Catarina Viegas de Freitas Moura

Licenciada em Ciências da Engenharia Electrotécnica e de Computadores

A JADE-Based Tool to Assess the Network Dynamics of Mechatronic Multiagent Systems

Dissertação para obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador : José António Barata de Oliveira, Professor Doutor, UNINOVA/CTS, FCT-UNL

Co-orientador : Luís Domingos Ferreira Ribeiro, Professor Doutor, Linköping University

Júri:

Presidente: Prof. Doutor Tiago Oliveira Cardoso

Arguente: Prof. Doutor João Paulo Pimentão

Vogal: Prof. Doutor José António Barata

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2014

A JADE-Based Tool to Assess the Network Dynamics of Mechatronic Multi-agent Systems

Aos meus ***Pais*** *e à minha* ***Irmã***

Agradecimentos

Em primeiro lugar, como não poderia deixar de ser, agradeço aos meus Pais por todas as oportunidades que me têm proporcionado e, nomeadamente, pela possibilidade de realizar o meu curso superior. Cheguei à concretização desta Tese devido a todo o apoio que sempre me deram ao longo destes anos de formação académica, acreditaram sempre em mim e por isso o sentimento que imprimo nestas linhas é apenas uma amostra da enorme gratidão que lhes tenho.

À minha querida irmã, também tenho de agradecer, por toda a paciência, força e incentivo que nunca deixou de me dar, enquanto escrevi e trabalhei neste projeto. A sua companhia de extrema dedicação, compreensão e empenho fortaleceu-me a todos níveis e sem ela, todos os momentos de dificuldade teriam sido muito mais difíceis de ultrapassar.

Ao meu orientador Professor Doutor José Barata, quero agradecer pela oportunidade e voto de confiança que me deu ao ter-me aceite como sua mestranda no desenvolvimento deste projeto e por toda a disponibilidade que sempre demonstrou ao longo destes anos de formação académica.

Ao meu co-orientador Professor Doutor Luís Ribeiro tenho muito a agradecer, por todo o trabalho e dedicação que me dispensou em todos momentos, ajudou-me incondicionalmente e para mim foi um privilégio ter a oportunidade de trabalharmos juntos. O seu profissionalismo, perfecionismo, caráter e dedicação são alvo da minha admiração.

Ao colega João Dias Ferreira que foi também essencial ao sucesso deste projeto, agradeço muito todo o seu apoio e disponibilidade durante a validação do trabalho desenvolvido.

Depois da minha família, professores e colega, mas não menos importantes no alcance deste objetivo, ao Tiago Moreira, ao Tiago Peralta, ao Daniel Marques, ao André Garrido e à Joana Robalo.

Ao Tiago Moreira, por tudo aquilo que representa para mim, tenho de agradecer por toda a força, carinho, paciência e compreensão em todas as fases deste projeto.

Para os meus amigos Tiago Peralta, Daniel Marques, André Garrido e Joana Robalo deixo aqui o meu profundo agradecimento, por todo o companheirismo e apoio, nos bons e nos piores momentos, mas principalmente, por toda a amizade.

Sinto-me uma privilegiada por ter beneficiado do apoio de todos e, por isso, muito obrigada!

Resumo

O aumento das exigências e da procura diversificada nos domínios da produção tem despoletado constantes mudanças nos domínios da manufatura. Este cenário tem impedido que os sistemas tradicionais reúnam os requisitos necessários para suportar estas mudanças.

Novos paradigmas têm surgido nos domínios da manufatura para suportar a evolução da mesma. Estes paradigmas pretendem fornecer aos sistemas de produção moderna, uma melhor capacidade de resposta, flexibilidade, robustez e reconfiguração.

No entanto, sistemas com estas caraterísticas implicam a necessidade de prever o seu comportamento em vários cenários de produção de modo a evitar falhas. Neste sentido, é necessário estudar e compreender o comportamento destes sistemas. Os métodos que permitem o estudo de sistemas distribuídos são ainda muito escassos.

A presente dissertação propõe uma arquitetura multiagente que suporta uma ferramenta para inferir a rede de agentes em qualquer sistema multiagente. A ferramenta desenvolvida, através das redes que infere, permite estudar a dinâmica das interações, normalmente camuflada neste tipo de sistemas, que contribui para a emergência de características desejadas e indesejadas.

A ferramenta proposta efetua a inferência da rede multiagente com base nas interações que ocorrem entre os agentes em tempo real e codifica a informação resultante dessas interações num formato compatível com ferramentas especializadas em análise de redes.

Para comprovar e testar a eficácia e a aplicabilidade da ferramenta desenvolvida, considerou-se a simulação e a análise de um Sistema Evolutivo de Produção. A utilização da ferramenta no estudo e na observação do sistema permitiu testar o desempenho da ferramenta num ambiente altamente dinâmico e validar o modo de funcionamento do próprio sistema.

Palavras Chave: Ferramenta de Inferência, Sistemas Multiagente, Auto-organização, Interações de Agentes, Redes de Agentes, Visualização de Redes de Agentes.

Abstract

The increase in requirements and diversified demand in the fields of production has been triggering constant changes in manufacturing areas. This scenario has prevented traditional systems to meet the needed requirements to support these changes.

New paradigms have been emerging in the fields of manufacturing in order to support the evolution of manufacturing. These paradigms intend to supply modern production systems with a better responsiveness, flexibility, robustness and reconfiguration.

However, systems with these characteristics imply the need to predict their behavior in various production scenarios in order to avoid failures. In this sense, it is necessary to study and understand the behavior of these systems. Methods that allow the study of distributed systems are still very scarce.

This thesis proposes a multiagent architecture that supports a tool to infer the agents' network in any multiagent system. The developed tool, through the networks that infers, allows to study the interactions' dynamics, often hidden in this type of systems, that contributes to the emergence of desired and undesired features.

The proposed tool infers multiagent networks based on the interactions that occur between agents in real time and codifies the information resultant of those interactions in a compatible format with specialized tools for network analysis.

To prove and test the effectiveness and applicability of the developed tool, an instance of an Evolvable Production System was simulated and analyzed by the tool. The usage of tool in the study and observation of the system allowed to test the performance of the tool in a highly dynamic environment and to validate the system's operation.

Keywords: Inference Tool, Multiagent Systems, Self-Organization, Agent's Interactions, Agent's Networks, Agent's Networks Visualization.

Acrónimos

ACL Linguagem de Comunicação de Agentes,
do inglês *Agent Communication Language*

AID Identificador de Agente,
do inglês *Agent Identifier*

AMS Agente de Gestão do Sistema,
do inglês *Agent Management System*

BMS Sistemas Biónicos de Manufatura,
do inglês *Bionic Manufacturing Systems*

CIM Manufatura Integrada por Computador,
do inglês *Computer Integrated Manufacturing*

DF Facilitador de Diretório,
do inglês *Directory Facilitator*

EAS Sistemas Evolutivos de Manufatura,
do inglês *Evolvable Assembly Systems*

ENS Serviço de Notificação de Eventos,
do inglês *Event Notification Service*

EPS Sistemas Evolutivos de Produção,
do inglês *Evolvable Production Systems*

HMS Sistemas Holónicos de Manufatura,
do inglês *Holonic Manufacturing Systems*

IADE *IDEAS Agent Development Environment*

MAS Sistemas Multiagente,
do inglês *Multiagent System*

mPOEMS *Multi-objective Prototype Optimization with Evolved iMprovement Steps*

FIPA *Foundation for Intelligent Physical Agents*

FIPA-ACL Linguagem de Comunicação de Agentes FIPA,
do inglês *FIPA Agent Communication Language*

FIPA-SL Linguagem de Semântica FIPA,
do inglês *FIPA Semantic Language*

FA *Firefly Algorithm*

GML *Graph Modelling Language*

GUI Interface Gráfica do Utilizador,
do inglês *Graphical User Interface*

JADE *Java Agent Development Framework*

PROSA *Product-Resource-Order-Staff Architecture*

RMA Agente de Monitorização Remota,
do inglês *Remote Monitoring Agent*

RMS Sistemas Reconfiguráveis de Manufatura,
do inglês *Reconfigurable Manufacturing Systems*

UML *Unified Modeling Language*

VTA *Visualization Tool Agent*

VAMD *Visualization Agent for Mobile Devices*

XML *eXtensible Markup Language*

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Descrição do Problema

A constante mudança e inovação da tecnologia tem-se feito sentir nas mais variadas áreas de investigação. Esta mudança tem provocado alterações significativas nos ambientes industriais. Para fazer face à concorrência, o mercado industrial tem apostado numa produção fundamentalmente caraterizada pela rapidez e eficiência.

No entanto, os problemas que se colocam atualmente às empresas não se limitam apenas à quantidade e à rapidez da produção, mas também à capacidade que estas têm de se adaptar à constante mudança na manufatura.

Devido à rigidez das estruturas de controlo dos atuais sistemas de produção, estes tendem a apresentar uma capacidade diminuída em termos de flexibilidade e adaptabilidade em situações imprevistas nos ambientes de produção. Como tal, a reestruturação destes sistemas tornou-se inevitável.

Assim, as transformações sentidas nos ambientes da indústria de produção têm desencadeado vários modelos de novos sistemas.

Os novos modelos pretendem tornam o ambiente de produção mais reconfigurável e adaptável reduzindo o tempo de paragem associado a todo o tipo de perturbações. Estas

perturbações podem ser falhas ou alterações na produção (introdução/remoção de produtos, alterações nos volumes de produção, etc...).

O paradigma de Sistemas Multiagente (MAS) tem apresentado a capacidade de lidar com este desafio. Uma das vantagens deste tipo de sistemas é a natureza distribuída dos seus componentes. Esta distribuição confere aos MAS a robustez necessária para lidar com situações de falhas e, por outro lado, facilidade de reconfiguração uma vez que cada uma das suas entidades constituintes funciona de forma auto-contida eliminando pontos centrais de decisão ou de falha.

O funcionamento proposto pelas arquiteturas dos MAS é por vezes considerado de difícil compreensão devido à quantidade e dinâmica das interações que se desenvolvem nestes sistemas.

Torna-se portanto evidente a necessidade de desenvolver ferramentas focadas no estudo e na observação destes sistemas, que contribuam para uma compreensão fidedigna sobre determinados aspetos que os caraterizam. Tais como, os seus padrões de interação emergentes e o modo como as suas entidades constituintes se auto-organizam e cooperam entre si para atingirem um objetivo global.

Como tal, a presente dissertação propõe, descreve e valida uma ferramenta que permite a dedução, o estudo e a observação das redes de interação num MAS.

1.2 Aspetos de Investigação e Proposta de Trabalho

Neste contexto, é conveniente analisar e ponderar as seguintes questões que se relacionam com o desenho e com o desenvolvimento da ferramenta:

1. É possível observar e capturar em tempo real a totalidade das interações num MAS no intuito de conhecer em detalhe o seu modo de funcionamento?

2. Qual a arquitetura adequada à projeção de uma ferramenta que tenha a capacidade de observar e armazenar os eventos que ocorrem num ambiente multiagente sem comprometer o seu funcionamento e recolhendo informação suficiente para a sua posterior análise?

A presente dissertação foca-se no estudo de MAS implementados sobre a tecnologia *Java Agent Development Framework* (JADE) [JAD14]. A ferramenta desenvolvida permite inferir a rede que resulta das interações que ocorrem entre as entidades que constituem estes sistemas.

Tratando-se de uma biblioteca que permite a implementação de MAS, a plataforma JADE tem sido amplamente utilizada em várias áreas de investigação e no desenvolvimento de protótipos industriais. Esta plataforma fornece uma abordagem lógica à programação de agentes na linguagem *JAVA* e garante uma interação robusta e estruturada entre os agentes [RCB+11].

Neste sentido, o JADE é igualmente umas das ferramentas com um grau de maturidade relativamente alto e, como tal, candidata a ser utilizada em ambientes industriais na implementação de sistemas de automação inteligentes.

A inferência da rede de agentes deve ser feita independentemente do contexto da aplicação das entidades do sistema. Neste sentido, a ferramenta permite desmascarar determinados padrões de interação mediante a captura de interações entre os agentes e posterior processamento das mesmas.

Após o processamento da informação sobre as interações deduz-se a rede de agentes, o que torna possível a extração de caraterísticas estruturais (grau médio, coeficiente de agrupamento, número de ligações, frequência de ligações) através de software especializado. Estas caraterísticas podem posteriormente ser relacionadas com o domínio de aplicação no sentido de estabelecer correlações entre o comportamento do sistema e o seu ambiente de atuação.

1.3 Visão Geral do Trabalho Desenvolvido

No sentido de capturar as interações e inferir a rede de agentes são necessárias três fases.

A primeira fase inclui a captura e armazenamento de todas as mensagens trocadas entre os agentes do sistema. Sempre que surge uma nova mensagem no sistema esta é automaticamente capturada e seguidamente armazenada.

Na segunda fase procede-se ao processamento das mensagens capturadas mediante a filtragem e o agrupamento das mesmas.

Na filtragem das mensagens é efetuada a seleção das mensagens relevantes à inferência da rede de agentes. A partir das mensagens selecionadas são criados os nós que representam os agentes do sistema e os recetores e os emissores das mensagens são agrupados consoante as suas interações. Cada agente no sistema é representado por um nó e a interação traduzida pela troca de uma mensagem entre dois agentes representa uma ligação.

O processamento das mensagens capturadas pode ser feito durante a execução do sistema. No entanto, durante o desenvolvimento da ferramenta, o processamento em tempo real não revelou os resultados esperados. Como tal, optou-se por efetuar o processamento das mensagens após a execução do sistema no sentido de minimizar o impacto no desempenho da plataforma JADE e, por sua vez, aumentar o desempenho da ferramenta desenvolvida.

Por fim, na terceira fase traduz-se a informação processada para gerar a rede de agentes. Esta tradução baseia-se na criação de um ficheiro do tipo *Graph Modelling Language* (GML) constituído por todos os nós e ligações do sistema. Este ficheiro, quando terminado, é disponibilizado ao utilizador para que o mesmo o possa importar para software especializado em análise de redes.

A arquitetura da ferramenta desenvolvida é constituída por cinco módulos de funcionamento. O primeiro módulo implementa a entidade principal do sistema que monitoriza os comportamentos dos restantes módulos. O módulo de memória armazena todas as mensagens capturadas para que estas possam ser posteriormente processadas. O módulo de filtragem efetua uma separação das mensagens relevantes e não relevantes à inferência da rede. O módulo de agrupamento agrupa os nós e as ligações do sistema consoante as interações capturadas. E finalmente, o módulo de tradução traduz os dados recolhidos e processados num formato compatível com ferramentas especializadas em análise de redes.

Para validar a arquitetura e a ferramenta considerou-se um caso de teste no âmbito dos Sistemas Evolutivos de Produção (EPS). Um EPS tem a capacidade de alterar o seu comportamento de forma a adaptar-se de modo autónomo e dinâmico à mudança das condições de operação.

Neste contexto, um EPS pode sofrer alterações estruturais. Estas alterações, que resultam do processo de adaptação, influenciam positiva ou negativamente o comportamento do sistema. A sua caraterização e estudo são de particular importância.

1.4 Contribuições Essenciais

Para o estudo e implementação de MAS têm sido desenvolvidas várias ferramentas e métodos que visam contribuir para a compreensão destes sistemas.

De entre estas contribuições é relevante destacar o agente *Sniffer* nativo da plataforma JADE e a aplicação *Java Sniffer* desenvolvida pela *Rockweel Automation, Inc.*

Ambas são consideradas ferramentas de grande importância no estudo e na análise de MAS e permitem a captura e a observação das interações que ocorrem em tempo real num MAS.

No entanto, apesar do seu valor apresentam ainda algumas limitações. Em particular, ambas as ferramentas denotam perda de mensagens durante o processo de captura quando a troca de mensagens na rede é muito intensa.

Assim, procurou-se solucionar algumas destas limitações mediante o desenvolvimento de uma nova ferramenta. A ferramenta proposta, faz, contrariamente às ferramentas existentes, o processamento da rede após a execução do sistema. Desta forma é possível agilizar os processos na plataforma do JADE evitando a representação gráfica das mensagens em tempo real e reduzindo o armazenamento de mensagens para representação.

Esta abordagem permite capturar um número muito superior de mensagens eliminando a perda nas listas de mensagens quando estas atingem o limite máximo de armazenamento.

As presentes arquitetura e ferramenta geram ainda a rede num formato que permite a sua análise por ferramentas especializadas nomeadamente a ferramenta *Gephi* [GEP14].

Esta ferramenta permite estudar as interações entre os agentes de uma forma muito mais abrangente do que as ferramentas tradicionais.

Capítulo 2

Estado da Arte

A Manufatura Integrada por Computador (CIM) sempre promoveu o conceito de que os novos sistemas iriam com certeza ser flexíveis, de fácil integração, e com grande capacidade de resposta [Tha96].

No entanto, a base deste conceito centrou-se apenas nas capacidades da computação de manufatura. Mediante alterações na implementação destes sistemas seria possível otimizá-los de modo a conseguir lidar com as futuras exigências [Tha96].

Contudo, os recentes desenvolvimentos no domínio da manufatura têm provocado profundas mudanças nos seus paradigmas. A produção tem vindo a experimentar um grande aumento na sua variedade e uma acentuada redução no volume [Tha96].

Para garantir uma produção diversificada e com volumes reduzidos, os novos sistemas de produção têm de ter uma elevada flexibilidade operacional que permita a fácil reconfiguração dos seus componentes.

Esta reconfigurabilidade implica que os novos sistemas sejam flexíveis ao ponto de conseguirem alterar rapidamente a estrutura do *shop-floor* industrial através da adição e da remoção de componentes, reduzindo substancialmente os tempos de *ramp-up* e *setup* da produção.

Como tal, para acompanhar a evolução na manufatura, não basta que os sistemas tradicionais sejam reprogramáveis, tal como proposto pela abordagem CIM [Tha96].

