resposta da plataforma quando um agente é sobrecarregado. Nos resultados apresentados na Figura 4.6 e nas Tabelas 4.8 e 4.9, foram criados cenários com 1, 5 e 10 agentes remetentes nos *hosts* do *Setup 2*, sendo o *Host 1*, que possui maior capacidade em termos de recursos computacionais, o *host* hospedeiro do agente destinatário. Cada agente remetente envia 1000 mensagens de 11 caracteres para o agente destinatário.

SPADE e JIAC V apresentaram desempenhos bem inferiores em relação a ASTRA e JADE. JIAC V, por exemplo, no cenário com 10 agentes remetentes, obteve um *avgRTT* de 12.72 ms no *Host 3*. SPADE por sua vez, parece ser mais suscetível a perda de desempenho à medida em que o sistema cresce em número de agentes e a quantidade de recursos é inferior. Considerando o *Host*

Figura 4.5: *Benchmark 2: N agentes Remetentes enviam cada um 1000 mensagens para o agente Destinatário. Remetentes estão distribuídos em dois hosts diferentes do host do Destinatário*

Tabela 4.8: *avgRTTs em LAN para o Benchmark 2 (em milisegundos).*

Plataforma	avgRTT (ms)		
	Host 2 (1 remetente)	Host 2 (5 remetentes)	Host 2 (10 remetentes)
ASTRA	2.641	2.904	3.8025
SPADE	4.716	11.995	18.959
JADE	0.339	0.247	0.186
JIAC V	9.516	21.059	35.607

Tabela 4.9: *avgRTTs em LAN para o Benchmark 2 (em milisegundos).*

Plataforma	avgRTT (ms)		
	Host 3 (1 remetente)	Host 3 (5 remetentes)	Host 3 (10 remetentes)
ASTRA	4.700	7.164	9.259
SPADE	8.363	39.678	71.389
JADE	0.597	0.490	0.367
JIAC V	12.718	23.646	39.790

3 hospedando 1 agente, SPADE obteve um *avgRTT* de 8.36 ms enquanto que no cenário com 10 agentes, o *avgRTT* resultante foi de 71.39 ms, resultando em um crescimento de aproximadamente 754%.

Finalmente, JADE e ASTRA apresentaram pouca variação no desempenho, com JADE novamente obtendo o menor *avgRTT*.

4.2.3 Benchmark 3

No *benchmark 3* (veja na Figura 4.8), N agentes remetentes enviam mensagens para M agentes destinatários que estão hospedados em um único *host*. Para tanto, alocamos 3 agentes destinatários no *Host 1* do *Setup 2* e 5 agentes nos *hosts 2 e 3* e calculamos o *avgRTT* de todos os agentes remetentes. Com uma pequena modificação do *benchmark* originalmente proposto, acrescentamos o fato de que todos os remetentes enviam mensagens para todos os destinatários, aumentando assim o número de mensagens enviadas em relação ao *Benchmark 2*, uma vez que cada agente destinatário recebe 1000 mensagens de cada remetente. O objetivo é avaliarmos se a distribuição do tráfego de mensagens entre vários receptores (que são colocados no mesmo *host*) fornece uma melhor resposta.

De maneira geral, mesmo com o aumento na quantidade de mensagens, todas as plataformas responderam positivamente ao aumento na quantidade de agentes destinatários. JIAC V, foi a

Figura 4.6: *avgRTT* para o Host 2 em LAN

Figura 4.7: *avgRTT* para o Host 3 em LAN

Tabela 4.10: *avgRTTs* em LAN para o Benchmark 3 (em milisegundos).

Plataforma	<i>avgRTT</i> (ms)		
	<i>Host 2</i> (1 destinatário)	<i>Host 2</i> (3 destinatários)	Variação de desempenho 1-3
ASTRA	2.904	2.412	-16.9%
SPADE	11.995	8.433	-29.7%
JADE	0.247	0.136	-44.9%
JIAC V	21.059	8.252	-60.8%

Figura 4.8: Benchmark 3: N agentes Remetentes enviam cada um 1000 mensagens para cada um dos três agentes Destinatários. Remetentes estão distribuídos em dois hosts diferentes do host dos Destinatários

Figura 4.9: $avgRTT$ no cenário intraplataforma

plataforma que melhor respondeu a esse aumento, enquanto ASTRA apresentou a menor variação nos dois *hosts* analisados (ver Figuras 4.9 e 4.10). SPADE, por sua vez, assim como nos resultados de outros *benchmarks* já apresentados, tem seu $avgRTT$ bem superior as demais quando executada em um *host* com menor capacidade computacional, enquanto JADE apresenta mais uma vez o menor $avgRTT$, em ambos os *hosts*.

Figura 4.10: *avgRTT no cenário interplataforma*

Tabela 4.11: *avgRTTs em LAN para o Benchmark 3 (em milissegundos).*

Plataforma	<i>avgRTT (ms)</i>		
	<i>Host 3 (1 destinatário)</i>	<i>Host 3 (3 destinatários)</i>	<i>Variação de desempenho 1-3</i>
ASTRA	7.164	5.133	-28.3%
SPADE	39.678	24.600	-38.0%
JADE	0.490	0.319	-34.9%
JIAC V	23.646	10.202	-43.1%

4.2.4 Benchmark 4

O último *benchmark* (veja nas Figuras 4.11 e 4.12) concentra-se em avaliar a escalabilidade das plataformas selecionadas, aumentando a quantidade de pares de agentes comunicantes. No primeiro cenário (veja na Figura 4.13), é analisado pares de agentes trocando mensagens em um único *host*.