Pois, a rigidez das estruturas de controlo hierárquicas dos sistemas tradicionais continua a ser um aspeto desfavorável. Em situações de mudanças ou perturbações no ambiente, esta rigidez tende a dificultar a rápida reconfiguração destes sistemas [Tha96].

Assim, torna-se necessário começar a apostar no desenvolvimento de arquiteturas de controlo distribuídas, facilmente reconfiguráveis, que não impliquem um enorme investimento, extensos prazos de execução e o desenvolvimento de sistemas rígidos [Tha96].

2.1 Arquiteturas e Sistemas Modernos de Produção

O conceito de negócio tem sofrido alterações. Uma produção eficaz no que toca aos custos associados, rapidez de produção, qualidade e outros aspetos relacionados com a reestruturação tecnológica dos tradicionais sistemas de manufatura têm sido considerados os requisitos necessários para o sucesso da modernização na produção industrial [NFO+13].

O nível de personalização que se impõe recentemente aos mercados industriais tem aumentado significativamente nos últimos anos. As novas condições de mercado estão a promover a produção em massa de produtos cada vez mais personalizados e diversificados [BB03].

Neste sentido, para lidar com a diversidade, personalização, variação no volume da produção e com os mercados imprevisíveis, foram introduzidos novos paradigmas no âmbito da manufatura [NFO+13].

Os novos paradigmas visam a utilização de arquiteturas de controlo distribuídas que suportam sistemas de controlo escaláveis e interoperáveis, módulos de produção *plug&play* e ambientes de produção ágeis e com capacidade de resposta [NFO+13].

O encapsulamento das funcionalidades de cada módulo e a facilidade de integração nos sistemas de produção são caraterísticas comuns aos novos paradigmas [NFO+13].

Posto que cada módulo é composto por hardware e software heterogéneo, esta composição representa um aspeto fundamental na interoperabilidade entre os sistemas e na reutilização dos equipamentos [NFO+13].

A resposta dinâmica e adaptável à mudança é, nos dias de hoje, a chave para a competitividade. Assim sendo, surge uma nova classe de sistemas inteligentes e distribuídos de controlo de manufatura que opera de forma totalmente diferente das abordagens tradicionais [Lei09].

Esta nova classe visa o desenvolvimento de arquiteturas com base na inovação, agilidade e reconfiguração e, utiliza para tanto, os paradigmas emergentes e várias tecnologias que fornecem soluções viáveis em situações de falha e de difícil resolução [Lei09].

De entre os vários paradigmas emergentes nos domínios da manufatura, destacam-se os paradigmas que suportam as tecnologias de agentes. Estes paradigmas têm tido um impacto acentuado no desenvolvimento de novos sistemas e têm contribuído significativamente para a evolução da manufatura [RCB+11].

2.1.1 Sistemas Multiagente

A tecnologia de agentes é considerada como um suporte relevante no desenvolvimento de sistemas de manufatura distribuídos e inteligentes. Esta tecnologia fornece uma alternativa viável para ultrapassar os problemas associados aos tradicionais sistemas centralizados de organização hierárquica [SN99].

Segundo *Wooldridge* e *Jennings* em [WJ95], o conceito de agente encontra-se diretamente ligado a propriedades com a autonomia, a sociabilidade, a reatividade e a proatividade.

A autonomia deriva da capacidade dos agentes operarem sem a intervenção de algum tipo de ação externa e do fato de apresentarem algum controlo sobre as suas próprias ações e sobre o seu estado interno. A sociabilidade deve-se às interações entre os agentes através das mensagens que trocam e da linguagem específica que utilizam para o fazer. Os agentes adaptam-se e reagem oportunamente às mudanças no seu meio ambiente, o que resulta diretamente na reatividade que lhes é caraterística. Por fim, a proatividade que estas entidades apresentam advém do fato de demonstrarem nos comportamentos que executam, os seus objetivos ou as metas que pretendem atingir [WJ95].

Os agentes auto-organizam-se de forma dinâmica e adaptável. Este mecanismo faculta a um sistema a capacidade de adquirir, manter e mudar a sua organização sem qualquer tipo de comando externo explícito, durante o seu tempo de execução. Não existe portanto controlo centralizado ou hierárquico. É essencialmente uma reorganização dinâmica e espontânea da estrutura ou composição do sistema [FS11].

Tendo em atenção que cada entidade autónoma possui um visão parcial do sistema, é importante referenciar a capacidade de comunicação que possui e que lhe permite atingir um objetivo predefinido ou de solucionar um problema. Isto requer que os agentes sejam capazes de se compreender utilizando para esse fim linguagens de comunicação apropriadas, ontologias e protocolos de interação [Lei09].

A aplicabilidade dos MAS tem progredido e evoluído nas mais diferenciadas áreas. Entre estas evidenciam-se o comércio eletrónico, o negócio, o controlo de tráfego aéreo, o controlo de processo, as telecomunicações, etc... [Lei09] Na área da manufatura, estes sistemas destacam-se nos domínios da integração de sistemas e gestão da cadeia de produção, planeamento, agendamento e controlo de execução, manuseamento de materiais, gestão de

inventário e desenvolvimento de novos paradigmas [SN99].

As arquiteturas distribuídas dos MAS e as propriedades que caraterizam as entidades que as constituem tornaram estes sistemas a ferramenta indicada para a implementação de viáveis, robustos e flexíveis sistemas de manufatura [BC06].

Os recentes desenvolvimentos na tecnologia de agentes têm permitido a implementação de sistemas escaláveis com um elevado grau de inovação. Pois, por um lado têm vindo a convergir com a recente tecnologia de software industrial, por outro, com os novos paradigmas na manufatura [MVK06].

Sistemas Holónicos de Manufatura

O autor *Arthur Koestler* inventou a palavra "*holon*"na tentativa de capturar o comportamento de sistemas complexos. Para *Koestler* um *holon* é simultaneamente uma parte e um todo, no contexto de um determinado sistema, e constitui uma unidade básica de organização em sistemas biológicos e sociais [BC06].

A noção de *holon* está na origem do conceito de Sistemas Holónicos de Manufatura (HMS) e descreve a natureza híbrida de subconjuntos/partes em sistemas. Um *holon* pode ser definido como um bloco autónomo e cooperativo de um sistema de manufatura para transformar, transportar, armazenar e/ou validar informação e objetos físicos. Um *holon* pode constituir parte de outro *holon* [BC06].

Koestler propôs ainda uma arquitetura formada por *holons*, designada de holarquia e associada ao conceito de uma hierarquia *open-ended*, sem limitações ascendentes ou descendentes. Uma holarquia é portanto um sistema constituído por *holons* que cooperam entre si para atingirem determinada meta ou objetivo [BC06].

A arquitetura *Product-Resource-Order-Staff Architecture* (PROSA) representa a arquitetura de referência em HMS. A sua composição apresenta três tipos de *holons* básicos

denominados de *Resource Holons*, *Product Holons* e *Order Holons*. Os *Staff Holons*, não sendo entidades obrigatórias na estrutura da arquitetura, podem ainda representar um papel importante através da sua assistência aos *holons* básicos no cumprimento das suas tarefas [MVK06].

Os conceitos de agregação e especialização formulam a base estrutural dos *holons* que constituem a arquitetura PROSA [VBWV$^+$98].

A agregação é definida através do agrupamento de *holons* que se relacionam entre si e formam um *holon* maior. Dependendo do ponto de vista do observador, os *holons* são separados em *sub-holons* ou podem ser vistos como um todo. A agregação entre *holons* pode mudar dinamicamente dependendo das necessidades do sistema [MVK06].

Posto que cada *holon* é responsável pelo desempenho de determinada tarefa no sistema de controlo de manufatura, o conceito de especialização permite a separação dos *holons* consoante as suas caraterísticas ou tarefas que executam no sistema. A especialização pode ainda ser utilizada na diferenciação dos vários tipos de *holons* [VBWV$^+$98].

Os conceitos que *Koestler* desenvolveu para organizações sociais e de organismos vivos foram traduzidos num leque de conceitos apropriados para as indústrias de manufatura. O principal objetivo passou por introduzir na manufatura benefícios como, a estabilidade em situações de distúrbios, a adaptabilidade e a flexibilidade em circunstâncias de mudança e o uso eficiente dos recursos disponíveis [VBWV$^+$98].

Uma das grandes vantagens dos HMS é que em condições de funcionamento normal, sem mudanças ou situações imprevistas, estes sistemas preservam a estabilidade da hierarquia (estrutura centralizada). No caso de perturbações e situações de falha, os *holons* são flexíveis e dinâmicos ao ponto de conseguirem adaptar-se de forma distribuída, autónoma e cooperativa às variações do seu meio ambiente.

A implementação dos conceitos holónicos de manufatura pode ser realizada com recurso à tecnologia de agentes. A utilização desta tecnologia na implementação de HMS garante a estes sistemas um alto nível de abstração e fornece diversas formas que facilitam a emergência das propriedades desejadas e o alcance dos objetivos pretendidos.

Sistemas Biónicos de Manufatura

Na natureza existem variados sistemas biológicos cujos organismos tendem a competir e a adaptar constantemente as suas caraterísticas e comportamentos, para conseguir sobreviver nas condições ambientes em que vivem [FRA+14].

A maioria dos sistemas biológicos são fundamentalmente caraterizados pela sua complexidade. Esta complexidade advém da composição destes sistemas em inúmeras partes que interagem entre si, mediante comportamentos autónomos e espontâneos que visam o alcance de um resultado global [FRA+14].

Neste contexto, é possível criar algum paralelismo entre os conceitos pertencentes aos sistemas biológicos e aos sistemas de manufatura. Como resultado deste paralelismo, surgem os Sistemas Biónicos de Manufatura (BMS) que têm como principal objetivo lidar com os imprevistos que tendem a surgir nos ambientes de manufatura. Estes sistemas são inspirados em ideias biológicas como a auto-organização, a aprendizagem, a evolução e a adaptação [UHFV00].

Os organismos biológicos têm a capacidade de se auto-adaptarem às modificações ambientais e de se auto-sustentarem através de funcionalidades como o auto-reconhecimento, o auto-crescimento e a auto-recuperação [UHFV00].

Estas funcionalidades são expressas mediante dois tipos de informação biológica, a informação genética que evolui durante as gerações (*DNA-type*) e a informação que é adquirida individualmente durante o ciclo de vida de cada organismo (*BN-type*). A união destes dois tipos de informação biológica numa entidade individual é o que torna os sistemas vivos autónomos e adaptáveis [UHFV00].

Os BMS e os HMS apresentam vários aspetos em comum. Ambos consideram as suas entidades constituintes, células e *holons* respetivamente, como entidades de manufatura autónomas, cooperativas e inteligentes. No entanto, o agrupamento em BMS é limitado à divisão inicial de células para criar uma entidade de manufatura (forma de vida) através do processo de especificação da informação do *DNA-type* (desenho dos requisitos de vida), o que faz com que os BMS sejam teoricamente mais dinâmicos que os HMS [Tha96].

Os BMS apresentam a flexibilidade necessária para evoluir de várias formas consoante as necessidades do sistema. Além disso, utilizam operadores enzimáticos e funções intracelulares na modelação da divisão das suas células constituintes [Tha96].

No ambiente de manufatura os produtos são desenvolvidos a partir das matérias-primas e da expressão da sua própria informação *DNA-type*. Os equipamentos de manufatura trabalham os produtos mediante a informação *BN-type*. Daí em diante, o BMS tende a adaptar-se às mudanças imprevistas no ambiente através das suas capacidades de evolução e de aprendizagem, tal como acontece nos sistemas vivos [UHFV00].

Posto isto, todos os elementos que constituem os BMS devem ser vistos como organismos autónomos e auto-organizados [UHFV00]. Estas caraterísticas devem-se ao processo dinâmico e adaptável que fornece a estes sistemas a capacidade de reestruturação sem que para isso, tenha de existir algum tipo de ação externa [FRA$^+$14].

As semelhanças entre os comportamentos dos BMS e os sistemas biológicos são portanto evidentes. Ambos os sistemas lidam com a emergência de um objetivo global. Sendo

este, maior do que aquele que possivelmente iria ser atingido por entidades individuais, sem a existência de qualquer interação que promovesse a cooperação entre elas [FRA+14].

Com o desenvolvimento dos sistemas de manufatura e com a inevitável convergência destes com as arquiteturas modulares, a complexidade destes sistemas está a atingir níveis de inovação mais elevados. E desta forma, os comportamentos dos mesmos tendem a aproximar-se cada vez mais do que se passa na natureza [FRA+14].

Sistemas Reconfiguráveis de Manufatura

A reconfiguração é considerada uma caraterística necessária nos novos sistemas de manufatura. Esta caraterística permite lidar com as mudanças imprevistas nos mercados de produção que têm ocorrido a ritmo cresceste nos últimos anos [MUK00].

Para produzir novos produtos e alterar os produtos já existentes, é necessário adicionar novas funções ao sistema de manufatura através da sua reconfiguração [KHJ+99].

No entanto, este tipo de reconfiguração, através da adição de novas funções, implica longos períodos de reconfiguração durante o tempo de vida do sistema, o que impossibilita uma reconfiguração rápida e facilitada [KHJ+99].

Assim, sistemas com capacidade de resposta em que os seus padrões de produção são ajustáveis às flutuações na procura do produto e, cujas funcionalidades podem ser adaptadas à produção de novos produtos, são um bem necessário na manufatura atual [MUK00].

Face a estas necessidades e numa linha menos bio-inspirada mas sustentada pela tecnologia, cujos componentes são constituídos essencialmente por máquinas e controladores totalmente reconfiguráveis, surge o paradigma de Sistemas Reconfiguráveis de Manufatura (RMS).

Da definição destes sistemas vem que, "*Um Sistema Reconfigurável de Manufatura foi projetado desde o início, para uma mudança rápida na sua estrutura, bem como nos seus componentes de hardware e software, a fim de ajustar rapidamente a capacidade e funcionalidade da produção dentro de um conjunto de peças, em resposta às modificações repentinas no mercado e aos requisitos regulamentares*"[KHJ+99].

Situações como a alta produtividade e a habilidade de um sistema reagir de modo rápido e eficiente a uma mudança no ambiente de produção, são indispensáveis ao sucesso destes sistemas e também da manufatura [MUK00].

Hardware e software reconfiguráveis são necessários mas não condições suficientes para um verdadeiro RMS. O cerne deste paradigma baseia-se numa reconfiguração que pretende combinar o desenho do sistema com o desenho de uma arquitetura aberta baseada em controladores reconfiguráveis [KHJ+99].

Estes sistemas possuem as caraterísticas chave necessárias à ótima relação de qualidade custo e que fazem com que sejam considerados sistemas de alto nível de reconfiguração. Entre elas destacam-se: a modularização, a integrabilidade, a personalização, a convertibilidade e a diagnosticabilidade [ElM05].

A modularização e a integrabilidade dos RMS deriva do fato destes sistemas serem constituídos por módulos de software e hardware que podem ser facilmente integrados. O diagnóstico nestes sistemas é indispensável na deteção rápida e eficaz de partes defeituosas. Estas três caraterísticas visam a redução dos tempos de esforço e de reconfiguração do sistema [KHJ+99].

A personalização destes sistemas engloba dois aspetos importantes, a flexibilidade da personalização e o controlo personalizado. A flexibilidade da personalização é manifestada na construção de máquinas para as várias partes/peças do produto final, fornecendo a flexibilidade necessária a cada parte específica. O controlo personalizado é conseguido

através da integração dos módulos de controlo e tem como objetivo fornecer a cada módulo funções de controlo especializadas [KHJ+99].

A convertibilidade dos RMS implica a configuração e a definição do modo ótimo de operação do sistema para a produção de uma quantidade específica de produtos. Esta produção deve ser completada num determinado intervalo de tempo com curtos períodos de conversão. A conversão requer por exemplo, a troca de ferramentas e de dispositivos no sistema físico [KHJ+99].

A personalização e a convertibilidade tendem a reduzir acentuadamente os custos de produção [KHJ+99].

Mediante estas caraterísticas, os RMS permitem a reconfiguração do sistema de manufatura na sua totalidade, do hardware da máquina e do software de controlo [ElM05].

Sistemas Evolutivos de Produção

A capacidade que um sistema tem em evoluir não reside apenas na agilidade que os seus componentes revelam no processo de adaptação às mudanças nas condições de operação. O sistema deve também conseguir auxiliar a evolução dos seus componentes, tal que os seus processos se tornem auto-evolutivos, auto-reconfiguráveis, auto-ajustáveis, auto-diagnosticáveis, etc... [OB09]

Este paradigma foca-se no desenho, na manutenção e na evolução dos sistemas industriais e representa uma aproximação holística baseada no suporte da co-evolução do produto/*shop-floor* [RBCO10].

O conceito dos EPS abrange o desenho, a arquitetura e as considerações técnicas exploradas no âmbito dos Sistemas Evolutivos de Manufatura (EAS) [RBCO10]. Sendo importante destacar o contributo valioso que estes sistemas tiveram no sucesso de projetos Europeus como o EUPASS, o A3 e o IDEAS [OLBH12].

As arquiteturas destes sistemas reúnem qualidades como a inteligência e a modularização e permitem o controlo distribuído dos seus módulos constituintes [RBCO10].

Através da tecnologia de agentes é possível realizar a agentificação das unidades do sistema a fim de fornecer aos componente físicos as caraterísticas pretendidas e que representam o conceito de agente [OLBH12].

A agentificação das unidades físicas do sistema resulta numa nova entidade constituinte do EPS, denominada de agente mecatrónico. O agente mecatrónico é um dispositivo físico de produção que incorpora a programação de um agente. Esta agentificação introduz nos EPS os conceitos de emergência e de auto-organização.

Estes sistemas preocupam-se sobretudo, com o que acontece quando o sistema de produção precisa de passar por uma mudança em alguma fase do seu estado físico, de controlo ou de produção [OB09].

Deste modo, surge a inspiração biológica destes sistemas, sendo que, é a mudança que impulsiona a sua capacidade de adaptação/evolução e não os cenários conhecidos que seguem um padrão definido e previsível, sem falhas ou alterações constantes [OB09].