No segundo cenário, o mesmo número de pares de agentes trocam mensagens em uma Rede Local (LAN) (veja na Figura 4.14). Estabelecemos esses dois cenários para avaliar os diferentes mecanismos de comunicação das plataformas, em especial, a plataforma JADE. Além disso, apresentamos resultados da plataforma SPADE com o servidor Jabber/XMPP sem alguns módulos ativados (SPADE(L)) a fim de mostrar o impacto no seu desempenho. Um fato interessante, é que o SPADE em ambas as configurações (com e sem alguns módulos) apresentou tempos similares em experiências de LAN em comparação com Localhost (veja tabelas 4.12 e 4.13 para *avgRTT*). Isso se deve à característica distribuída do servidor Jabber/XMPP uma vez que todas mensagens são encaminhadas ao servidor. Além disso, SPADE(L) tem um *avgRTT* consideravelmente menor em relação ao SPADE, dado o fato de menor robustez no tratamento de mensagem, ao passo que a quantidade de mensagens concluídas é inferior. Embora graficamente não tenha sido evidenciado, JADE apresentou melhor desempenho em uma LAN, como pode ser visto na tabela 4.14. De fato, isso pode estar relacionado ao fato de que, em um sistema distribuído, cada plataforma/contêiner

Figura 4.11: *Benchmark 4 em localhost: Pares de agentes Destinatários e Remetentes. Cada agente Remetente envia 1000 mensagens para o seu respectivo agente Destinatário*

Figura 4.12: *Benchmark 4 em LAN: Pares de agentes Destinatários e Remetentes. Cada agente Remetente envia 1000 mensagens para o seu respectivo agente Destinatário*

tem menos agentes competindo por recursos (*threads*).

Já ASTRA tem um desempenho inferior quando distribuída, enquanto JIAC V tem desempenho similar em experiências de Localhost e LAN.

Figura 4.13: *avgRTT no cenário intraplataforma*

Tabela 4.12: *SPADE avgRTT em Localhost e LAN (em milisegundos).*

pares de agentes	Cenário 1 <i>avgRTT</i> (ms)	Cenário 2 <i>avgRTT</i> (ms)	Variação de desempenho 1-2
20	31.566	26.806	-15.1%
40	87.852	87.306	-0.6%
60	116.320	123.502	-6.2%

Tabela 4.13: *SPADE(L) avgRTT em Localhost e LAN (em milisegundos).*

pares de agentes	Cenário 1 <i>avgRTT</i> (ms)	Cenário 2 <i>avgRTT</i> (ms)	Variação de desempenho 1-2
20	11.749	5.626	-52.1%
40	25.237	12.523	-50.4%
60	39.504	18.512	-53.1%

Figura 4.14: *avgRTT no cenário interplataforma*

Tabela 4.14: *JADE avgRTT em Localhost e LAN (em milisegundos).*

pares de agentes	Cenário 1 <i>avgRTT</i> (ms)	Cenário 2 <i>avgRTT</i> (ms)	Variação de desempenho 1-2
20	0.038	0.019	-50.0%
40	0.037	0.015	-59.4%
60	0.038	0.016	-57.9%

Tabela 4.15: *ASTRA avgRTT em Localhost e LAN (em milisegundos).*

pares de agentes	Cenário 1 <i>avgRTT</i> (ms)	Cenário 2 <i>avgRTT</i> (ms)	Variação de desempenho 1-2
20	0.236	1.063	350.4%
40	0.366	1.385	278.4%
60	0.583	1.711	193.5%

Os resultados apresentados mostram que JADE teve melhor desempenho nos cenários intraplataforma e interplataforma considerando o *avgRTT*. Já SPADE, possui uma camada de transporte que pode ser considerada mais complexa que o mecanismo adotado pelas outras plataformas, devido ao fato de que todas as mensagens tem de ser encaminhadas a um servidor que necessita ser desco-

Tabela 4.16: *JIAC V avgRTT em Localhost e LAN (em milisegundos).*

pares de agentes	Cenário 1 <i>avgRTT</i> (ms)	Cenário 2 <i>avgRTT</i> (ms)	Variação de desempenho 1-2
20	12.009	11.316	-5.8%
40	26.818	25.962	-3.2%
60	40.208	39.924	-0.7%

berto na rede. JIAC V por sua vez, possui um desempenho inferior as outras plataformas baseadas em Java. Isso pode estar relacionado a complexidade criada ao oferecer um agente extremamente modular, com estruturas de dados complexas como a implementada na memória do agente. ASTRA, assim como JIAC V, apresentam como ponto positivo a transparência na comunicação, exigindo minimamente que o programador explicita em código informações sobre os agentes destinatários de mensagens.

4.3 Considerações sobre o capítulo

Neste capítulo, foram analisadas a escalabilidade e o desempenho da camada de transporte de quatro plataformas multiagente: JADE, JIAC V, SPADE e ASTRA. JADE apresentou os melhores resultados em todos os cenários. De fato, a plataforma JADE é bem estabelecida na comunidade multiagente, com um desenvolvimento maduro, possuindo muitas funcionalidades para o desenvolvimento de MAS. Além disso, JADE já foi usada em cenários de larga escala como mostra alguns casos apresentados na Seção 3.4. Em termos de desenvolvimento, JADE além de ser baseada em Java, que é uma linguagem bastante conhecida, não introduz uma sobrecarga significativa no Java RMI, base de sua comunicação interplataforma, como mostra o trabalho [CQV02]. Por fim, notamos que JADE é capaz de prover ferramentas importantes considerando tolerância à falhas e gerenciamento da plataforma, dada as funcionalidades do AMS e do DF.

Já a plataforma ASTRA, apresenta bons resultados considerando todas as plataformas. Como ponto positivo se tratando de desenvolvimento, podemos citar a transparência na comunicação e a integração com o CArtAgO. Por outro lado, o ambiente de desenvolvimento oferece pouca informação/auxílio e a documentação carece de melhor organização. JIAC V, por sua vez, parece introduzir uma sobrecarga, considerando também ser uma plataforma baseada em Java, e seus resultados apresentados. A modularidade derivada do conceito de *Beans* é um ponto positivo, tornando JIAC V, dentre as plataformas, bastante atrelada ao paradigma de Orientação a Objetos. Por outro lado, seria interessante novas funcionalidades relativas aos conceitos de Nodo e execução dos agentes.

Por fim, analisamos também a plataforma SPADE. Apesar da tecnologia Jabber/XMPP ser bem conhecida, a ideia de criar uma plataforma interoperável que ao mesmo tempo é capaz de se comunicar usando uma ACL é bastante interessante. Os resultados de escalabilidade mostram que SPADE apresenta um *avgRTT* superior em quase todos os cenários. Além disso, tanto o *benchmark* 2 e o 3, pode indicar que SPADE é mais suscetível a perda de desempenho considerando os recursos computacionais empregados.

Capítulo 5

Aplicando o resultado da análise de escalabilidade na construção de um MAS escalável

No capítulo anterior foram analisadas a escalabilidade e o desempenho da camada de transporte de mensagens das plataformas JADE, SPADE, JIAC V e ASTRA. Para todos os cenários definidos no *benchmark* de avaliação observou-se melhor desempenho da plataforma JADE. Neste sentido, a construção de um MAS para um cenário onde escalabilidade é um requisito pode ser considerada uma aplicação simples dos resultados da análise.

Considere o cenário descrito no primeiro parágrafo do capítulo introdutório deste texto, aqui reproduzido:

Imagine um cenário onde uma empresa de distribuição de energia elétrica necessita automatizar parte de suas operações de atendimento ao cliente. Tal empresa possui um sistema de informação distribuído que apoia sua operação atual. Para melhorar seus processos e reduzir custos, a empresa espera integrar ao sistema existente a solução de serviço de atendimento ao cliente. Tal serviço possui duas vertentes: atender demandas oriundas diretamente da interação do cliente com o sistema (solicitação de 2a. via de conta, reclamação de falta de energia, solicitação de religação de energia, etc) e antecipar demandas do cliente, a partir de alguma falha no fornecimento de energia. A ideia é atuar proativamente na ocorrência de falha, contactando o cliente e informando-lhe que foi detectado o problema, quais providências foram tomadas e qual o tempo esperado até a solução. Por um lado o cliente/usuário usa uma interface Web para solicitar serviços e o sistema pode usar chatbots para contactar os clientes.

Observa-se que este cenário envolve uma solução onde ocorre a comunicação proativa que visa agrupar e notificar os clientes em caso de interrupção do fornecimento de energia de uma empresa de distribuição de energia elétrica. No caso real, a distribuidora de energia usa o *software* SCADA [Boy09] para monitorar a rede de distribuição, porém o mesmo não é capaz de detectar ocorrências em um único transformador elétrico. Assim, os clientes precisam ligar e solicitar assistência. Se uma falha for detectada em um único transformador elétrico, a probabilidade de todos os clientes conectados ao mesmo transformador estarem sem fornecimento é muito alta. A Figura 5.1 mostra as etapas da distribuição de energia. A "comunicação proativa" refere-se à comunicação com todos os *outros* possíveis afetados por uma falha do serviço, antes que eles solicitem assistência. Esse problema envolve agentes capazes de se comunicar com os clientes e se coordenar para atender as demandas.

O processo de identificação de falha de fornecimento energia a nível de transformadores começa

Figura 5.1: *Etapas de distribuição de eletricidade - especificação funcional [EY18]*

quando o cliente faz um contato com a empresa de distribuição. Quando o cliente fala com um atendente, ele verifica as ocorrências registradas para o local relatado e, nos casos de procedimentos já iniciados, um tempo esperado de normalização do serviço é retornado.

Atualmente, a confirmação de uma ocorrência é feita através de perguntas feitas pelo atendente. O sistema deve automatizar este processo desde o atendimento telefônico até o despacho de uma equipe que irá verificar a ocorrência. Neste processo estão envolvidos: (i) atendimento de um robô e confirmação de ocorrências; (ii) agrupamento de clientes afetados; (iii) um modelo estatístico para previsão de retorno de fornecimento; (iv) identificação de quem deve ser comunicado de acordo com os critérios estabelecidos; (v) comunicação com os clientes afetados por meio de canais de comunicação, como mensagens de texto e ligações telefônicas.

A etapa (i) será executada por um robô com a capacidade de identificar respostas afirmativas (SIM) e negativas (NÃO) por meio da interação com o cliente via ligação telefônica.