Os EPS fornecem soluções de uma granularidade razoável devido ao fato de cada módulo e/ou unidade de produção possuir o seu próprio poder de processamento. Quando estes módulos se juntam para formar sistemas, células ou estações de trabalho, a emergência de novas habilidades é possibilitada [OB09].

A granularidade está associada ao nível de complexidade de cada componente do sistema de manufatura. Por exemplo, se uma linha de montagem é composta por vários módulos que são facilmente adicionados ou removidos, o sistema apresenta uma baixa granularidade. Se, por outro lado, os componentes a serem adicionados ou removidos forem pinças/garras, sensores ou cilindros pneumáticos, o sistema apresenta uma alta granulari-

dade [OB09].

Os paradigmas de HMS e RMS pretendem fundamentalmente atingir soluções de alto nível. O paradigma de EPS difere dos anteriores no sentido em que se preocupa com as mudanças previstas e imprevistas numa série limitada de produtos (género) [OB09].

Conceitos como a emergência e auto-organização são cada vez mais importantes e necessários a incorporar na próxima geração dos sistemas de produção de manufatura. No entanto, as verdadeiras implementações destes novos conceitos sobre os *shop-floors*, são ainda muito escassas [OLBH12].

2.2 Necessidade de Observação e Análise de MAS

No contexto de 2.1 é importante reter que os sistemas de produção de manufatura passaram a ser ágeis, flexíveis e adaptáveis. Estas caraterísticas possibilitam a fácil adição/remoção dos componentes distribuídos em tempo real, sem a necessidade de encerrar todo o sistema como acontecia nos sistemas tradicionais. [NROB14].

Os MAS apresentam caraterísticas desejadas nos ambientes de produção no sentido em que promovem a capacidade de resposta do sistema e ao mesmo tempo evitam as perturbações durante a produção. Ou seja, a composição distribuída destes sistemas garante a flexibilidade dos *shop-floors* industriais e ainda contribui à sua tolerância a falhas.

No entanto, num contexto mecatrónico, os módulos distribuídos destes sistemas são propícios a alterações constantes e, tendo em conta a sua composição com um elevado número de entidades que não são necessariamente homogéneas, tornam-se difíceis de modelar [RFMB14].

A alteração dos componentes do sistema em tempo real influencia de várias formas o comportamento global do sistema. Assim sendo, é crucial compreender de que forma as mudanças sofridas afetam a topologia e o desempenho do sistema [NROB14]. Pois, dependendo das interações e dos agrupamentos entre as entidades do sistema, o impacto destas mudanças no desempenho e na robustez do sistema apresenta variadas diferenças [NFO+13].

Devido aos comportamentos emergentes das várias entidades que compõem os MAS, surge a necessidade de estudar e testar os agentes de modo a assegurar que o sistema funciona apropriadamente e que o comportamento emergente que o mesmo apresenta é o esperado [TSS+06].

A compreensão das tomadas de decisão seguidas pelas entidades autónomas, que definem o modo como as redes dos agentes crescem e são estruturadas, e os tipos de relações geridos pela rede são aspetos importantes que permitem prever o funcionamento destes sistemas de modo a evitar possíveis falhas [MN05].

Como tal, para corresponder a estas exigências, a aplicação de métodos e o desenvolvimento de novas ferramentas que tenham a capacidade de analisar o comportamento de sistemas auto-organizados em cenários de constante evolução são portanto bens necessários para cumprir com os requisitos da produção evolutiva [NROB14].

Mediante modelos de referência e metodologias que permitam a análise, o estudo e a observação destes sistemas é possível desenhar novos sistemas que possuam a capacidade de reagir de modo ágil e eficaz às situações de perturbação no ambiente de produção. O que por conseguinte, tende a aumentar significativamente o desempenho e o cumprimento dos objetivos globais destes sistemas.

2.3 Ferramentas e Metodologias para Observação de MAS

2.3.1 Ferramenta *Java Sniffer*

Face à necessidade de ferramentas que permitam a visualização e a análise da comunicação nos MAS a *Rockwell Automation* desenvolveu a ferramenta *Java Sniffer*.

A ferramenta *Java Sniffer* permite a ligação remota aos sistemas JADE que se encontram em execução e recebe todas as mensagens trocadas entre os agentes do sistema. A informação capturada é apresentada ao utilizador em vários pontos de vista [BCG07].

Desta forma, e tal como representado na Figura 2.1, o ecrã de visualização da ferramenta divide-se em quatro secções principais e cada secção fornece ao observador informações com diferentes níveis de detalhe [TSS+06]. As quatro secções são:

1. *Message Detail View* : Fornece informações sobre o conteúdo das mensagens selecionadas;

2. *List of Messages* : Apresenta a lista de mensagens trocadas entre os agentes do sistema através de diagramas sequenciais na linguagem *Unified Modeling Language* (UML). Esta secção contém ainda uma lista dos agentes em que cada seta representa uma mensagem trocada entre dois agentes;

3. *List of Work Units* : Mostra a lista das unidades de trabalho requisitadas pelos agentes. Cada unidade de trabalho é responsável pela execução de uma parte específica do trabalho global a completar;

4. *Workflow View* : Apresenta a árvore dinâmica do fluxo de trabalho correspondente à unidade de trabalho selecionada na lista das unidades de trabalho. O fluxo de trabalho representa todas as mensagens que pertencem à execução de uma parte do trabalho em particular. As mensagens podem ser de planeamento, de objetivo de trabalho ou de execução de trabalho.

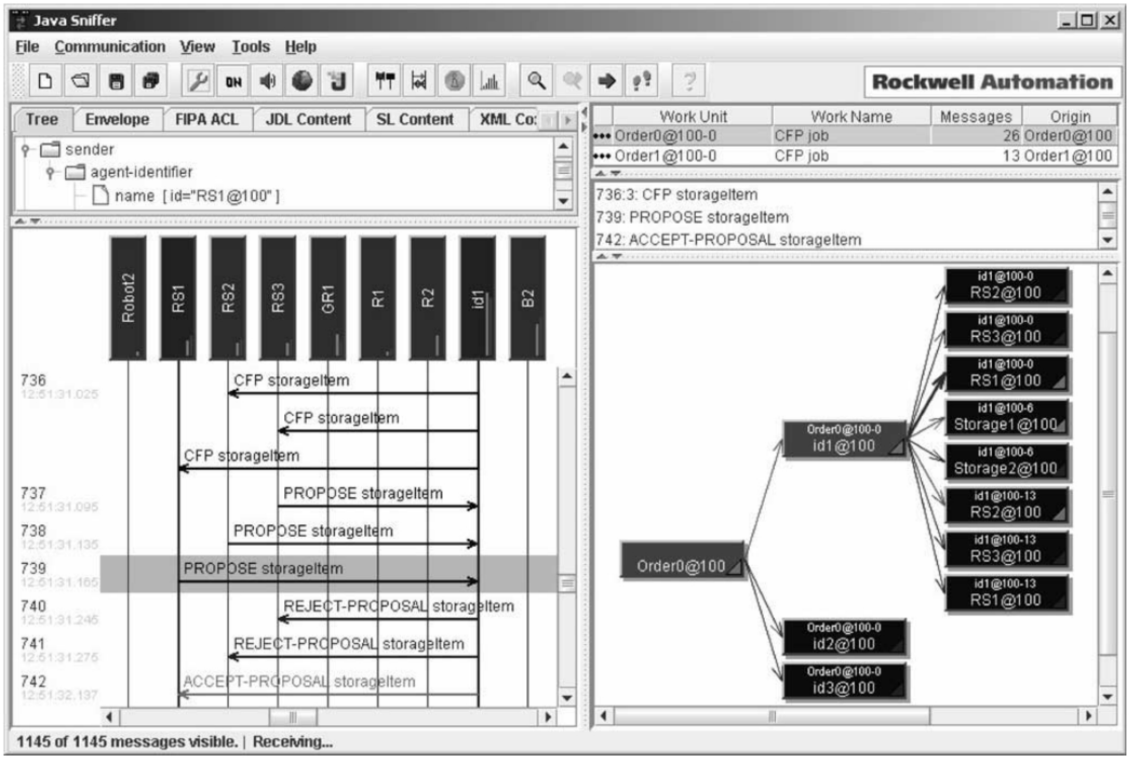

Figura 2.1: Janela Principal da Ferramenta *Java Sniffer*, retirada de [TSS+06].

A ferramenta em descrição é capaz de representar as mensagens em diagramas de sequência UML de baixo nível e de fornecer uma visão de alto nível, através dos diagramas de fluxo de trabalho rastreados e criados dinamicamente [BCG07].

São suportadas codificações de mensagem como *eXtensible Markup Language* (XML), Lisp e BitEfficient, de acordo com as especificações da linguagem de comunicação de agentes proposta pela *Foundation for Intelligent Physical Agents* (FIPA) (FIPA-ACL) e ainda, conteúdos de linguagem como *Semantic Language* (SL), XML e JDL (Job Description Language) [BCG07].

O *Java Sniffer* oferece visualização de informações estatísticas, filtragem de mensagens e de agentes e a criação automática de ficheiros de arquivo [BCG07].

Para desenvolver a ferramenta *Java Sniffer*, os membros da *Rockwell Automation, Inc.* basearam-se no conceito de agrupamento.

O agrupamento, termo que na língua inglesa é denotado por *clustering*, tem sido estudado em várias áreas e todas elas partilham o mesmo problema. Dada uma série de entidades, criar um conjunto, tal que, as entidades do mesmo agrupamento têm mais semelhanças umas com as outras do que com outras entidades que se encontram num agrupamento diferente [OOVSB03].

Agrupamento de Agentes utilizado na Aplicação *Java Sniffer*

A metodologia para visualização dos comportamentos associados aos agrupamentos e às relações entre os pares de agentes nos MAS, que cimentou a implementação da ferramenta *Java Sniffer*, teve como principal objetivo desmistificar a seguinte problemática: como se podem posicionar agentes que pertencem a determinado agrupamento num dado número de unidades computacionais tal que a comunicação entre estas unidades seja minimizada? [STŠM07]

Esta metodologia baseia-se num modelo dinâmico para agrupamentos de agentes, seguido de uma análise estática aos mesmos.

No modelo dinâmico os autores consideram a coleção de agentes como um sistema de massas que exercem forças entre elas. No seguimento desta ação, observam as mudanças de posição e de velocidade em cada agente [STŠM07].

A análise estática do agrupamento pretende suportar o modelo dinâmico utilizado e compõe-se em dois agrupamentos distintos: agrupamento topológico de agentes e agrupamento estático de agentes a partir das mensagens trocadas [STŠM07].

O agrupamento topológico de agentes utiliza o algoritmo *Gustafson-Kessel* para identificar e inferir o agrupamento de agentes através das posições dos agentes num determinado intervalo de tempo. Neste agrupamento, cada agente é associado a um agrupamento de agentes [STŠM07].

O agrupamento estático de agentes a partir das mensagens trocadas considera a formação de agrupamentos à medida que as mensagens vão sendo trocadas entre os agentes. Para isso, é utilizado um critério que pretende minimizar a comunicação entre agrupamentos e maximizar a comunicação dentro do mesmo agrupamento [STŠM07].

Nesta primeira abordagem, o agrupamento dinâmico demonstrou ser ágil na estruturação funcional do sistema a partir da troca de mensagens num MAS. No entanto, alcançar um padrão nas posições dos agentes não foi assim tão simples [STŠM07].

Assim, em [KTŠS08] a proposta inicial é melhorada. Nesta nova abordagem os autores trabalharam a análise estática já referida e utilizaram na mesma um algoritmo evolutivo denominado de *Multi-objective Prototype Optimization with Evolved iMprovement Steps* (mPOEMS), que permitiu transformar o problema numa otimização multiobjetiva.

Mediante este algoritmo é possível utilizar múltiplos objetivos e várias medidas de qualidade na procura do agrupamento ideal. A otimização através de múltiplos objetivos é considerada adequada à problemática do agrupamento. Pois, além de minimizar a comunicação entre diferentes agrupamentos, ainda torna possível definir outros aspetos que desqualificam soluções não ótimas [KTŠS08].

O algoritmo mPOEMS retém as melhores soluções encontradas, designadas por soluções base. Em cada iteração deste algoritmo, uma solução é escolhida de entre as várias soluções base. Posteriormente, uma série de ações sequenciais gera novas soluções que são associadas à solução base, o que resulta numa nova solução base [KTŠS08].

Para a implementação de mPOEMS, os autores geraram soluções a partir da solução base com o intuito de maximizar as comunicações internas ao agrupamento. Primeiro, os k agentes que representam as sementes de cada agrupamento são encontrados, sendo que a comunicação entre estes é mínima. Posteriormente, os agentes que permanecem são selecionados um por um e, para cada agente, a frequência da comunicação para todos os

agrupamentos é calculada. Por fim, o agente é associado ao agrupamento com que comunicou mais [KTŠS08].

No contexto da análise estática de agrupamentos de agentes em MAS, são muitas as vantagens inerentes aos algoritmos de otimização multiobjetiva. Os objetivos são otimizados em simultâneo e, ao contrário do que acontece nos métodos tradicionais, que retornam apenas uma solução por iteração, o algoritmo mPOEMS retorna um conjunto de soluções ótimas, de onde se podem escolher as melhores [KTŠS08].

O modelo dinâmico para o agrupamento de agentes, aplicado a um sistema real, permitiu aos autores descobrir dependências entre agentes, observar o comportamento do sistema em tempo real e ainda, através das técnicas de aprendizagem automática (do inglês, *Machine Learning*), identificar automaticamente agrupamentos a partir das posições dos agentes com resultados razoáveis. [KTSS10].

Agrupamento de Agentes em MAS muito Extensos

Os autores de [OOVSB03] abordaram o agrupamento numa perspetiva diferente. Os autores basearam-se num problema de pesquisa, no qual cada agente individual tem o objetivo de encontrar outros agentes que se assemelhem de alguma forma a ele próprio.

Com esse propósito, os autores criaram um modelo abstrato de agentes com um número de ações limitado e que agem recorrendo ao uso de funções simples de decisão. Neste modelo os agentes são caraterizados segundo os seus atributos e cada um deles tem um pequeno número de ligações dentro de uma rede aleatória. Estas ligações representam canais de comunicação e permitem a definição da vizinhança de cada agente [OOVSB03].

O objetivo do sistema prende-se na restruturação das ligação dos agentes e na seleção de algumas delas a fim de formar linhas de comunicação entre os mesmos. Posteriormente, é gerado um grafo de ligações que representa o agrupamento de agentes [OOVSB03].

Nas simulações efetuadas pelos autores, cada agente inicia o agrupamento a partir de uma única entidade com ligações para outros agentes escolhidos ao acaso. À medida que a simulação decorre, os agentes escolhem algumas das suas ligações com o intuito de formar correspondências ou ligações correspondentes, com base nas caraterísticas semelhantes dos agentes que se encontram unidos por essas ligações [OOVSB03].

Por conseguinte, os agrupamentos escolhem as melhores correspondências propostas pelos seus agentes, tal que essas correspondências passam a ser ligações. Os agentes unidos pelo caminho formado por essas ligações formam por sua vez, um único agrupamento [OOVSB03].

Internamente, cada agente possui um atributo principal que descreve as suas caraterísticas básicas e um número limitado de objetivos com base no seu atributo principal. Assim, mediante esta associação entre os atributos e os objetivos dos agentes torna-se possível agrupar agentes consoante os seus atributos [OOVSB03].

Sendo ambos representados por pontos num espaço bidimensional, os objetivos são escolhidos em função dos atributos de cada agente. Desta forma, os autores conseguiram estruturar uma nuvem de pontos à volta do atributo central do agente formada por objetivos [OOVSB03].

Os autores focaram-se na natureza do comportamento coletivo dos agrupamentos. Por esta razão, os agentes foram limitados a processos de decisão simples, o que permitiu realçar o comportamento básico do agrupamento e fornecer um bom fundamento para estudar agentes mais complexos utilizados em aplicações reais [OOVSB03].

Os agrupamentos de agentes conseguem ajustar o seu tamanho ao tamanho do agrupamento do conjunto e conseguem aprender as melhores correspondências. Isto demonstra como é que a utilização de agentes autónomos e racionais, que mediante as suas capacidades de alterar os seus critérios de decisão ao longo do tempo, pode ser vantajosa na

problemática do agrupamento [OOVSB03].

Do estudo apresentado em [OOVSB03], os autores concluíram que os sistemas descentralizados podem ser utilizados na procura de agrupamentos em sistemas de grandes dimensões em períodos de tempo razoáveis e com uma boa qualidade. O comportamento dos agentes estudados permitiu a análise das dinâmicas associadas à formação de agrupamentos com base nas semelhanças entre os atributos dos agentes.

2.3.2 Ferramentas de *Debbuging* de Agentes no *JADE*

O JADE oferece diversas ferramentas para a gestão e para o *debugging* de agentes ativos na plataforma. Entre estas, é importante destacar: *The Sniffer Agent*, *The DummyAgent*, *The Instrospector Agent*, *The Log Manager Agent* e o serviço de notificação de eventos.

The Sniffer Agent

A ferramenta *The Sniffer Agent* é uma ferramenta nativa do JADE extensivamente usada para *debugging*, ou para simplesmente documentar as conversas entre os agentes do sistema [BCG07].

O agente *Sniffer* subscreve o Agente de Gestão do Sistema (AMS) da plataforma para ser notificado de todos os eventos que ocorrem na mesma e de todas as mensagens trocadas entre um conjunto de agentes específico. O AMS é uma entidade nativa do JADE responsável pela supervisão da plataforma e pela gestão das suas operações [BCG07].

Quando o utilizador do JADE decide efetuar o *"sniff"* de determinado agente ou de um grupo de agentes, todas as mensagens dirigidas a esse(s), ou vindas desse(s) agente(s), são rastreadas e exibidas na Interface Gráfica do Utilizador (GUI) do agente *Sniffer*. A GUI do agente *Sniffer* permite que o utilizador interaja especificamente com este agente mediante a sua interface gráfica, onde o agente reage às ações do utilizador [BCG07].