Se o problema for resolvido através das perguntas, a chamada é concluída, caso contrário, uma equipe é atribuída ao local informado para realizar a manutenção. Uma vez verificada a ocorrência, o robô fará chamadas para os demais clientes potencialmente afetados, informando o tempo estimado de retorno do fornecimento.

O tempo estimado de retorno é calculado através do registro histórico de ocorrências e considera variáveis como dia da semana, deslocamento e equipes disponíveis, clima, prioridade do cliente (hospitais, escolas e certas atividades comerciais têm prioridade, por exemplo), entre outros. Os clientes que devem ser informados são selecionados com base nos critérios internos da empresa. O cenário de comunicação proativa também é válido para manutenção programada.

5.1 Uma proposta de MAS escalável

A partir da identificação dos requisitos do problema real descrito no início do capítulo, partiu-se para o mapeamento destes requisitos para uma solução utilizando agentes em larga escala. A escalabilidade aqui pode ser considerada em várias situações: (i) no número de agentes envolvidos na ponta do processo (chatbots), no que se refere à comunicação proativa com os clientes; (ii) no número de agentes responsáveis pelo monitoramento do funcionamento da rede de distribuição; (iii) no número de mensagens trocadas entre os agentes responsáveis pelo monitoramento e o agente que coordena as demandas oriundas dos clientes; (iv) no número de agentes de manutenção disponíveis em cada região, dentre outros.

É válido ressaltar que o problema real é bastante complexo, e a solução atual conta com *softwares* proprietários que gerenciam um grande volume de dados adaptados a um cenário específico. Sendo assim, uma solução utilizando agentes é complementar a estas ferramentas de sensoramento, armazenamento de dados, entre outras. No entanto, é possível encontrar soluções para cenários com desafios similares, como por exemplo o apresentado em [BCG15a].

Como parte do processo de mapeamento dos requisitos do problema para uma solução utilizando agentes, o primeiro passo é responder a seguinte pergunta: “por que utilizar agentes?”. Essa é uma pergunta pertinente uma vez que a solução adotada pelas partes envolvidas não inclui agentes levando em consideração as definições deste trabalho sobre o conceito do que é um agente. A solução do problema real é centralizada em um *software* de RPA, que basicamente faz a *orquestração* do sistema no sentido de decidir quando dados serão gerados e/ou transferidos e consumidos. Esse tipo de solução pode ser entendida como um sistema de alto acoplamento, e um dos grandes desafios é a

sua própria coordenação. Nesse sentido, uma solução utilizando MASs consegue ser eficiente devido a sua própria natureza. Dentre as suas características, mas não limitado a elas estão:

- **Escalável:** dado que uma empresa pode atender diversos clientes em várias regiões de atuação e a demanda por agentes requer flexibilidade
- **Agentes inteligentes:** uma vez que agentes precisam tomar decisões dada a percepção do ambiente, que inclui uma variedade de variáveis desde ao clima, deslocamento da equipe até a região afetada, disponibilidade de peças e equipamentos no estoque, prioridade do cliente, regularidade do cliente perante a empresa, entre outras (demanda autonomia)
- **Sistemas Multiagente (MAS):** uma vez que no cenário proposto pode ser empregado desde agentes reativos simples para tarefas como monitoramento da distribuição até agentes proativos para tomada de decisões, como por exemplo decidir quem e quando será comunicado sobre a falta de fornecimento. Além disso, há comunicação interna com o próprio sistema e externa com o cliente, exigindo coordenação entre as entidades envolvidas (habilidade social)
- **Baixo acoplamento:** MAS são sistemas inerentemente distribuídos e de baixa acoplagem. Isso permite que vários agentes executem ações paralelamente em sistemas abertos. O cenário proposto faz uso dessa capacidade uma vez que agentes que podem atuar por exemplo como sensores da rede, enquanto outros podem ser embarcados em dispositivos móveis e ambos agindo paralelamente para atingir um objetivo coletivo e desativados quando não se fazem necessários

5.2 Projeto e Implementação

Como já estabelecido no início do capítulo, a plataforma JADE foi escolhida para o desenvolvimento do protótipo da solução (veja na Figura 5.2). JADE tem uma comunidade ativa que desenvolve soluções que podem ser utilizadas em conjunto com a plataforma. Essas soluções vão desde o uso de agentes em dispositivos móveis até interoperabilidade com outros tipos de serviços. Essas facilidades podem ser úteis uma vez que equipes de manutenção podem estar conectadas ao sistema através de dispositivos móveis, como aparelhos celulares. Para o desenvolvimento dos *Chatbots*, foi utilizado a ferramenta RASA NLU ¹ que é baseada na biblioteca spacy ².

A Figura 5.2 ilustra a arquitetura do protótipo da solução proposta utilizando agentes. Como mencionado anteriormente, a solução atual do problema demanda *softwares* proprietários, processamento de um grande volume de dados, modelos estatísticos, entre outros, que podem ser usados em conjunto com tecnologias de agentes. O objetivo deste protótipo é mapear alguns desses requisitos e apresentar um cenário simples que representa um esboço para a solução do caso da comunicação proativa. Para tanto, é utilizada a ferramenta Rasa NLU em conjunto com a plataforma JADE.