The DummyAgent

A ferramenta *The DummyAgent* é uma ferramenta de fácil utilização e tipicamente utilizada no desenvolvimento de aplicações que suportam a tecnologia de agentes. Esta ferramenta permite testar os comportamentos de determinado agente que esteja ativo na plataforma [BCG07].

Mediante a sua capacidade de enviar e receber mensagens no formato Linguagem de Comunicação de Agentes (ACL) e destinadas especificamente ao utilizador, o *DummyAgent* pode ser utilizado para enviar estímulos (mensagens) a determinado agente. O envio destes estímulos permite analisar a reação do agente face à receção dos mesmos [BCG07].

The Instrospector Agent

A ferramenta *The Instrospector Agent* é utilizada para o *debug* do comportamento de determinado agente [BCG07].

O *IntrospectorAgent* garante ao agente selecionado um ciclo de vida e consegue monitorizar e controlar todas as mensagens que este envie ou receba. Esta ferramenta fornece a lista dos comportamentos agendados para o agente. Através da lista de comportamentos, o utilizador pode monitorizá-los ou até executá-los passo a passo, o que permite a introspeção da execução do agente selecionado [BCG07].

The Log Manager Agent

A ferramenta *The Log Manager Agent* pretende essencialmente simplificar a dinâmica e o controlo distribuído do serviço de *logging* do JADE. O serviço de *logging* permite a visualização das mensagens relativamente às funções e aos métodos e/ou interfaces externas que emitem mensagens de *log.* [BCG07].

Para visualizar as mensagens de *log*, a presente ferramenta fornece uma GUI onde apresenta as mensagens de *log* de cada componente que vai sofrendo alterações durante as suas execuções. A GUI deste agente inclui todos os componentes que estão a operar em nós remotos e as mensagens de *log* específicas da aplicação [BCG07].

Serviço de Notificação de Eventos

O Serviço de Notificação de Eventos (ENS) trata-se de um serviço que gere a distribuição das notificações relativamente a todos os eventos gerados por cada nó (agente) na plataforma [BCG07].

Cada vez que surge um evento na plataforma (agente nasceu, mensagem enviada, etc...), o evento é intercetado pelo ENS e encaminhado para todos os agentes que solicitaram a receção de notificações sobre determinado tipo de evento. Os agentes interagem com o ENS através da troca de mensagens ACL com o AMS [BCG07].

Vantagens e Limitações das Ferramentas do JADE

Num contexto mecatrónico, todas as ferramentas *supra* apresentadas contribuem significativamente para a observação e análise dos comportamentos dos agentes mecatrónicos durante as suas execuções em tempo real no *shop-floor* industrial.

Por exemplo, o *Sniffer* permite a monitorização de todas as comunicações efetuadas entre os agentes da plataforma mecatrónica. O *DummyAgent* possibilita a análise da reação de um agente quando submetido a determinada alteração no seu comportamento. O *IntrospectorAgent* permite controlar e monitorizar todos os passos de execução associados ao cumprimento das tarefas dos agentes. O *The Log Manager Agent* pode ser essencial na monitorização das mudanças que ocorrem no ambiente mecatrónico a fim de detetar possíveis pontos de falha. Por fim, o serviço de notificação de eventos pode representar um papel relevante no controlo dos eventos que surgem no ambiente mecatrónico e, por conseguinte, restringir as tomadas de decisão dos agentes consoante os eventos que ocorrem de forma a agilizar os processos em execução na plataforma.

Ainda assim, estas ferramentas poderão apresentar algumas limitações. Devido à distribuição dos agentes num MAS em vários componentes, estes sistemas executam inúmeros processos em paralelo. O que dificulta bastante o *debugging* do sistema.

Em sistemas muito complexos onde existem muitas interações, muitas tarefas a executar, várias adições/remoções de componentes em tempo real e constantes eventos no decorrer do sistema, a utilização destas ferramentas pode ser difícil e provocar um impacto negativo no desempenho da plataforma JADE, o que por conseguinte, irá impossibilitar uma análise válida ao MAS em observação.

2.3.3 Ferramenta de Visualização para Sistemas Mecatrónicos Multiagentes

Com o intuito de desenvolver uma nova metodologia que permita a visualização e a interpretação da informação trocada entre os agentes em sistemas distribuídos mecatrónicos, os autores de [FRN+12] apresentam uma nova ferramenta de visualização.

A ferramenta *Visualization Tool* foi desenvolvida utilizando o JADE e é de salientar o seu contributo no projeto *IDEAS Agent Development Environment* (IADE).

Baseada em técnicas passivas de captura de mensagens e tirando alguma vantagem do agente *Sniffer* do JADE, o agente *Visualization Tool Agent* (VTA) subscreve ao AMS, solicitando a ativação do ENS para começar a intercetar as mensagens trocadas no sistema IADE [FRN+12].

Desta forma, é possível obter todas as informações importantes e necessárias com um impacto mínimo no desempenho da plataforma mecatrónica [FRN+12].

Posto que o principal objetivo da VTA é fornecer ao seu utilizador uma forma simplificada de visualizar toda a informação importante extraída da plataforma foram desenvolvidos cinco modos de visualização diferentes e várias funcionalidades associadas a cada um deles [FRN+12].

Os modos de visualização disponíveis são os seguintes:

1. *The Hardware View* : Apresenta todo o equipamento físico que compõe o sistema. Este modo de visualização permite configurar o hardware de acordo com o plano do *shoop-floor* e a sua opção de navegação permite a observação de todos os componentes da mesma hierarquia ao mesmo tempo;

2. *The Skill View* : Efetua a tabulação de todas as habilidades (*skills*) disponíveis na plataforma mecatrónica. No caso de uma habilidade composta (*composite skill*) esta pode ser decomposta e apresentada ao utilizador de acordo com a sua composição. Neste modo de visualização é possível verificar qual o agente mecatrónico que abstrai determinada *skill*, os parâmetros da *skill* e o tipo da *skill*. Também é possível seguir a execução da *skill*;

3. *The Platform View* : Pretende fornecer uma perspetiva do sistema que permita a visualização em tempo real da dinâmica das relações entre os agentes mecatrónicos e das relações entre a plataforma mecatrónica e o sistema físico;

4. *The Performance View* : Permite a visualização de alguns indicadores de desempenho relativamente ao componente selecionado. O desempenho pode ser visto na perspetiva de hardware ou na perspetiva de *skill*;

5. *The Sniffer View* : Permite a visualização em tempo real de todas as mensagens trocadas e alguns dos seus parâmetros. Tais como, a hora do envio da mensagem, o recetor e o emissor da mensagem, o tipo da mensagem, o propósito da mensagem e o conteúdo da mensagem. É ainda possível verificar e selecionar os agentes que se encontram sob observação.

Além do VTA, um agente de visualização para dispositivos móveis, ou *Visualization Agent for Mobile Devices* (VAMD), foi também projetado de forma a facilitar o acesso remoto à informação desejada [FRN+12].

Com a ferramenta VAMD, o utilizador pode seguir à distância a plataforma e as execuções individuais através de pedidos de informação relativamente aos componentes de hardware ou às habilidades disponíveis na plataforma. Isto pode ser de grande importância, por exemplo, para fins de manutenção [FRN+12].

O protótipo industrial desenvolvido no projeto IDEAS realçou a necessidade de uma nova abordagem para visualizar a informação do sistema e para recolher informação de sistemas distribuídos semelhantes [FRN+12].

Desta forma, a *Visualization Tool* representa uma alternativa viável para recolher informação de uma plataforma distribuída com um impacto mínimo na implementação dos agentes mecatrónicos. Além disso, fornece ainda uma perspetiva diferente relativamente às relações entre o hardware, os componentes lógicos e as habilidades disponíveis no sistema mecatrónico multiagente [FRN+12].

2.4 Conclusões Gerais

Perante o exposto no presente capítulo, é notável a evolução dos sistemas de manufatura nas últimas décadas. Vários paradigmas surgiram na área, na esperança de tornar a manufatura sustentável e apta a fornecer sistemas de produção cada vez mais ágeis e flexíveis.

Os paradigmas de HMS, BMS, RMS e EPS fizeram parte de uma grande variedade de investigações e hoje, suportam muitas das exigências nos domínios da manufatura. Estes paradigmas permitem alcançar soluções válidas na resolução de situações que perturbam os ambientes de produção e que limitam o progresso da manufatura industrial.

As arquiteturas abertas e os novos paradigmas promovem a reconfiguração e a reutilização lógica dos equipamentos, o que contribui para a redução de massa dos mesmos e de outras fontes de desperdício de energia. [RB11].

No entanto, a maioria das aproximações emergentes abrangem comportamentos complexos, cenários inesperados e interações constantes entre as entidades físicas e lógicas dos sistemas, o que muitas vezes não representa um aspeto favorável ao estudo e à validação destes sistemas.

Apesar de ainda muito escassas, têm surgido novas metodologias que visam melhorar o desempenho dos atuais sistemas de produção através da observação dos comportamentos e reações destes sistemas face às alterações dos vários componentes do *shop-floor* industrial. As comunidades de investigação das áreas da manufatura têm apostado no desenvolvimento de metodologias baseadas no estudo do agrupamento de agentes em MAS.

À medida que os eventos ocorrem no sistema de controlo, os valores das ligações entre os agentes vão variando. A informação relativamente a estas ligações ou aos agrupamentos de agentes pode ser utilizada de várias formas. Por exemplo, a estrutura dos agrupamentos de agentes pode ser mapeada no hardware de execução do agente ou o posicionamento de agentes pertencentes a um agrupamento numa unidade computacional pode otimizar o desempenho do sistema [STŠM07].

Nestes termos, o estudo aprofundando dos agrupamentos de agentes poderá fornecer algum conhecimento extra relativamente às dinâmicas de interação num MAS. Este conhecimento pode ser utilizado na validação do modo de funcionamento destes sistemas o que, por sua vez, permitirá criar comunidades de agentes auto-organizados com elevados níveis de complexidade sem diminuir o desempenho do sistema [OOVSB03].

As ferramentas para observação e análise dos sistemas de manufatura têm contribuído para a otimização dos atuais sistemas de produção. No entanto, a eficácia destas ferramen-

tas em sistemas de elevado dinamismo, cujas entidades se mantêm em constante interação para garantir a flexibilidade do sistema nas várias situações de perturbação, não é ainda garantida.

Em conclusão, a introdução de metodologias e ferramentas desta natureza é considerada altamente benéfica à projeção de sistemas flexíveis e adaptáveis. Posto que, é esta flexibilidade que os torna tolerantes às várias perturbações nos ambientes de produção e, como tal, aptos a adaptarem as suas condições de trabalho de forma a cumprirem com os seus objetivos globais sem comprometer o desempenho do sistema.

Capítulo 3

Arquitetura

A presente arquitetura suporta a inferência e o pré-processamento de redes de agentes num contexto mecatrónico.

Deste modo, a arquitetura visa a análise das interações entre os agentes constituintes de um MAS e, por sua vez, efetua a estruturação da rede resultante dessas interações no formato GML.

3.1 Visão Geral da Arquitetura

A arquitetura da ferramenta é composta por cinco módulos e cada um desempenha um papel específico em cada uma das fases do processo de inferência da rede de um MAS.

Para efetuar o controlo dos módulos que compõem a arquitetura, o agente desenvolvido (*Sniffer*) incorpora vários algoritmos que implementam os comportamentos que definem as funcionalidades da ferramenta.

O módulo de memória permite que o *Sniffer* consiga capturar as interações entre os agentes do MAS em execução. Os módulos de filtragem e de agrupamento efetuam o processamento dos dados capturados, de modo a representar devidamente a rede gerada, consoante as interações dos seus agentes participantes. Posteriormente à captura e ao processamento, o *Sniffer* traduz os dados filtrados e agrupados num formato de ficheiro específico apropriado à representação de redes através de grafos.

O processamento dos dados recolhidos (filtragem e agrupamento) em tempo real é um processo que consome muitos recursos ao sistema, o que por conseguinte, influencia negativamente o desempenho do MAS e da ferramenta desenvolvida. Por este motivo, a captura das interações é feita durante o tempo de execução do MAS e a filtragem e o agrupamento dos dados são feitos apenas quando o sistema termina a sua execução.

O *Sniffer* tem ainda a capacidade de ser informado relativamente à morte e ao aparecimento de novos agentes na plataforma mecatrónica. O agente reage à receção dessas informações consoante o resultado das mesmas.

Os agentes que surgem na plataforma mecatrónica são automaticamente colocados num estado de observação (ação de *Sniff*). Os agentes que desaparecem da plataforma são automaticamente removidos desse estado (ação de *Unsniff*).

As habilidades do *Sniffer* e os algoritmos que compõem cada um dos seus módulos permitiram a implementação de uma ferramenta que visa a observação e a análise de qualquer MAS, independentemente do seu grau de dinamismo ou de complexidade.

A Figura 3.1 pretende esquematizar a arquitetura da ferramenta consoante os módulos que a compõem.

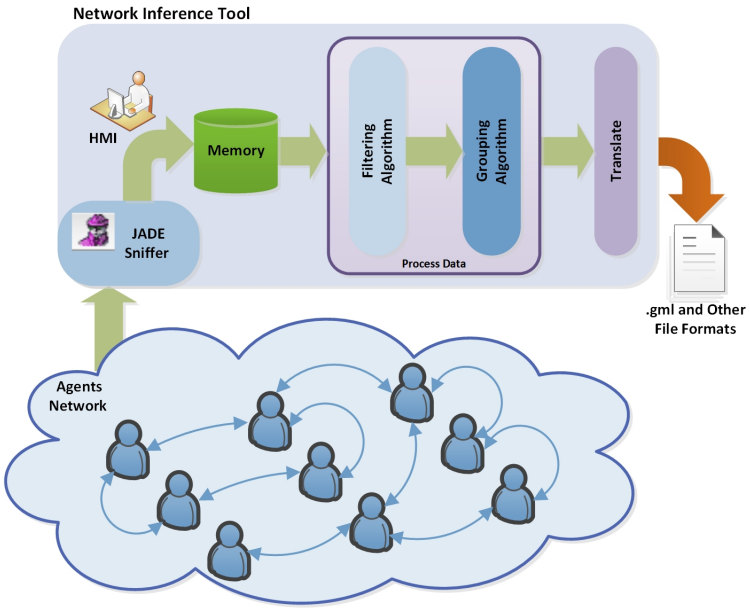

Figura 3.1: Arquitetura da Ferramenta de Inferência da Rede, adaptada de [RFMB14].

3.2 Noções de *Sniff* e *Unsniff* de um Agente

Dependendo da plataforma, para ativar (*Sniff*) ou desativar (*Unsniff*) a observação de determinado agente poderá ser necessário o estabelecimento de uma comunicação com algum agente especializado, nativo da plataforma de suporte ao MAS, que permita estas ações. Pois, ambas as ações são consideradas ações de gestão ou controlo da plataforma.

Os agentes especializados possuem a capacidade de notificar os agentes interessados sobre os mais variados tipos de eventos que caraterizam as mudanças às quais a plataforma multiagente está sujeita. De entre estes eventos importa referir a adição e a remoção de *containers*, o nascimento e a morte de um agente, a suspensão e o reiniciar da execução de um agente, o movimento e a clonagem de um agente, etc...

No entanto, para conseguir efetuar o *Sniff* ou o *Unsniff* de determinado agente, o *Sniffer* apenas precisa de ser notificado relativamente ao nascimento e à morte de agentes.

Com este intuito, o *Sniffer* envia uma mensagem ao agente especializado pedindo para começar a ser informado relativamente aos eventos mencionados. Desta forma, sempre que recebe uma notificação correspondente a estes eventos age de acordo com ela.

Ou seja, quando um novo agente surge na plataforma, o *Sniffer* solicita ao agente especializado a ativação do *Sniff* sobre o novo agente. Quando um agente desaparece da plataforma, o *Sniffer* solicita ao agente especializado o *Unsniff* desse agente. Ambas as solicitações são efetuadas através do envio de uma mensagem ao agente especializado.

A seguinte Figura 3.2, esquematiza o comportamento associado à receção das notificações e à reação do *Sniffer* face às mesmas.

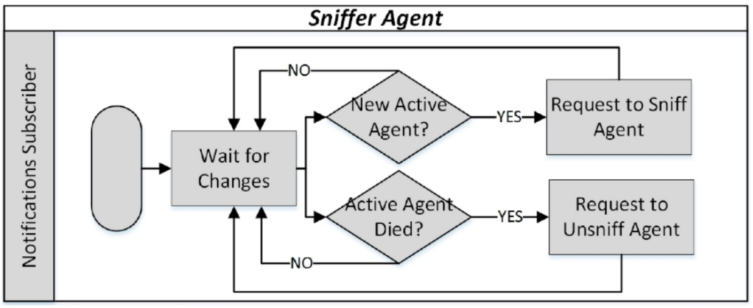

Figura 3.2: Comportamento do Mecanismo de Notificações.

No caso de *Sniff*, assim que o agente especializado responde à mensagem, o novo agente começa a ser observado pelo *Sniffer*.

Esta observação implica que todas as interações que o novo agente tenha com outros agentes, através da troca de mensagens, sejam acompanhadas e todas as mensagens que o agente envie ou receba sejam replicadas com destino ao *Sniffer*.

No caso de *Unsniff*, assim que o agente especializado responde à mensagem, todas as mensagens associadas ao agente que abandonou a plataforma deixarão de ser replicadas para o *Sniffer*.

3.3 Descrição dos Módulos da Ferramenta Desenvolvida

3.3.1 Módulo de Memória

O módulo de memória tem como finalidade guardar todas as mensagens que surgem na plataforma multiagente numa lista de mensagens.

Tal como se pode concluir perante a análise da seguinte Figura 3.3, o comportamento do módulo em apresentação, assim que ativado, permanece num estado de "escuta"permanente à espera de novas mensagens trocadas entre os agentes que se encontrem sob ação de *Sniff*. Este comportamento só é desativado assim que o utilizador da ferramenta termine a captura das mensagens.

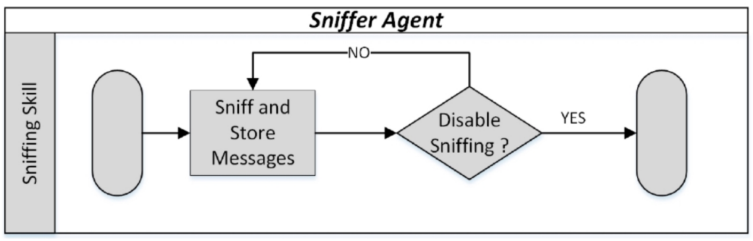

Figura 3.3: Captura de Mensagens num Sistema Multiagente.

Assim, à medida que as mensagens vão surgindo consoante as interações entre os agentes que se encontram a ser observados, o *Sniffer*, mediante o seu processo de captura de mensagens, efetua o armazenamento de cada uma delas. Neste sentido, o pós processamento de toda a informação recolhida é possível e facilitado.

3.3.2 Algoritmo de Filtragem

Tal como já abordado em 3.1, a ferramenta implementada é sensível às mudanças na rede dos agentes e, devido a este fato, o algoritmo de filtragem apenas é executado após o momento em que termina o processo de captura das mensagens.

Num ambiente com muitas interações, onde a carga de mensagens é excessiva, o processamento da informação recolhida em tempo real demonstrou, durante o desenvolvimento da ferramenta, estar diretamente relacionado com a perda de mensagens no sistema.

Assim, quando a captura da informação termina, momento em que o sistema mecatrónico multiagente deverá também ter terminado a sua execução, o algoritmo de filtragem inicia o seu processamento sobre a informação angariada.

O algoritmo de filtragem foi desenhado com o objetivo de efetuar uma seleção sobre todas as mensagens capturadas.

Enquanto decorrem as interações no MAS, é possível que existam muitas mensagens a serem trocadas com os agentes nativos da plataforma. No entanto, estas mensagens não devem estar incluídas nos dados que irão permitir a inferência da rede do sistema observado.

Pois, se estas mensagens forem incluídas na inferência da rede dos agentes, a integridade dos resultados obtidos poderá ficar comprometida, no sentido em que a inferência da rede apenas se destina a estudar o sistema em observação. A inclusão destas mensagens na rede dos agentes impede a observação e o estudo das interações exclusivas ao MAS.

As mensagens que envolvem as entidades nativas da plataforma multiagente destinam-se unicamente à gestão das ações e dos eventos que nela ocorrem. Devido a este fato, não têm influência sobre a cooperação entre as entidades do MAS em execução e, como tal, não deverão constar na rede gerada.

O funcionamento deste algoritmo é implementado mediante um ciclo principal que tem como finalidade iterar todas as entradas da lista de mensagens.

O conteúdo de cada mensagem vai sendo analisado de modo a averiguar as entidades que correspondem ao recetor e ao emissor da mensagem. Caso estas entidades não correspondam a nenhum dos agentes nativos da plataforma, a mensagem é guardada numa nova estrutura de dados, utilizada para guardar todas as mensagens que resultaram do processo de seleção. Cada entrada desta estrutura é composta por uma chave e por um valor associado.

A estrutura tem como chave o par de agentes que participaram naquela interação. Ou seja, o emissor e o recetor da mensagem. O valor associado a essa chave é representado por uma estrutura adicional que contém informação relativamente ao emissor, ao recetor e ao peso da mensagem trocada. Por definição, cada mensagem trocada entre dois agentes tem um peso unitário.

As ligações paralelas existentes entre os agentes, originadas pelo envio repetido de mensagens no mesmo sentido, não são ainda suportadas pela ferramenta. Assim, apenas a primeira mensagem trocada entre um par de agentes é depositada na estrutura *supra* citada.

A partir do momento em que um par de agentes interaja a primeira vez, a mensagem proveniente dessa interação é armazenada na estrutura. Daí em diante, as seguintes mensagens trocadas entre o mesmo par de agentes são tratadas como mensagens de reforço.

Para isso, antes de criar uma nova entrada na estrutura das mensagens selecionadas, é efetuada sobre a mesma uma pesquisa da chave representada pelo par de agentes e, caso a chave já exista, o que indica que os agentes em questão já interagiram pelo menos uma vez, o peso associado à interação desse par de agentes é sujeito a um incremento de uma unidade. Caso contrário, é criada uma nova entrada na estrutura.

3.3.3 Algoritmo de Agrupamento

A execução do algoritmo de agrupamento ocorre durante a execução do algoritmo de filtragem.

Além da seleção de mensagens e da passagem das mesmas para uma nova estrutura de dados durante o processo de filtragem, à medida que cada mensagem vai sendo analisada, uma outra estrutura de dados auxiliar é utilizada para definir os nós do grafo que irão representar toda a rede do MAS.

Na filtragem, o emissor e o recetor das mensagens trocadas vão sendo extraídos. Como cada recetor ou cada emissor corresponde a um agente do sistema, esta estrutura de dados auxiliar tem como finalidade guardar todos os agentes que integram o sistema. Isto é, guardar todos os nós da rede.

A interação que corresponde à troca de uma mensagem entre um emissor e um recetor corresponde a uma ligação entre dois agentes.

Cada uma destas ligações é guardada no valor associado à chave que corresponde a cada entrada da estrutura utilizada na filtragem das mensagens. Por conseguinte, estas ligações irão representar as arestas do grafo que representa a rede de agentes.

À medida que as mensagens vão sendo filtradas este algoritmo tem então como finalidade, definir os vários nós da rede (agentes) e as arestas que interligam esses nós (ligações entre agentes).

3.3.4 Módulo de Tradução

Para inferir devidamente a rede de agentes é necessário traduzir os dados processados pelos algoritmos de filtragem e de agrupamento.

Neste contexto, o módulo de tradução foi incluído na arquitetura da ferramenta de modo a permitir a criação de um ficheiro no formato GML a partir dos dados já trabalhados.

A criação deste ficheiro permite que ferramentas especializadas em análise de redes o possam importar com o intuito de, através do seu conteúdo, possibilitarem a observação e uma análise detalhada à rede gerada.

Após a execução dos algoritmos de filtragem e de agrupamento é iniciada a geração do ficheiro GML. A escrita deste ficheiro recorre às duas estruturas de dados auxiliares utilizadas para guardar as arestas e os nós da rede.

Os nós e as arestas da rede serão escritos no ficheiro considerando as especificações impostas pelo formato GML. As especificações do GML impõem que este formato seja construído através da inclusão de determinadas palavras-chave.

A primeira palavra-chave, *graph*, inicializa o ficheiro. Seguidamente, todos os nós do sistema serão incluídos e associados à palavra-chave *node*. Por fim, as ligações que representam as interações no sistema associar-se-ão à palavra-chave *edge*, e também estas irão fazer parte do ficheiro GML.

A seguinte Figura 3.4 esquematiza a sequência de processos necessários à geração do ficheiro GML.

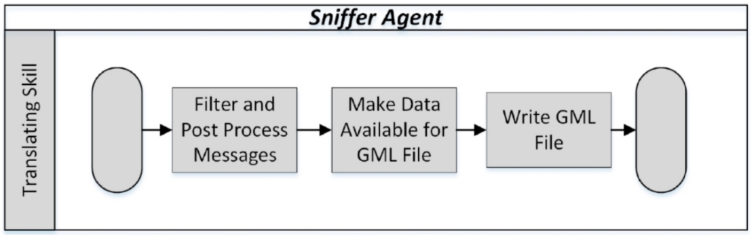

Figura 3.4: Tradução dos Dados para o Formato GML.

Ferramentas compatíveis com o formato GML, como o *Gephi*, encarregar-se-ão de efetuar uma caraterização da rede de agentes mecatrónicos no que toca às medidas tipicamente utilizadas para analisar redes. Tais como, o grau médio e a distribuição, o número de agentes e de interações, a persistência das interações, as mudanças na topologia do sistema, e outras relevantes.

Capítulo 4

Implementação

Este capítulo pretende clarificar os detalhes da implementação que constituem a ferramenta desenvolvida neste trabalho.

A seguinte secção apresenta uma breve descrição das tecnologias que suportaram toda a implementação. Posteriormente, através das restantes secções, será possível correlacionar as tecnologias utilizadas com o funcionamento interno da ferramenta.

4.1 Tecnologias de Suporte

4.1.1 *JADE - Java Agent Development Framework*

As plataformas multiagente têm sido utilizadas nos mais variados domínios com a missão de criar sistemas inteligentes e inerentemente robustos [RFMB14].

A plataforma JADE é uma tecnologia que permite uma programação distribuída no desenvolvimento de aplicações ponto-a-ponto e considera a execução de aplicações em tempo real que podem trabalhar em ambientes com ou sem fios [CPR03].

Esta tecnologia é implementada na linguagem *Java* e suporta um modelo com base no paradigma de agentes. O JADE implementa as especificações da FIPA e neste contexto, um ambiente em JADE é visto como um conjunto de *containers* onde vivem os agentes.

Os agentes são implementados através de *threads* em *Java* e executam as suas tarefas de forma cooperativa mediante as comunicações que estabelecem entre si.

Quanto ao controlo/gestão dos agentes, este é da responsabilidade dos componentes que integram o modelo de referência de gestão da plataforma JADE. Estes componentes são: o *Agent Platform*, o *Agent*, o Facilitador de Diretório (DF), o AMS e o *Message Transport Service*.

Neste contexto, as seguintes subseções apresentam alguns aspetos caraterísticos da plataforma JADE que foram essenciais à implementação da ferramenta desenvolvida nesta dissertação.

AMS - Agent Management System

O AMS é considerado um componente obrigatório na plataforma do JADE visto ser o responsável pela sua supervisão mediante a gestão das operações efetuadas sobre a plataforma. Este componente é o único que pode criar ou matar agentes, matar *containers* ou até encerrar a plataforma multiagente [BCG07].

Faz parte das competências deste agente efetuar o registo e o cancelamento do registo de todos os agentes da plataforma. Sempre que um novo agente entra na plataforma JADE, o AMS atribui ao novo agente um identificador único denominado de Identificador de Agente (AID) que o distingue dos restantes agentes. No caso do desaparecimento de um agente, o AMS procede ao cancelamento do registo do agente que desapareceu [BCG07].

Apenas um único AMS pode existir numa plataforma e no caso da plataforma abranger várias máquinas, o AMS é a autoridade responsável pela gestão de todas elas [BCG07].

Devido ao fato de este agente especial desempenhar o papel de autoridade na plataforma JADE, sempre que os agentes pretendem executar ações envolvidas na gestão da plataforma, como por exemplo subscrever o mecanismo de notificação dos eventos que ocorrem na mesma, estes deverão comunicar ao AMS o que pretendem e só depois, terão acesso livre para executar as ações pretendidas.

Tarefas dos Agentes

Os comportamentos associados às operações que os agentes desempenham durante o seu tempo de execução são denominados de *behaviours*. Para fazer com que um agente execute uma tarefa definida segundo um objeto do tipo *behaviour*, este deverá ser adicionado à classe principal do agente através do método *addBehaviour()*. Este método é disponibilizado pela classe *Agent* que, por sua vez, se encontra incluída no pacote nativo do JADE, *jade.core* [BCG07].

Os comportamentos podem ser adicionados ao agente a qualquer altura desde o início da sua execução, no método *setup()*, ou mesmo dentro de outros *behaviours* [BCG07].

Cada classe que estenda a classe *Behaviour* deverá implementar os métodos *action()* e *done()*. O método *action()* define as operações a realizar durante a execução do *behvaiour* e o método *done()* retorna um valor do tipo *boolean* que indica se o *behaviour* já se encontra concluído ou não. Quando os *behaviours* terminam deverão ser removidos da pilha de *behaviours* que o agente se encontra a executar [BCG07].

Comunicação de Agentes

A comunicação de agentes é considerada a caraterística mais importante do JADE [BCG07].

O paradigma da comunicação de agentes baseia-se numa troca de mensagens assíncrona e a cada agente é atribuída uma "caixa de correio"que o JADE utiliza para armazenar as mensagens enviadas pelos outros agentes [BCG07].

Sempre que uma mensagem é depositada na caixa de correio de um agente, o agente recetor da mensagem é automaticamente notificado [BCG07].

O formato das mensagens no JADE segue o formato definido na estrutura das mensagens Linguagem de Comunicação de Agentes FIPA (FIPA-ACL) e cada mensagem é constituída pelos seguintes campos:

- ***sender*** - Emissor da mensagem;
- ***receivers*** - Lista de recetores da mensagem;
- ***performative*** - Ato comunicativo que indica a intenção do emissor ao enviar a mensagem. Dependendo do protocolo de interação utilizado, a *performative* poderá ser *REQUEST*, *INFORM*, *PROPOSE*, *CFP (Call for Proposal)*, ou outras. No caso de *REQUEST*, o emissor pretende que o recetor execute uma ação, o *INFORM* notifica o recetor relativamente a um fato, e o *PROPOSE* ou *CFP* indicam que o emissor pretende iniciar uma negociação com o recetor da mensagem. Outros tipos de *performatives* podem ser encontrados em [FIP14];
- ***content*** - Contém a informação atual a ser trocada através da mensagem, como por exemplo a ação a ser executada numa mensagem *REQUEST*;
- ***language*** - Indica a sintaxe utilizada para expressar o conteúdo da mensagem. Para a comunicação ser eficaz, ambas as entidades emissora e recetora da mensagem

deverão ser capazes de codificar e analisar as expressões definidas pela linguagem referenciada neste campo;

- ***ontology*** - Indica o vocabulário dos símbolos utilizados no conteúdo da mensagem. Para a comunicação ser eficaz, ambas as entidades emissora e recetora da mensagem deverão atribuir o mesmo significado a esses símbolos;

- **Campos Adicionais** - Existem ainda outros campos adicionais utilizados para facilitar o controlo de várias conversas em simultâneo e para especificar os tempos limite para receção de respostas. Estes campos poderão ser o *conversation-id*, o *reply-with*, o *in-reply-to* ou o *reply-by*.

No JADE, a implementação de cada mensagem é conseguida através de um objeto da classe *jade.lang.acl.ACLMessage* que através dos seus métodos *get* e *set* permite o acesso a todos os campos especificados pelo formato ACL [BCG07].

Todos os tipos de *performatives* definidos pelas especificações FIPA são mapeados como constantes na classe *ACLMessage* [BCG07].

4.1.2 *Gephi - The Open Graph Viz Platform*

Para compreender as redes, várias técnicas que possibilitam a visualização de extensos grafos têm sido desenvolvidas nos últimos anos em vários projetos de sucesso. Estas técnicas são úteis para aumentar a capacidade de perceção dos seres humanos na procura de informação e propriedades que caraterizam a estrutura das redes [BHJ+09].

Contudo, este processo é difícil e requer alguma estratégia em termos de exploração das redes. Sendo tecnicamente necessárias e visualmente atrativas, as ferramentas que permitem a exploração e análise de redes devem encaminhar-se para os domínios de visualizações e análises em tempo real, a fim de, melhorar o processo de exploração por parte do utilizador [BHJ+09].

Neste contexto surge o *Gephi*, um software de código aberto que permite a exploração, a análise e a manipulação de grafos e redes [BHJ+09].

Os módulos do *Gephi* foram desenvolvidos de modo a permitir que o utilizador possa importar, visualizar, espacializar, filtrar, manipular e exportar todos os tipos de redes para o seu ambiente de trabalho [BHJ+09].

O *Gephi* incorpora na sua estrutura um módulo dinâmico que permite importar informações de uma rede através de um ficheiro de grafos compatível ou a partir de fontes de informação externa [BHJ+09]. Neste sentido, é considerado como uma ferramenta de elevado potencial no âmbito do estudo especializado em análise de redes.

GML - Graph Modelling Language

O GML é um formato de ficheiro portátil para representação de grafos. As suas caraterísticas chave são a portabilidade, a sintaxe simplificada, a extensibilidade e a flexibilidade [Him97].

Um ficheiro GML consiste em listas de chave-valor hierarquizadas que facilitam a construção de grafos a partir de estruturas de dados arbitrárias. Cada nó na árvore corresponde a uma chave etiquetada por determinado valor que identifica o nó [Him97].

Os grafos em GML são representados pelas chaves *graph*, *node* e *edge*. A estrutura topológica da rede é modelada de acordo com o identificador do nó (*node*) e com os atributos que correspondem à origem e ao destino de cada aresta (*edge*) do grafo. Como cada *edge* é constituído por dois nós, um de origem e outro de destino, o identificador do nó referencia a origem e o destino de cada *edge* no grafo [Him97].

4.2 Funcionamento Interno da Ferramenta

De acordo com as necessidades e com os requisitos respeitantes à análise e observação dos MAS, a plataforma JADE revelou ser a mais adequada para desenvolver a arquitetura proposta. Por outro lado, com recurso ao *Gephi* e ao formato GML foi possível deduzir e analisar certas caraterísticas típicas destes sistemas que permitem compreender o modo como as suas entidades se adaptam e reagem às constantes mudanças do seu ambiente.

Assim, com o auxílio das tecnologias apresentadas na secção anterior e da linguagem de programação *Java* foi possível implementar a arquitetura proposta.

Tradicionalmente, as ferramentas de *sniffing* limitam-se apenas a exibir as várias mensagens trocadas pelos agentes num sistema [RFMB14].

A ferramenta em questão vai um pouco mais além do tradicional pois, além de capturar as interações que ocorrem entre os agentes ao longo do tempo de execução do sistema, ainda tem a capacidade de inferir a rede dos agentes através da codificação da informação que resulta dessas interações no formato GML.

Após a captura das mensagens a informação é sujeita a um pós-processamento. Este pós-processamento efetua a codificação da informação no formato GML para que ferramentas especializadas em análise de redes permitam retirar conclusões mais aprofundadas relativamente à rede de agentes inferida [RFMB14].

A seguinte Figura 4.1 apresenta a esquematização de cada um dos processos que constituem a ferramenta desenvolvida.