A ferramenta Rasa pode ser facilmente integrada com ferramentas para expor seus agentes em páginas da *Web* e aplicativos. Por essa razão é um meio de interfaceamento para expor uma aplicação que faz uso de agentes. No protótipo da solução, agentes Rasa ³ são capazes de acessar uma base de dados fictícia que armazena dados dos clientes para confirmar suas informações. Ao terminar o atendimento e verificar que se trata de uma ocorrência, o agente Rasa insere o chamado numa fila de ocorrências.

Um agente JADE chamado *Coordenador* por sua vez consome essa fila de ocorrências. O objetivo é encaminhar a ocorrência para um dos agentes *Monitores*, que armazenam dados sobre quais atendimentos já estão em andamento. Quando um agente Monitor recebe uma mensagem do Coordenador, ele busca no DF todos os agentes de *Manutenção* disponíveis. Um agente de Manutenção só está inscrito no DF se está disponível para atender a chamados. O agente Monitor então solicita

¹<https://rasa.com/>

²<https://spacy.io/>

³Implementação disponível em <https://gitlab.com/hdonancio/chatbot-agent/tree/master/chatbot-rasa>

Figura 5.2: Arquitetura do protótipo da solução

ao agente Manutenção mais próximo da ocorrência para atender ao chamado. Um atendimento é simulado através de um intervalo de tempo t , enquanto o agente *Manutenção* processa um valor da sequência de Fibonacci. O agente Manutenção fica indisponível durante o período de atendimento. Um atendimento envolve planejamento, dado que a equipe precisa se deslocar e dispor dos recursos necessários para atender a ocorrência. Para tanto, como forma de simular esse planejamento, cada agente calcula um valor randômico da sequência de Fibonacci em um intervalo predeterminado. Quando o agente finaliza o atendimento, ele se inscreve novamente no DF e comunica ao agente Monitor.

5.3 Testes de escalabilidade da solução proposta

Considerando as características de um MAS, apresentamos um protótipo da solução desenvolvido em JADE utilizando fortemente capacidades relacionadas a especificação FIPA, como o DF. JADE foi escolhida para o desenvolvimento, tanto pelos resultados superiores apresentados em 4.2 quanto pelas funcionalidades presentes. Para avaliar essas funcionalidades, além de uma implementação do protótipo, realizamos testes em um único *host* (ver Tabela 5.1). Nesse ponto, é válido ressaltar que uma avaliação de escalabilidade nos moldes de alguns dos cenários descritos avaliados no capítulo 4 é inviável dada a arquitetura da solução, que distribui agentes em uma quantidade heretogênea de dispositivos.

O objetivo dos testes era estressar a arquitetura do JADE em uma abordagem semelhante a realizada por [BvSW⁺01]. Para tanto, o protótipo apresentado explora algumas características do JADE relativas a FIPA. Uma dessas características é em relação ao registro e busca de serviços no DF. Como apresentado, os agentes estão constantemente fazendo buscas, cadastrando e descadastando serviços no DF. A ideia é verificar se a implementação do DF poderia eventualmente ser um gargalo no sistema. Uma segunda característica do cenário é o processamento de sequências de Fibonacci por cada agente, a fim de sobrecarregar a plataforma, podendo assim estimar um limite de agentes operando simultaneamente. A taxa de ocorrências variava de acordo com a quantidade de agentes no sistema. Essa taxa foi definida em uma ocorrência por segundo para cada agente. A quantidade de agentes Monitores e de Manutenção era igual, sendo a soma de ambos, o total de agentes, mais um agente Coordenador que tem a função de enviar ocorrências.

Tabela 5.1: *Hardware e Software usados nos testes do protótipo.*

Processador	Intel(R) XEON(R) CPU E3-1270 v6 Quad-Core + HT 3.80 GHz
Memória total	8 Gb
Sistema Operacional	Windows 10 Pro
Java	Sun JDK 1.8_211
JADE	4.5

Na avaliação do teste, foi possível executar 1600 agentes, com agentes de Manutenção processando sequências de Fibonacci que variavam de 1 até 40. A quantidade de agentes, por si só, parece não ter muito impacto no desempenho, o que nos leva a crer que uma instância inoperante de um agente na plataforma pode eventualmente consumir *pouco* recurso computacional. Isto foi evidenciado pelo fato de que todas as simulações em que o valor da sequência de Fibonacci calculado possuía um intervalo inferior (menor que 1-40), o uso do processador foi inferior aqueles com a mesma quantidade de agentes. Vale também ressaltar que o fato da plataforma ter executado 1600 agentes simultaneamente, não significa que ela é capaz de permanecer executando por um intervalo de tempo maior livre de falhas.

Capítulo 6

Conclusões

Neste trabalho, foram analisadas algumas das plataformas mais importantes para construir e executar um MAS. JADE é uma plataforma bem conhecida, com algumas aplicações em larga escala na literatura, ao passo que SPADE e ASTRA são plataformas que vem ganhando espaço na comunidade. JIAC, por sua vez, traz novas abordagens para o uso de agentes, principalmente na questão de modularização dos componentes. O objetivo foi compará-las de acordo com seu apoio ao desenvolvimento de agentes em larga escala.

Foi avaliada a escalabilidade das plataformas selecionadas com foco no canal de comunicação. Como *middleware* baseados em Java, JADE herda características importantes da linguagem, sem introduzir uma sobrecarga significativa (ou seja, desenvolvimento orientado a objetos, mecanismos de comunicação). Uma de suas vantagens mais importantes, incluindo também ASTRA e JIAC V, quando comparada com o SPADE, está relacionada à escalabilidade: as plataformas baseadas em Java podem lidar com um número maior de agentes ao mesmo tempo. Isso se deve principalmente às características e mecanismos herdados.