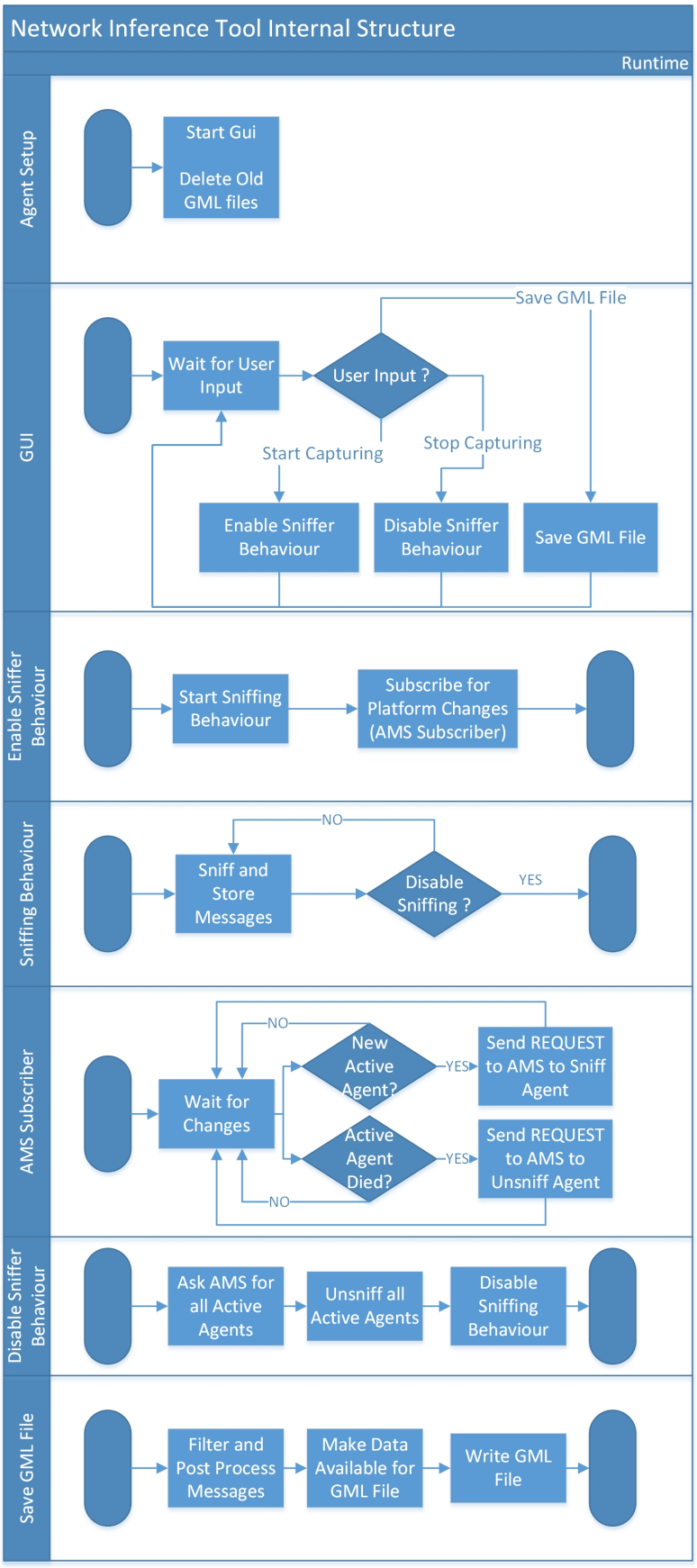

Figura 4.1: Arquitetura Interna da Ferramenta de Inferência da Rede de Agentes, adaptada de [RFMB14].

Cada um dos processos apresentados na Figura 4.1 desempenha uma tarefa específica na inferência da rede dos agentes do MAS.

De um modo geral, o *Agent Setup* inicializa o agente *Sniffer* e a GUI da ferramenta. A GUI da ferramenta reage às opções do utilizador e inicia os restantes processos conforme a opção escolhida. Caso o utilizador inicie a captura das mensagens (*Enable Sniffer Behaviour*), são automaticamente iniciados os comportamentos *Sniffing Behaviour* e *AMS Subscriber*. Quando o utilizador termina a captura (*Disable Sniffer Behaviour*), o *Sniffing Behaviour* é desativado. Por último, o processo *Save GML File* permite gerar o ficheiro GML necessário à inferência da rede de agentes do MAS mediante o pós-processamento da informação capturada.

As seguintes secções apresentam cada um destes processos e descrevem os seus detalhes de implementação. Todas as classes, métodos, objetos, variáveis e estruturas de dados utilizados na implementação de *JadeSniffer* encontram-se definidos no modelo de dados, esquematizado na seguinte Figura 4.2.

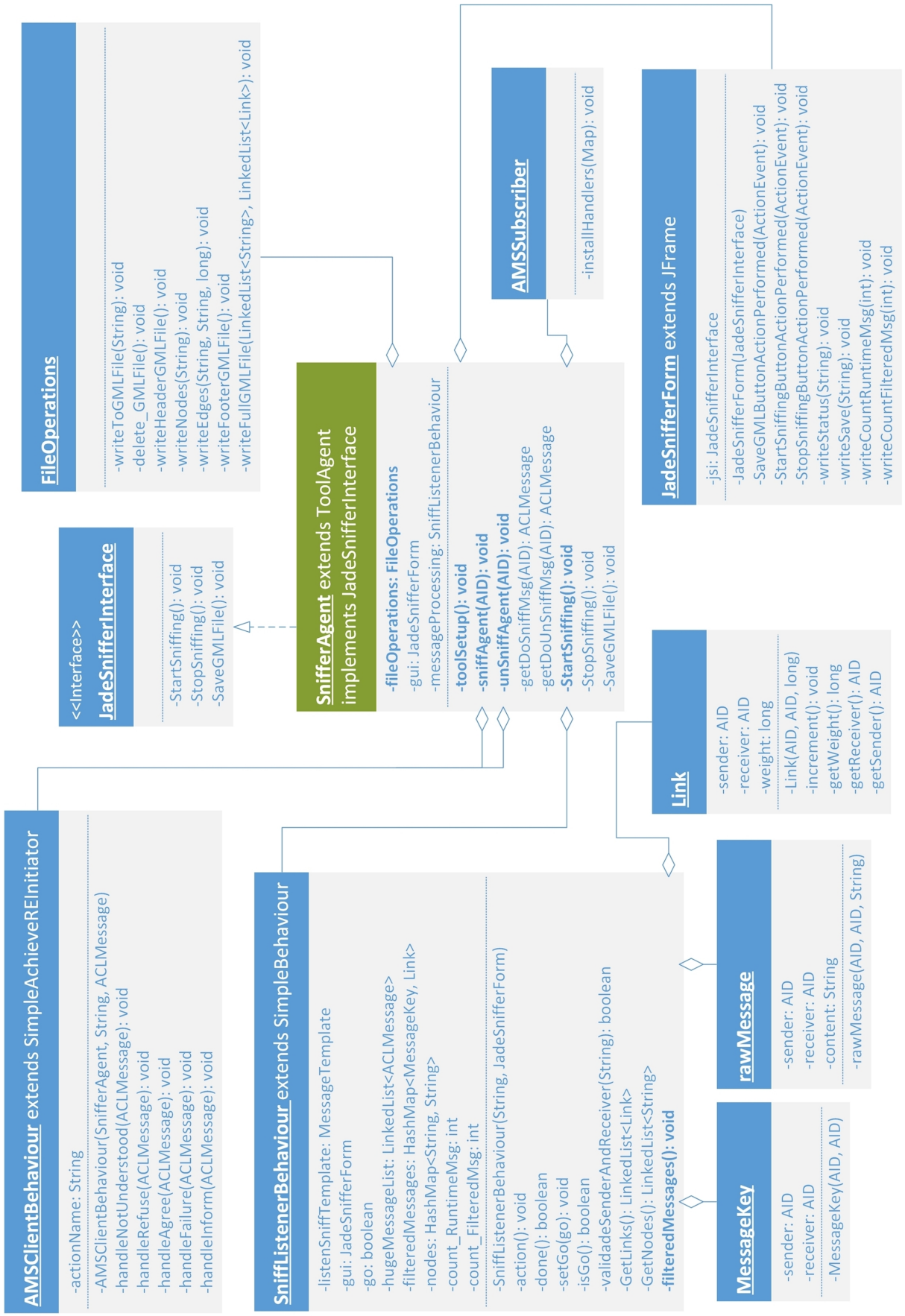

Figura 4.2: Modelo de Dados.

4.3 *Sniffer Agent*

4.3.1 Inicialização do Agente

Sendo que o agente desenvolvido é uma extensão da ferramenta *The Sniffer Agent*, apresentada em 2.3.2, para implementar este agente recorreu-se a duas classes nativas do JADE, *jade.tools.sniffer.Sniffer* e *jade.tools.ToolAgente*.

Observa-se no modelo de dados na Figura 4.2 que a classe principal do agente (*SnifferAgent*) é estendida à classe *jade.tools.ToolAgente*. Como se trata de um agente especial que implementa uma ferramenta, a classe principal do agente não contém um método *setup()*, mas sim um método *toolSetup()*.

O processo de inicialização do agente é o primeiro a ser executado e só ocorre uma vez a partir do momento em que o utilizador inicia a ferramenta. Todas as instruções que inicializam o *Sniffer* fazem parte do método *toolSetup()* e encontram-se representadas na seguinte Figura 4.3.

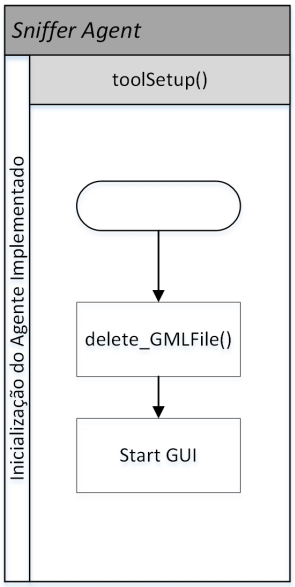

Figura 4.3: Inicialização do Agente *Sniffer*.

Tal como se pode afirmar perante a análise da Figura 4.3, a inicialização do *Sniffer* divide-se fundamentalmente em duas operações:

1. Remoção de possíveis ficheiros gerados em execuções passadas da ferramenta;
2. Criação e inicialização da interface gráfica da aplicação *JadeSniffer*.

A primeira operação tem o objetivo de remover os ficheiros gerados durante as execuções anteriores do programa e que possivelmente ainda se encontram arquivados na diretoria da aplicação. No caso da segunda operação, esta é caraterizada pela criação de uma pequena interface gráfica que permite ao utilizador controlar a execução da ferramenta.

As duas subsecções seguintes apresentam os detalhes relativamente ao modo de implementação dos dois blocos esquematizados na Figura 4.3.

Remoção de Ficheiros de Execuções Anteriores

Por cada execução da ferramenta, durante a observação de um sistema, são gerados novos ficheiros correspondentes. Como os ficheiros são guardados com o mesmo nome, o fato de ainda existirem ficheiros de execuções anteriores implica que toda a informação dos ficheiros consequentes seja adicionada ao ficheiro anterior, o que irá influenciar a integridade dos dados gerados.

Por este motivo, optou-se por incluir a operação de apagar ficheiros antigos logo no arranque da aplicação, para garantir que os dados gerados sejam exclusivos ao MAS em observação e não misturados com outros ficheiros de execuções anteriores.

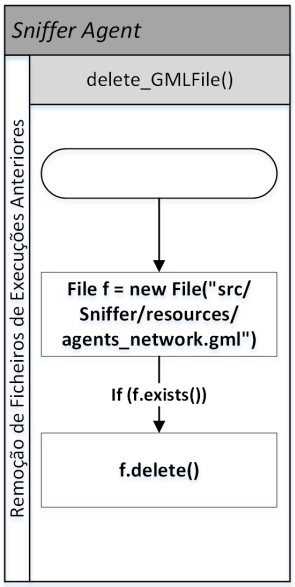

Figura 4.4: Remoção de Antigos Ficheiros GML.

Como se verifica através da Figura 4.4, para remover estes ficheiros implementou-se o método *delete_GMLFile()* na classe *FileOperations*. Este método acede à diretoria reservada para o armazenamento dos ficheiros e utiliza o método *delete()*, disponibilizado pela classe *File*, para apagar os ficheiros de execuções anteriores.

A classe *File* é uma classe nativa da linguagem *Java* e encontra-se disponível no pacote *java.io*.

Graphical User Interface da Aplicação

A interface visual da aplicação foi implementada no sentido de proporcionar ao utilizador uma utilização *user friendly* com base na simplicidade das operações disponíveis.

A GUI da ferramenta é um processo paralelo à execução do agente, ou seja, é uma *thread*. Como tal, o *Sniffer* controla a GUI a partir de uma interface *Runnable* de modo a não perturbar o seu funcionamento normal.

Assim, para integrar a GUI de *JadeSniffer* com o agente *Sniffer* implementou-se a interface *JadeSnifferInterface* que, tal como representado no modelo de dados, contém a definição dos três métodos associados às ações disponíveis na GUI da ferramenta: *StartSniffing()*, *StopSniffing()* e *SaveGMLFile()*. Estes métodos encontram-se implementados na classe principal do *Sniffer* e através desta interface, o *Sniffer* consegue controlar todas as operações efetuadas sobre a GUI.

Quando o *Sniffer* lança a GUI da ferramenta, esta mantém-se num estado de espera até surgir alguma ordem por parte do utilizador. Sempre que surge uma ordem do utilizador, a GUI interage com o *Sniffer* através do lançamento do *behaviour* que corresponde à ação do utilizador. Desta forma, a GUI está sempre a funcionar corretamente enquanto o *Sniffer* executa os *behaviours* que lhe competem, não havendo portanto interferências entre a execução da GUI e os comportamentos do agente.

A seguinte Figura 4.5 representa a interface gráfica desenvolvida e como se pode afirmar através da observação da mesma, o utilizador tem à sua disposição três botões de atuação.

Figura 4.5: Interface Gráfica de *JadeSniffer*.

Cada botão disponível representa uma ação e, tal como representado na seguinte Figura 4.6, o utilizador dispõe de três opções distintas.

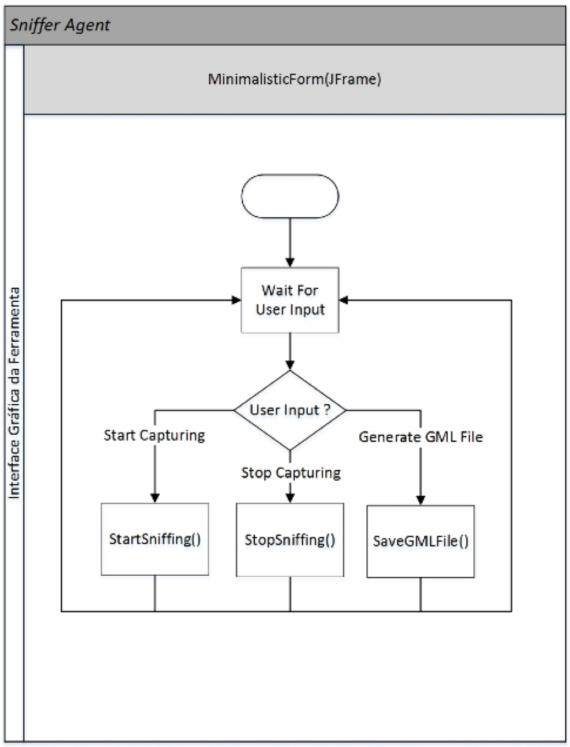

Figura 4.6: Funcionamento Interno da GUI de *JadeSniffer*.

As opções disponíveis na GUI da ferramenta são:

1. Iniciar a captura de mensagens trocadas no MAS;
2. Terminar a captura de mensagens trocadas no MAS;
3. Gerar e guardar o ficheiro GML.

Sempre que o utilizador aciona o botão que inicia a captura das mensagens ("*Start Sniffing*"), o comportamento de *Sniffing Behaviour* inicia o seu funcionamento através do método *StartSniffing()*. Este comportamento pertence à classe *SniffListenerBehaviour* e carateriza-se pela habilidade do agente *Sniffer* em capturar as interações que ocorrem em MAS.

Figura 4.7: Ação de Início da Captura de Mensagens do Sistema Multiagente.

No caso do utilizador optar por terminar a captura das mensagens no botão ("*Stop Sniffing*"), o comportamento de *Sniffing Behaviour* é desativado através do método *StopSniffing()*

Figura 4.8: Ação de Paragem da Captura de Mensagens do Sistema Multiagente.

Quando o utilizador pressiona o botão "*Save GML*"é invocado o método *SaveGMLFile()*. Este método processa a informação capturada e gera os dados necessários à criação do ficheiro GML que permite inferir a rede do MAS.

Figura 4.9: Ação de Geração e Armazenamento do Ficheiro GML.

4.3.2 Ativação do Processo de Captura de Mensagens

Posteriormente à sua inicialização, o *Sniffer* mantém-se num estado de espera até que o utilizador da ferramenta opte por iniciar a inferência da rede do seu sistema.

Nesta fase é importante referir que, quando o utilizador inicia a inferência da rede do seu sistema deve primeiramente lançar a ferramenta *JadeSniffer*. Só após o lançamento da mesma, deverá iniciar a execução do MAS.

Caso contrário, a ferramenta poderá não capturar a totalidade das mensagens trocadas. Se o MAS for iniciado primeiro e os seus agentes começarem logo a trocar mensagens, a ferramenta não irá capturar essas mensagens. Isto irá influenciar os dados gerados e, por conseguinte, a análise da rede de agentes.

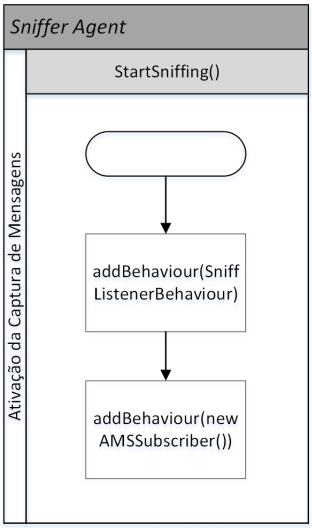

Figura 4.10: Ativação da Captura de Mensagens.