Por outro lado, a camada de transporte de mensagens usada pelo SPADE (Jabber/XMPP) é muito mais adequada para construir sistemas interoperáveis pela Internet. Se o Python apresenta uma limitação em relação aos agentes simultâneos, os servidores Jabber/XMPP podem manipular grandes quantidades de mensagens muito bem. Isso é importante, especialmente no contexto da Internet das Coisas [KCVGAZ15], onde pode ser crucial lidar com um grande número de dispositivos intercomunicantes ao mesmo tempo (como por exemplo em uma Cidade Inteligente (*Smart City*)).

Outras vantagens importantes do SPADE sobre plataformas Java são as preocupações de escalabilidade mencionadas em [BvSW⁺01]. Segundo os autores, existem certas características que devem existir dentro de um MAS para aumentar a tolerância a falhas e a modularidade do sistema como um todo. Em particular, um MAS escalável deve ser flexível em termos de criação, migração e exclusão de agentes no sistema. Os módulos fornecidos por um servidor Jabber/XMPP podem lidar com esses processos muito bem, já que eles podem abstrair o roteamento de mensagens, manter cópias de todas as mensagens e manipular as notificações de presença e disponibilidade entre os agentes.

Apesar das características arquitetônicas mencionadas acima, os resultados mostram um melhor desempenho das plataformas Java em relação ao SPADE, considerando o *avgRTT*. De fato, SPADE parece ser mais suscetível a mudanças de desempenho relacionadas aos recursos computacionais. No entanto, quando se trata de sobrecarga maciça de comunicações sobre um único agente, JADE e ASTRA se destacam das demais plataformas, com JADE obtendo os melhores resultados.

Em um segundo momento, apresentamos um cenário de aplicação em larga escala e discutimos sobre as vantagens que um sistema multiagente oferece para soluções dessa natureza, dentre elas: escalabilidade, heterogeneidade dos agentes que o sistema pode conter, arquiteturas de agentes inteligentes capazes de tomar decisões e baixa acoplagem. Como forma de validar esses conceitos, foi proposto e implementado um protótipo da solução, que explora recursos presentes na especificação FIPA.

6.1 Sugestões para Pesquisas Futuras

Em trabalhos futuros, os seguintes aspectos podem ser explorados:

- **Interoperabilidade entre plataformas aderentes a especificação FIPA:** Mesmo as plataformas seguindo as especificações estabelecidas pela FIPA, devido as particularidades de cada uma, não há comunicação por meio de mensagens FIPA-ACL entre seus agentes. Nesse sentido, o protocolo Jabber/XMPP pode ser útil ao estabelecer um canal de comunicação desacoplado da plataforma. Para tanto, é necessário trabalhos que explorem a implantação de clientes Jabber/XMPP em agentes aderentes a FIPA.
- **Utilização de arquiteturas computacionais de alto desempenho:** MAS são sistemas inerentemente concorrentes e que muitas vezes demandam uma grande quantidade de recursos para sua execução. No entanto, poucos trabalhos se concentram a utilizar recursos e/ou técnicas computacionais de alto desempenho em MAS como o apresentado por [ADF16].
- **Questões de escalabilidade:** No presente trabalho, escalabilidade foi tratada com foco na camada de transporte das plataformas e no seu desempenho geral para implantação de um MAS. No entanto, outros fatores influenciam no desempenho de um sistema, principalmente se tratando de agentes proativos. Na arquitetura BDI por exemplo, o processo de deliberação pode ser muitas vezes custoso. Estudos comparativos entre diferentes plataformas com foco na execução dos agentes podem ser úteis na otimização de recursos por parte de um MAS.

Referências Bibliográficas

- [AD10] Huib Aldewereld e Virginia Dignum. Operetta: Organization-oriented development environment. Em *International Workshop on Languages, Methodologies and Development Tools for Multi-Agent Systems*, páginas 1–18. Springer, 2010. 6
- [ADF16] Tobias Ahlbrecht, Jürgen Dix e Niklas Fiekas. Scalable multi-agent simulation based on mapreduce. Em *Multi-Agent Systems and Agreement Technologies*, páginas 364–371. Springer, 2016. 17, 20, 23, 48
- [BCG15a] Federico Bergenti, Giovanni Caire e Danilo Gotta. Large-scale network and service management with wants. Em *Industrial Agents*, páginas 231–246. Elsevier, 2015. 22, 42
- [BCG15b] Federico Bergenti, Giovanni Caire e Danilo Gotta. A scalable platform for mobile social gaming. Em *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, páginas 2239–2244. ACM, 2015. 17, 22, 23
- [BdSW18] Maurício Fontoura Blos, Robson Marinho da Silva e Hui-Ming Wee. A framework for designing supply chain disruptions management considering productive systems and carrier viewpoints. *International Journal of Production Research*, páginas 1–17, 2018. 7
- [BHW07] Rafael H Bordini, Jomi Fred Hübner e Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007. 5, 11
- [BMO⁺02] F.M.T. Brazier, D.G.A. Mobach, B.J. Overeinder, S. Splunter, van Steen M. e N.J.E Wijngaards. Agentscape: Middleware, resource management, and services. Em *Proceedings of the 3rd International SANE Conference (SANE 2002)*, páginas 403–404, 2002. 18
- [Boy09] Stuart A. Boyer. *Scada: Supervisory Control And Data Acquisition*. International Society of Automation, USA, 4th edição, 2009. 41
- [Bra87] Michael Bratman. Intention, plans, and practical reason. 1987. 6
- [Bro86] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, March 1986. 2, 6
- [BvSW⁺01] FM Brazier, MR van Steen, NJE Wijngaards et al. On mas scalability. 2001. 17, 18, 23, 45, 47
- [CDIPDPC15] Pablo Chamoso, Fernando De la Prieta, Francisco De Paz e Juan M Corchado. Swarm agent-based architecture suitable for internet of things and smartcities. Em *Distributed Computing and Artificial Intelligence, 12th International Conference*, páginas 21–29. Springer, 2015. 17, 20, 23

- [CL90] Philip R Cohen e Hector J Levesque. Intention is choice with commitment. *Artif. Intell.*, 42(2-3):213–261, 1990. 6
- [CQV02] Elisabetta Cortese, Filippo Quarta e Giosue Vitaglione. Scalability and performance of jade message transport system. 2002. 2, 17, 18, 23, 28, 39
- [CRL15] Rem W Collier, Seán Russell e David Lillis. Reflecting on agent programming with agentspeak (1). Em *International Conference on Principles and Practice of Multi-Agent Systems*, páginas 351–366. Springer, 2015. 2
- [CTG⁺04] Krzysztof Chmiel, Dominik Tomiak, Maciej Gawinecki, Pawel Karczmarek, Michal Szymczak e Marcin Paprzycki. Testing the efficiency of jade agent platform. Em *Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, páginas 49–56. IEEE, 2004. 17, 18, 19, 23
- [Det01] Ralph Deters. Scalability & multi-agent systems. Em *2nd International Workshop Infrastructure for Agents, MAS and Scalable MAS. 5th int. conference on Autonomous Agents*, 2001. 14, 17, 18, 23
- [Erl08] Thomas Erl. *Soa: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008. 8
- [EY18] EY. Modelagem de segmentação de clientes, previsão do retorno de energia elétrica e callback, 2018. ix, 42
- [Fer99] Jacques Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999. 1, 2, 5
- [FFMM94] Tim Finin, Richard Fritzson, Don McKay e Robin McEntire. Kqml as an agent communication language. Em *Proceedings of the Third International Conference on Information and Knowledge Management, CIKM '94*, páginas 456–463, New York, NY, USA, 1994. ACM. 7
- [FGLO10] Victor Fernández, Francisco Grimaldo, Miguel Lozano e Juan M Orduna. Evaluating jason for distributed crowd simulations. Em *ICAART (2)*, páginas 206–211, 2010. 17, 21, 23
- [FGM03] Jacques Ferber, Olivier Gutknecht e Fabien Michel. From agents to organizations: an organizational view of multi-agent systems. Em *International workshop on agent-oriented software engineering*, páginas 214–230. Springer, 2003. 6
- [GC04] Dominic Greenwood e Monique Calisti. Engineering web service-agent integration. Em *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 2, páginas 1918–1925. IEEE, 2004. 2
- [GCB06] Miguel Escrivá Gregori, Javier Palanca Cámara e Gustavo Aranda Bada. A jabber-based multi-agent system platform. Em *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, páginas 1282–1284. ACM, 2006. 2, 10
- [HSB07] Jomi F Hubner, Jaime S Sichman e Olivier Boissier. Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3-4):370–395, 2007. 6, 11, 12, 20
- [Huh02] Michael N Huhns. Agents as web services. *IEEE Internet computing*, 6(4):93–95, 2002. 2

- [IM19] Prosody IM. <https://prosody.im/>, 2019. Último acesso em 24/05/2019. xi, 30
- [Ita19] Telecom Italia. <https://jade.tilab.com/doc/tutorials/JADEAdmin/JadeContainerTutorial.html>, 2019. Último acesso em 24/05/2019. ix, 10
- [Jen00] Nicholas R Jennings. On agent-based software engineering. *Artificial intelligence*, 117(2):277–296, 2000. 2
- [KB15] Kalliopi Kravari e Nick Bassiliades. A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015. 2, 25
- [KCVGAZ15] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego e Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud computing*, 3(1):11–17, 2015. 10, 47
- [KN80] Kurt Konolige e Nils J Nilsson. Multiple-agent planning systems. Em *AAAI*, volume 80, páginas 138–142, 1980. 6
- [KPK18] Renuka Kamdar, Priyanka Paliwal e Yogendra Kumar. A state of art review on various aspects of multi-agent system. *Journal of Circuits, Systems and Computers*, página 1830006, 2018. 7
- [Lab01] Yannis Labrou. Standardizing agent communication. Em *ECCAI Advanced Course on Artificial Intelligence*, páginas 74–97. Springer, 2001. 7, 8
- [LK13] Shujun Li e Mieczyslaw M. Kokar. *Agent Communication Language*, páginas 37–44. Springer New York, New York, NY, 2013. 7
- [LKK⁺13] Marco Lützenberger, Tobias Küster, Thomas Konnerth, Alexander Thiele, Nils Masuch, Axel Heßler, Jan Keiser, Michael Burkhardt, Silvan Kaiser e Sahin Albayrak. Jiacc v: A mas framework for industrial applications. Em *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, páginas 1189–1190. International Foundation for Autonomous Agents and Multiagent Systems, 2013. 2
- [LNNDW98] Lyndon C Lee, Hyacinth S Nwana, Divine T Ndumu e Philippe De Wilde. The stability, scalability and performance of multi-agent systems. *BT Technology Journal*, 16(3):94–103, 1998. 14
- [LSY09] Dong Yeol Lee, Seung Yeop Shin e Hee Yong Youn. Replication decision mechanism for service-oriented multi-agent system. Em *2009 Congress on Services-I*, páginas 762–768. IEEE, 2009. 17, 21, 23
- [LW05] Till C Lech e Leendert WM Wienhofen. Ambieagents: a scalable infrastructure for mobile and context-aware information services. Em *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, páginas 625–631. ACM, 2005. 17, 22, 23
- [MSA06] Luis Mulet, Jose M Such e Juan M Alberola. Performance evaluation of open-source multiagent platforms. Em *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, páginas 1107–1109. ACM, 2006. 17, 18, 23
- [ON98] Paul D O’Brien e Richard C Nicol. Fipa—towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998. 7
- [PCGLO14] Pascual Pérez-Carro, Francisco Grimaldo, Miguel Lozano e Juan M. Orduña. Characterization of the jason multiagent platform on multicore processors. *Sci. Program.*, 22(1):21–35, Janeiro 2014. 17, 20, 23