A Figura 4.10 mostra que para iniciar a captura das mensagens são adicionados dois comportamentos, ou *behaviours*, ao agente *Sniffer*. Estes *behaviours* são adicionados à classe principal do agente (*SnifferAgent*) no método *StartSniffing()*, através da instrução *addBehaviour()* que permite ao *Sniffer* começar a executar as tarefas que lhe competem.

O primeiro *behaviour*, denominado de *Sniffing Behaviour*, mantém-se à escuta de todos os eventos que correspondem à troca de mensagens entre os agentes ativos na plataforma JADE. Este comportamento é responsável pela captura e pelo armazenamento de todas as mensagens.

Quanto ao segundo *behaviour*, este denomina-se por *AMS Subscriber* e recorre a uma das funcionalidades do agente AMS. Pois, além de responder a pedidos de execução de operações respeitantes à gestão da plataforma, o AMS também suporta um mecanismo de subscrição tal que os agentes interessados podem receber notificações relativamente aos eventos que ocorrem em tempo real na plataforma.

Mediante esta funcionalidade e após a subscrição com o AMS, o *Sniffer* passa a receber todas as notificações relativas ao aparecimento e ao desaparecimento dos vários agentes que compõem o MAS em execução. Desta forma, é possível efetuar um balanceamento de recursos equilibrado respeitante às ações de *sniff* e *unsniff* efetuadas sobre os agentes do sistema.

As duas subsecções seguintes detalham o modo de implementação de cada um dos comportamentos de *Sniffing Behaviour* e *AMS Subscriber*.

Comportamento *Sniffing Behaviour*

O comportamento *Sniffing Behaviour* foi implementado na classe *SniffListenerBehaviour* e é composto por um algoritmo de funcionamento cíclico. Como se verifica no modelo de dados, a classe *SniffListenerBehaviour* estende um *SimpleBehaviour* que permite implementar os métodos *action()* e *done()* do *behaviour*.

A operação de um comportamento do tipo *SimpleBehaviour* é caraterizada por um ciclo que itera todas as instruções incluídas no seu método *action()*, até que surja uma ordem de paragem.

Nestes termos, após a execução da instrução *addBehaviour(SniffListenerBehaviour)*, ilustrada na Figura 4.10, é imediatamente iniciado um ciclo sobre as instruções incluídas no método *action()* do comportamento.

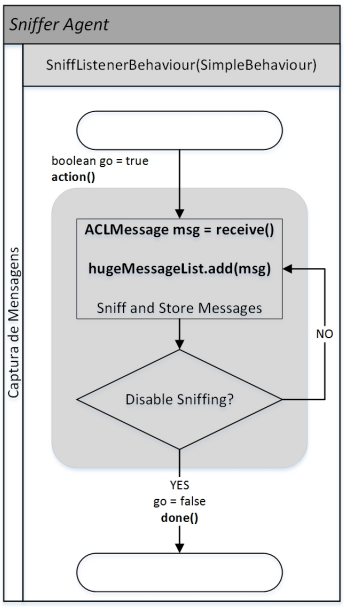

Figura 4.11: Funcionamento Interno de *Sniffing Behaviour*.

O funcionamento do *Sniffing Behaviour* encontra-se esquematizado na Figura 4.11 e tal como se pode observar, este comportamento limita-se a receber e a armazenar todas as mensagens que resultam das interações entre os agentes da plataforma.

Por cada mensagem trocada entre um par de agentes, o *Sniffer* recebe essa mensagem através do método *receive()* e guarda-a numa lista de mensagens do tipo ACL. A lista utilizada para este fim, definida na classe *SniffListenerBehaviour*, é do tipo *LinkedList* e denomina-se por *hugeMessageList*.

Para terminar o ciclo deste comportamento terá de existir uma ordem de paragem. Uma variável de controlo do tipo *boolean*, também definida na classe *SniffListenerBehaviour*, permite controlar as ordens de arranque e paragem do ciclo de captura. Esta ordem é disparada quando o utilizador termina a captura das mensagens.

Ou seja, o ciclo termina através da mudança de estado da variável *go*, ilustrada na Figura 4.11, para o estado de *false* no momento em que o utilizador pressiona o botão da GUI da ferramenta "*Stop Sniffing*". Após isto, o *behaviour* termina o seu processo de operação mediante o retorno do valor da variável *go* no método *done()*. No caso do arranque do ciclo, a variável *go* é colocada a *true* e a captura e o armazenamento das mensagens retomam o seu funcionamento normal.

No final da execução do sistema, a *LinkedList hugeMessageList* irá conter todas as mensagens trocadas durante a execução do sistema em tempo real. Além das mensagens entre os agentes do MAS em observação, a *LinkedList* também irá conter as mensagens enviadas ou recebidas pelos agentes nativos do JADE, tais como o AMS, o DF, o Agente de Monitorização Remota (RMA) e o próprio *Sniffer*.

Comportamento *AMS Subscriber*

Para subscrever o agente AMS com a finalidade de iniciar o mecanismo que permite ao *Sniffer* ser notificado sempre que surgem novos eventos na plataforma multiagente, recorreu-se aos conceitos da ontologia *JADE-Introspection* implementados pela classe *IntrospectionOntology*.

Estes conceitos encontram-se disponíveis no pacote *jade.domain.instrospection* nativo do JADE e todos os nomes que lhe correspondem são disponibilizados através de constantes na interface *InstrospectionVocabulary* [BCG07].

Todos os eventos que ocorrem na plataforma do JADE são descritos através de conceitos e todas as suas classes relacionadas implementam a interface *jade.domain.introspection.Event* [BCG07].

A tabela que se segue, adaptada de [BCG07], efetua a apresentação de alguns destes conceitos que descrevem os eventos da plataforma JADE.

Tabela 4.1: Exemplos de Eventos da Plataforma da Ontologia *JADE-Introspection*.

Nome da Ação	Classe	Descrição
added-container	AddedContainer	Indica que um novo *container* entrou na plataforma
removed-container	RemovedContainer	Indica que um *container* desapareceu da plataforma
born-agent	BornAgent	Indica que um novo agente nasceu num dado *container*
dead-agent	DeadAgent	Indica que um agente morreu num dado *container*
suspended-agent	SuspendedAgent	Indica que um agente ficou suspenso
resumed-agent	ResumedAgent	Indica que um agente recomeçou a sua execução
moved-agent	MovedAgent	Indica que um agente foi movido de um *container* para outro *container*
cloned-agent	ClonedAgent	Indica que um agente foi clonado num dado *container*

De entre os eventos tabulados apenas dois deles são utilizados, tendo em conta que, apenas dois eventos foram considerados relevantes à implementação prática desta dissertação.

Através da utilização da classe *AMSSubscriber*, também esquematizada no modelo de dados, foi possível fornecer ao agente *Sniffer* o método *installHandlers(Map handlers)*. Este método associa *handlers* aos eventos e recebe as notificações sempre que ocorre alguma mudança na plataforma multiagente. A classe *AMSSubscriber* encontra-se integrada no pacote *jade.domain.instrospection*.

Os *handlers* implementam a interface *AMSSubscribe.EventHandler* que inclui o método *handle(Event ev)*. A cada evento corresponde um método *handle* e, sempre que um evento do tipo associado ao *handle* é recebido pelo AMS, o método dispara com a receção de uma notificação.

Assim, no momento da execução da instrução *addBehaviour(new AMSSubscriber())*, incluída no método *StartSniffing()* e esquematizada na Figura 4.10, apenas são considerados os eventos de *born-agent* e *dead-agent*. Cada vez que nasce ou morre um agente na plataforma, o agente *Sniffer* recebe uma notificação no *handle* correspondente ao evento ocorrido.

A seguinte Figura 4.12 pretende esquematizar o funcionamento do *behaviour AMS Subscriber*.

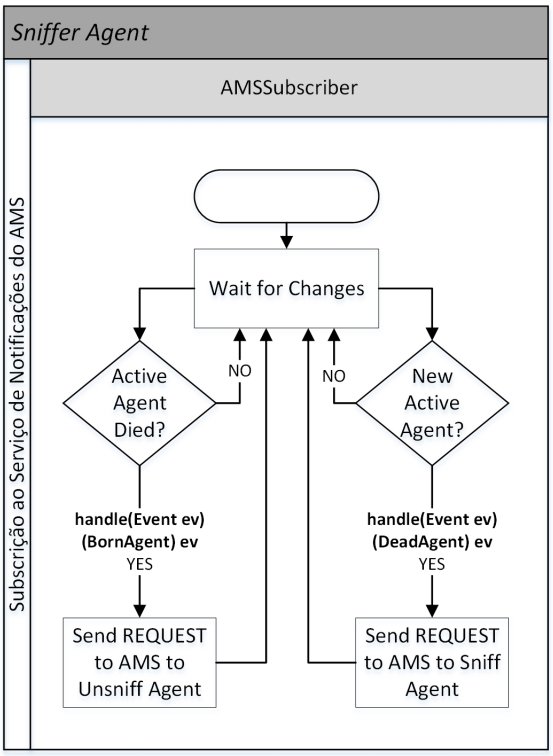

Figura 4.12: Funcionamento Interno de *AMS Subscriber*.

Tal como se pode afirmar através da observação da Figura 4.12, após a subscrição deste mecanismo, o *behaviour AMS Subscriber* mantém-se inicialmente num estado de espera "à escuta"de eventuais mudanças na plataforma.

A Figura 4.12 mostra que existem dois casos distintos a analisar:

- Evento correspondente ao aparecimento de um novo agente na plataforma JADE;
- Evento correspondente ao desaparecimento de um agente na plataforma JADE.

No primeiro caso, o método *handle* que corresponde ao evento *born-agent* dispara e, por conseguinte, o *Sniffer* analisa o novo agente. Esta análise é necessária para descartar os casos em que o agente que nasceu seja algum dos agentes nativos do JADE.

Quanto ao caso que corresponde à morte de um agente, o *handle* associado ao evento *dead-agent* dispara e, à semelhança do que acontece no nascimento de um agente, o agente removido também é sujeito a uma análise.

Posteriormente à analise do novo agente ou do agente que morreu, o *Sniffer* envia uma mensagem ACL do tipo *REQUEST* ao AMS pedindo para executar sobre o agente, a ação de *sniff* ou *unsniff* respetivamente. Daí que a análise aos agentes seja importante, pois, as ações de *sniff* e *unsniff* só devem ser efetuadas sobre os agentes do MAS em observação e não, sobre os agentes nativos do JADE.

As seguintes subsecções apresentam todos os detalhes da implementação dos processos de *Sniff* e *Unsniff* de agentes.

Processo de *Sniff* de um Agente

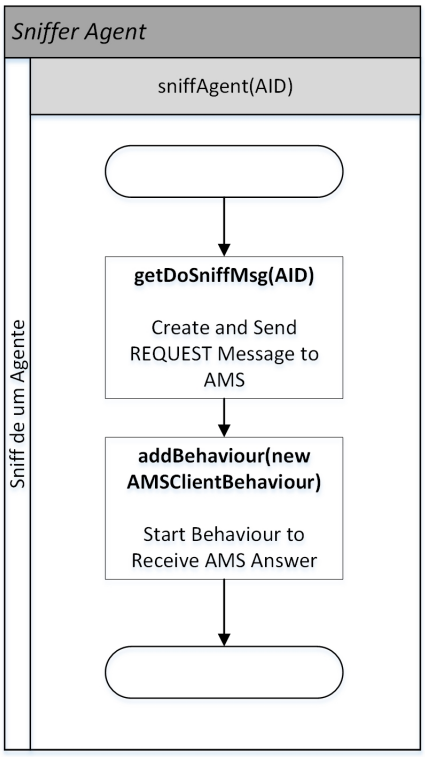

Figura 4.13: Esquematização do *Sniff* de um Agente.

Para efetuar o *sniff* de um agente ativo na plataforma, é necessário seguir uma sequência de passos para que o agente possa ser devidamente observado pelo *Sniffer* implementado.

Depreende-se da descrição apresentada em 4.3.2 que a última fase do mecanismo do *AMS Subscriber* implica uma comunicação com o agente AMS.

Como existe um pedido por parte do *Sniffer* ao AMS para efetuar o *sniff* de determinado agente que surgiu na plataforma, torna-se necessário criar uma mensagem ACL de *performative REQUEST* que irá ser enviada ao AMS solicitando a observação do novo agente.

Tal como representado na Figura 4.13, o processo de *sniff* divide-se essencialmente em duas fases:

1. Criação de uma mensagem ACL e envio da mesma ao agente AMS;
2. Criação de um novo *behaviour* que permite ao *Sniffer* receber a resposta do AMS à mensagem de *REQUEST* enviada.

Todas as operações associadas à gestão da plataforma JADE suportadas pelo agente especializado AMS são modeladas segundo as ações da ontologia *JADE-Agent-Management*. Esta ontologia é implementada pela classe *JADEManagementOntology* incluída no pacote *jade.domain.JADEAgentManagement* nativo do JADE [BCG07].

Assim, para enviar uma mensagem ao AMS com pedido de uma ação de *sniff* sobre um agente recorreu-se à utilização desta ontologia que, por sua vez, forneceu a ação *SniffOn*.

Como todas as ações que integram a ontologia *JADE-Agent-Management* devem ser requisitadas ao AMS dentro dos parâmetros do protocolo de interação *FIPA-Request* e devidamente codificadas na linguagem Linguagem de Semântica FIPA (FIPA-SL), a criação da mensagem a enviar ao AMS implica:

- Instanciação de um objeto que representa a ação de *SniffOn*;
- Definição do agente *Sniffer* como requerente da ação sobre o objeto instanciado;
- Adição do novo agente à lista de agentes sob ação de *sniffing*;
- Instanciação de um objeto que implementa o operador *action*, necessário para expressar nos parâmetros da linguagem FIPA-SL as ações dos agentes;
- Definição do AMS como ator na ação, na medida em que este será o recetor da mensagem;
- Definição do objeto representante da ação *SniffOn* no objeto que implementa o operador *action*;
- Declaração de uma mensagem ACL para formulação do pedido de *REQUEST* a enviar;
- Definição da ontologia *JADEManagementOntology.NAME* na mensagem declarada;
- Finalização da mensagem de *REQUEST* a enviar através do preenchimento da *slot :content* da mensagem ACL com os conteúdos da linguagem e da ontologia indicados nos campos *:language e :ontology* da mensagem, e da definição do conteúdo da mesma segundo o objeto já instanciado representante do operador *action*.

Todos os passos acima são executados através do método *getDoSniffMsg(AID)* implementado na classe *SnifferAgent*. Caso todos os passos tenham sido processados com sucesso, a mensagem *REQUEST* é finalmente enviada ao agente AMS.

Como se observa na Figura 4.13, a última fase deste processo resume-se à criação do *behaviour AMSClientBehaviour*, também representado no modelo de dados. Este *behaviour* mantém-se à espera da mensagem de resposta do AMS ao pedido de *SniffOn* do novo agente.

Traduzindo-se numa interação entre o agente *Sniffer* e o AMS, esta troca de mensagens entre estes agentes é implementada nos parâmetros especificados pela FIPA segundo o protocolo *FIPA-Request*.

Sempre que existe uma conversação entre dois agentes conduzida por um protocolo de interação, o agente que inicia a conversa toma o papel de *Initiator*, enquanto que o outro, recetor da primeira mensagem enviada, representa o papel de *Responder*.

Posto isto, torna-se claro que o agente *Sniffer* é nesta comunicação o *Initiator* e o AMS o *Responder*. Assim, a classe associada ao agente *Sniffer* que implementa o comportamento *AMSClientBehaviour* é uma classe *Initiator*.

Como mostra o modelo de dados, a classe *AMSClientBehaviour*, considerada uma classe do protocolo de interação e também um *behaviour* do JADE, é estendida à classe *SimpleAchieveREInitiator* inserida no pacote *jade.proto*.

Devido ao fato desta classe representar uma comunicação de 1:1, tendo em conta que o *Initiator (Sniffer)* envia uma única mensagem, a classe *SimpleAchieveREInitiator* revelou ser a mais adequada à implementação desta interação entre o *Sniffer* e o AMS.

Nos parâmetros do construtor da classe *AMSClientBehaviour* é incluída a *ACLMessage* de *performative REQUEST* enviada pelo agente *Sniffer* e que representa a mensagem de inicialização do protocolo. Esta classe implementa métodos do tipo *handle* para receber as mensagens de resposta vindas do *Responder* da conversa (AMS).

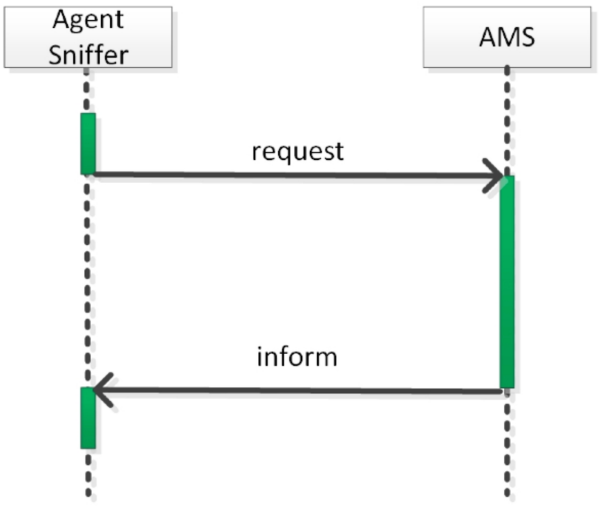

Figura 4.14: Comunicação entre Agente *Sniffer* e AMS.

A Figura 4.14 representa o protocolo de interação implementado entre o agente *Sniffer* e o AMS. Como se pode observar, em resposta à mensagem enviada pelo *Sniffer*, o AMS responderá com uma mensagem de *performative INFORM*. O método *handleInform(ACLMessage reply)*, implementado na classe *AMSClientBehaviour* irá disparar aquando da receção da resposta esperada. Posteriormente à receção do *INFORM*, o *behaviour AMSClientBehaviour* termina a sua execução.