- [PM08] Michal Pěchouček e Vladimír Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems*, 17(3):397–431, Dec 2008. 6
- [RABA18] Silviu Răileanu, Florin Daniel Anton, Theodor Borangiu e Silvia Anton. Design of high availability manufacturing resource agents using jade framework and cloud replication. Em *Service Orientation in Holonic and Multi-Agent Manufacturing*, páginas 201–215. Springer, 2018. 7
- [Rao96] Anand S Rao. Agentspeak (1): Bdi agents speak out in a logical computable language. Em *European workshop on modelling autonomous agents in a multi-agent world*, páginas 42–55. Springer, 1996. 11
- [RCS18] G. Radhakrishnan, V. Chithambaram e K.L Shunmuganathan. Comparative study of jade and spade multi agent system. 2018. 17, 19, 23
- [RG91] Anand S. Rao e Michael P. Georgeff. Modeling rational agents within a BDI-architecture. Em James Allen, Richard Fikes e Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, páginas 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991. 6
- [RN03] Stuart J. Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edição, 2003. 1, 2, 5
- [RS00] Omer F Rana e Kate Stout. What is scalability in multi-agent systems? Em *Proceedings of the fourth international conference on Autonomous agents*, páginas 56–63. ACM, 2000. 14
- [RVO06] Alessandro Ricci, Mirko Viroli e Andrea Omicini. Cartago: A framework for prototyping artifact-based environments in mas. Em *International Workshop on Environments for Multi-Agent Systems*, páginas 67–86. Springer, 2006. 11, 12
- [SAM⁺07] Jose M Such, Juan M Alberola, Luis Mulet, Agustin Espinosa, Ana Garcia-Fornes e Vicent Botti. Large-scale multiagent platform benchmarks. 2007. 2, 17, 23, 25
- [Sim96] Herbert A Simon. *The sciences of the artificial*. 1996. 2
- [SMM⁺09] Yue Suo, Naoki Miyata, Hiroki Morikawa, Toru Ishida e Yuanchun Shi. Open smart classroom: Extensible and scalable learning system in smart space using web service technology. *IEEE transactions on knowledge and data engineering*, 21(6):814–828, 2009. 17, 22, 23
- [SS69] John R Searle e John Rogers Searle. *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press, 1969. 7
- [Tea18] JIAC Development Team. Jiac manual. http://jiac.de/Downloads/jiac/jiac_manual.pdf, 2018. Último acesso em 21/08/2019. ix, 13
- [TJ00] Phillip J Turner e Nicholas R Jennings. Improving the scalability of multi-agent systems. Em *Workshop on Infrastructure for Scalable Multi-Agent Systems at the International Conference on Autonomous Agents*, páginas 246–262. Springer, 2000. 14, 17, 21, 23
- [TRBC09] Dante I Tapia, Sara Rodríguez, Javier Bajo e Juan M Corchado. Fusion@, a soa-based multi-agent architecture. Em *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, páginas 99–107. Springer, 2009. 2

- [UFR19] Prosoft UFRGS. Arquitetura bdi. <http://www.inf.ufrgs.br/prosoft/bdi4jade/>, 2019. Último acesso em 24/05/2019. ix, 2, 7
- [Vrb03] Pavel Vrba. Java-based agent platform evaluation. Em *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, páginas 47–58. Springer, 2003. 17, 19, 23
- [vST17] M. van Steen e A.S. Tanenbaum. *Distributed Systems*. CreateSpace Independent Publishing Platform, 2017. 1
- [WJ95] Michael Wooldridge e Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995. 5
- [WM15] Danny Weyns e Fabien Michel. Agent environments for multi-agent systems—a research roadmap. Em *Agent Environments for Multi-Agent Systems IV*, páginas 3–21. Springer, 2015. 17, 19, 23
- [Woo99] M. Wooldridge. Intelligent agents. Em *Multiagent Systems*. 1999. 6
- [Woo09] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009. 5, 11