Quando o *behaviour AMSClientBehaviour* termina, conclui-se o processo que envolve a ação de *sniff* de um novo agente que tenha surgido na plataforma do JADE. Consequentemente, o novo agente passará a estar sobre a observação do agente *Sniffer*, pelo que, todas as mensagens que o mesmo enviar ou receber serão capturadas e armazenadas no decorrer do algoritmo associado ao *Sniffing Behaviour* descrito em 4.3.2.

Processo de *Unsniff* de um Agente

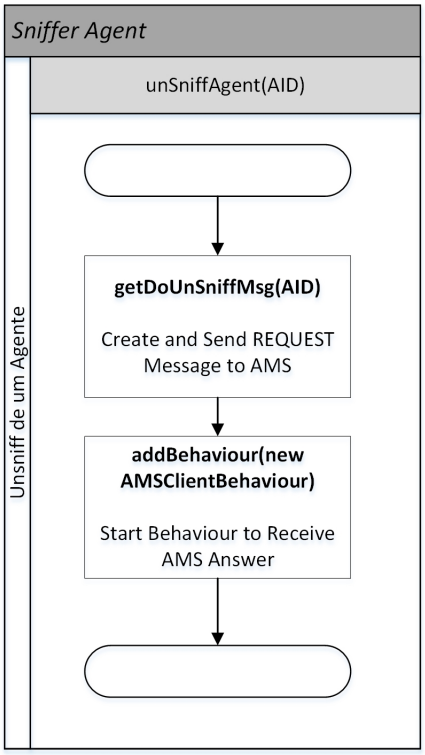

Figura 4.15: Esquematização do *Unsniff* de um Agente.

O *unsniff* de um agente representa o processo inverso ao *sniff*. Ou seja, ao invés de ser requisitada ao AMS a colocação de determinado agente num estado de observação, agora o pedido resume-se à remoção de determinado agente que já se encontre nesse estado.

A Figura 4.15 mostra que, também a ação de *unsniff* divide-se nas duas fases já apresentadas e discutidas no processo de *sniff*:

1. Criação de uma mensagem ACL e envio da mesma ao agente AMS;
2. Criação de um novo *behaviour* que permite ao *Sniffer* receber a resposta do AMS à mensagem de *REQUEST* enviada.

Contudo, surgem algumas diferenças relativamente à ação anterior que são de salientar.

Recorrendo novamente às ações da ontologia *JADE-Agent-Management*, é necessário enviar uma mensagem ao AMS requisitando que este volte a operar segundo a gestão da plataforma do JADE, mas agora para proceder à execução da ação de *SniffOff* sobre determinado agente que tenha desaparecido da plataforma.

Para a criação e formulação da mensagem com pedido de *unsniff*, implementou-se o método *getDoUnsniffMsg(AID)* na classe principal do *Sniffer*. Os passos a executar neste método são:

- Instanciação de um objeto que representa a ação de *SniffOff*;
- Definição do agente *Sniffer* como requerente da ação sobre o objeto instanciado;
- Adição do agente a remover do processo de *sniff* à lista de agentes sob ação de *unsniff*;
- Instanciação de um objeto que implementa o operador *action*, necessário para expressar nos parâmetros da linguagem FIPA-SL as ações dos agentes;
- Definição do AMS como ator na ação na medida em que este será o recetor da mensagem;
- Definição do objeto representante da ação *SniffOff* no objeto que implementa o operador *action*;
- Declaração de uma mensagem ACL para formulação do pedido de *REQUEST* a enviar;
- Definição da ontologia *JADEManagementOntology.NAME* na mensagem declarada;
- Finalização da criação da mensagem de *REQUEST* a enviar através do preenchimento da *slot :content* da mensagem ACL com os conteúdos da linguagem e da ontologia indicados nos campos *:language e :ontology* da mensagem, e da definição do conteúdo da mesma segundo o objeto já instanciado representante do operador *action*.

As operações efetuadas nos métodos *getDoSniffMsg(AID)* e *getDoUnsniffMsg(AID)* são praticamente iguais, no entanto a principal diferença baseia-se na definição da ação a requerer.

Enquanto que no caso do *sniff* a ação é de *SniffOn*, no caso de *unsniff* esta passa a ser de *SniffOff*. As restantes instruções que englobam o processo de *unsniff* acabam por ser idênticas ao processo de *sniff* e também elas são implementadas dentro dos parâmetros do protocolo de interação *FIPA-Request* e devidamente codificadas na linguagem FIPA-SL.

Segue-se a última fase do processo de *unsniff* de um agente. Conforme se mencionou, a sequência dos passos que englobam a criação do novo *behaviour* que permite ao *Sniffer* receber a resposta do AMS à mensagem de *REQUEST* enviada é idêntica à seguida no processo de *sniff*.

Como tal, o *behaviour AMSClientBehaviour* segue novamente o protocolo de interação apresentado na Figura 4.14 a partir do momento em que o *Sniffer* envia ao AMS a mensagem de *REQUEST* com o pedido de *unsniff* de um agente.

Quando o agente AMS responde com o *INFORM* informando o *Sniffer* que o agente foi removido do processo de *sniff*, o *behaviour AMSClientBehaviour* termina a sua execução.

Terminado o comportamento *AMSClientBehaviour* conclui-se também o processo de *unsniff* e, como tal, o agente que se encontrava sobre *sniff* deixa de o estar e, por conseguinte, possíveis mensagens associadas ao mesmo não serão mais capturadas ou armazenadas.

4.3.3 Desativação do Processo de Captura de Mensagens

Para efetuar a desativação do processo de captura das mensagens (*Sniffing Behaviour*) implementou-se o método *StopSniffing()* na classe *SnifferAgent*. O funcionamento deste método baseia-se num *behaviour* do tipo *OneShotBehaviour* e, como este tipo de *behaviour* conclui o seu processamento numa única fase de execução, todo o corpo do método *action()* que o constitui apenas é executado uma vez.

A Figura 4.16 apresenta o processo de desativação do *Sniffing Behaviour* e destacam-se três ações sequenciais.

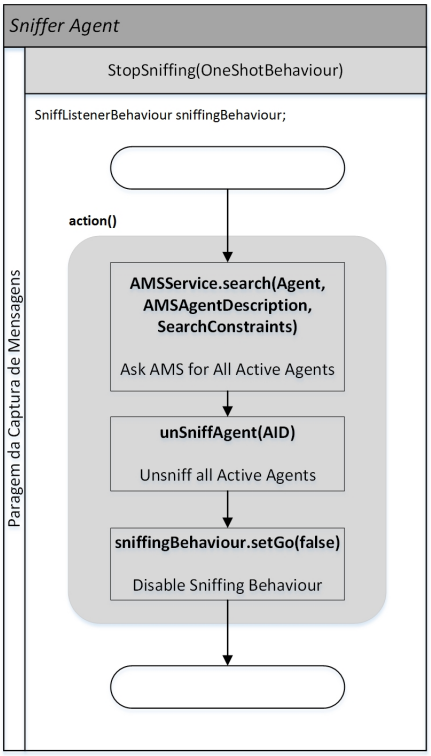

Figura 4.16: Desativação da Captura de Mensagens.

A primeira ação recorre novamente aos serviços do agente AMS mas desta vez com um pedido de informação relativamente a todos os agentes que se encontram ativos na plataforma. Para esse fim, utilizou-se a classe *AMSService* nativa do JADE que fornece um conjunto de métodos estáticos que permitem comunicar com um serviço do AMS [BCG07].

Assim, recorrendo aos serviços do AMS efetua-se a procura de todos os agentes ativos na plataforma e obtém-se um vetor constituído pelos mesmos.

A segunda ação, esquematizada na Figura 4.16, efetua o *unsniff* de todos os agentes ativos que estavam a ser observados. Ou seja, obtido o vetor de agentes, um ciclo *For-Each* é utilizado para percorrer o mesmo e em cada iteração é efetuado o *unsniff* do agente iterado. No fim, todo o vetor terá sido percorrido e os agentes que se encontravam sobre a ação de *sniff* terão sido removidos desse processo.

A última ação associada a esta desativação interage com a variável de controlo *go* do *Sniffing Behaviour* esquematizada na Figura 4.11. A alteração do valor da variável *go* para *false* é feita através do método *setGo(boolean)* implementado na classe *SniffListener-Behaviour*. Após a execução da instrução *setGo(false)* no método *StopSniffing()*, termina o ciclo do *SimpleBehaviour* a decorrer no *Sniffing Behaviour* e a captura das mensagens é desativada.

4.3.4 Tradução dos Dados num Ficheiro GML

Para o desenvolvimento deste processo implementou-se o método *SaveGMLFile()* que é definido por um *SimpleBehaviour*.

Quando o método *SaveGMLFile()* é invocado, o funcionamento cíclico, caraterístico deste tipo de *behaviours*, é iniciado sobre o corpo do método *action()* do *behaviour*.

As três ações que abrangem a tradução dos dados num ficheiro GML estão esquematizados na seguinte Figura 4.17 e são: a filtragem de mensagens, o agrupamento de nós e interações do sistema e a escrita do ficheiro GML.

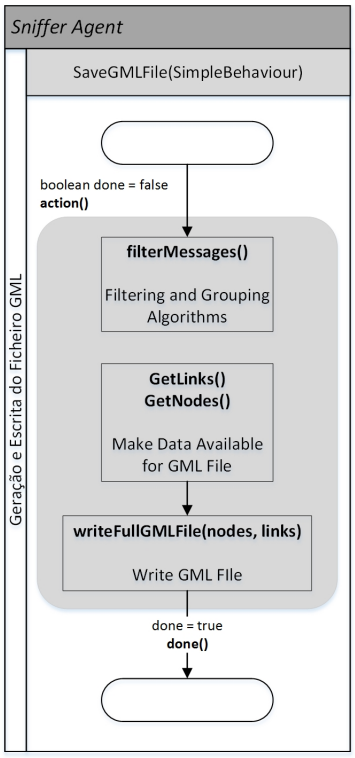

Figura 4.17: Processo de Geração e Armazenamento do Ficheiro GML.

As seguintes subsecções apresentam a implementação de cada etapa do processo de tradução dos dados num ficheiro GML.

Filtragem de Mensagens

Sendo que nesta etapa o utilizador já terminou a captura e o armazenamento de todas as mensagens trocadas durante a execução do MAS, a primeira ação do processo de geração e armazenamento do ficheiro GML, representada na Figura 4.17, efetua a filtragem das mensagens capturadas.

Para esse feito, desenvolveu-se um algoritmo de filtragem na classe *SniffListenerBehaviour* através do método *filterMessages()*. Este método irá ser aplicado à *LinkedList hugeMessageList*, mencionada em 4.3.2 e esquematizada na Figura 4.11, que neste momento já se encontra preenchida por todas as mensagens ACL capturadas.

O funcionamento do algoritmo de filtragem é caraterizado por um ciclo *For-Each* que percorre a lista *hugeMessageList* inspecionando cada uma das mensagens que esta contém. Por cada mensagem iterada no ciclo o seu conteúdo é analisado e, caso o evento ao qual esta se associa seja do tipo *SentMessage*, são extraídos os seguintes atributos:

- Emissor da mensagem;
- Recetor da mensagem;
- *Payload* da mensagem, correspondente a todo o "conteúdo escrito"enviado na mensagem.

Após a extração destes atributos, estes são utilizados para a criação de um objeto na classe *SniffListenerBehaviour* de tipo *rawMessage*, cujo construtor irá ser criado e definido consoante o valor que cada um destes atributos contém.

Os atributos de recetor e emissor do objeto *rawMessage* são seguidamente utilizados para criar e definir o construtor de um novo objeto de tipo *MessageKey*, também criado na classe *SniffListenerBehaviour*.

Enquanto o algoritmo do *Sniffing Behaviour* decorre não é aplicada nenhuma restrição ao armazenamento de mensagens. Caso contrário, seriam consumidos demasiados recursos do sistema, o que poderia originar a perda de mensagens.

Como tal, a lista de mensagens que se encontra a ser percorrida nesta etapa da filtragem (*hugeMessageList*) contém mensagens de todos os agentes que interagiram na plataforma, nomeadamente mensagens dos agentes nativos da plataforma JADE.

Assim, por cada mensagem iterada da *LinkedList hugeMessageList* e após a criação dos objetos *rawMessage* e *MessageKey*, é efetuada a validação do emissor e do recetor da mensagem. Esta validação é feita através de um método *booleano* que retorna o valor *true* caso as entidades em questão não sejam o DF, o AMS, o RMA ou o próprio *Sniffer*. Este método denomina-se por *validadeSenderAndReceiver(String)* e encontra-se implementado na classe *SniffListenerBehaviour*.

A validação destas entidades é indispensável no sentido em que permite selecionar as mensagens exclusivas ao MAS em observação. Pois, para inferir a rede do MAS, não deverão constar nos dados filtrados interações entre os agentes nativos do JADE.

Desta forma, caso o método *validadeSenderAndReceiver(String)* retorne *true* na análise ao recetor e ao emissor da mensagem, indicando que nenhuma das entidades corresponde aos agentes nativos da plataforma JADE, a mensagem é filtrada e o algoritmo de agrupamento inicia a sua execução. Caso contrário, a mensagem é descartada.

Agrupamento de Nós e Interações do Sistema

O objetivo deste algoritmo prende-se na definição de todos os agentes do sistema como novos nós da rede a inferir e no agrupamento das suas interações consoante as mensagens geradas em todas as suas comunicações.

Para implementar o agrupamento dos nós do sistema utilizou-se uma *HashMap* denominada de *nodes* e definida na classe *SniffListenerBehaviour*, que armazena os emissores e os recetores das mensagens como novos nós da rede a inferir.

Sendo a *HashMap* uma estrutura de dados abstratos cujas entradas são do tipo chave-valor, para cada nó é gerada uma entrada na estrutura *nodes* constituída pela chave e pelo valor representados pelo nome do agente correspondente ao emissor ou ao recetor da mensagem.

Assim, por cada mensagem filtrada no algoritmo de filtragem são criadas duas novas entradas na *HashMap nodes* para definir dois novos nós na rede considerando, para esse efeito, o par de agentes que interage na troca da mensagem.

Definidos os dois nós participantes da mensagem filtrada segue-se a criação de um objeto de tipo *Link* que define a aresta (ligação) originada pelo evento correspondente à troca da mensagem entre os dois agentes. Este objeto também se encontra definido na classe *SniffListenerBehaviour* e o seu construtor irá conter:

- Emissor da mensagem definido no objeto de tipo *rawMessage*;
- Recetor da mensagem definido no objeto de tipo *rawMessage*;
- Peso da ligação entre o emissor e o recetor da mensagem trocada.

Os algoritmos de filtragem e de agrupamento são finalmente concluídos através da criação de uma entrada numa nova estrutura também do tipo *HashMap*, denominada de *filteredMessages*. A estrutura *filteredMessages* também se encontra definida na classe *SniffListenerBehaviour* e a chave para cada uma das suas entradas é representada pelo objeto *MessageKey* e o valor dessa chave irá corresponder ao objeto de tipo *Link*.

Contudo, antes de serem criadas novas entradas nas *HashMaps nodes* e *filteredMessages* é realizada uma verificação sobre a *HaspMap filteredMessages*.

Nesta verificação é feita uma pesquisa na estrutura *filteredMessages* da chave *MessageKey* que representa a mensagem que se encontra nesse momento a ser iterada. Caso essa chave já exista em *filteredMessages* não serão criadas novas entradas em *nodes* ou em *filteredMessages*. Ao invés disso, é efetuado um incremento de uma unidade sobre o atributo peso do objeto *Link* que representa o valor da chave em pesquisa. Para incrementar o peso da ligação recorre-se ao método *increment()* definido na classe *Link*.

Acabadas as iterações efetuadas à lista de mensagens *hugeMessageList*, a *HaspMap filteredMessages* irá conter no final do ciclo todas as entradas relevantes à inferência da rede multiagente, que correspondem às ligações originadas pelas interações entre os pares de nós (agentes) no sistema. A *HashMap nodes* irá também apresentar-se devidamente preenchida contendo todos os nós do MAS observado.

Escrita do Ficheiro GML

Posteriormente à execução dos algoritmos de filtragem e de agrupamento, é necessário extrair todos os nós e todos *links* (ligações entre pares de agentes) filtrados, para inferir a rede do MAS através da construção de um ficheiro compatível com o formato GML.

Para extrair os nós e as ligações que representam a rede multiagente implementaram-se os métodos *GetLinks()* e *GetNodes()* na classe *SniffListenerBehaviour*. Ambos são executados mediante um ciclo *For-each* para percorrer, respetivamente, as estruturas *filteredMessages* e *nodes*.

O método *GetLinks()* percorre a estrutura *filteredMessages* e copia cada uma das suas entradas (do tipo *Link*) para uma nova *LinkedList* que guarda as ligações filtradas. Seguidamente, o método *GetNodes()* percorre todas as entradas da estrutura *nodes* e copia para uma outra estrutura também do tipo *LinkedList*, todos os nós da rede descritos pelo nome do agente que o representa. As duas *LinkedLists* irão permitir a escrita do ficheiro GML.

Com o intuito de criar um ficheiro GML compatível com ferramentas especializadas em análise de redes como o *Gephi*, desenvolveu-se o método *writeFullGMLFile(LinkedList nodes, LinkedList links)*. Este método encontra-se implementado na classe *FileOperations* e utiliza as *LinkedLists* que contêm as ligações e os nós da rede para criar o ficheiro pretendido.

A escrita do ficheiro GML divide-se em 4 fases:

1. Escrita do cabeçalho que inicia o ficheiro;
2. Escrita dos nós do sistema;
3. Escrita das ligações do sistema;
4. Escrita da terminação do ficheiro.

Para escrever o cabeçalho do ficheiro implementou-se o método *writeHeaderGML-File()*, também definido na classe *FileOperations*, cujo objetivo consiste apenas na escrita da palavra-chave *graph* e no caracter especial *[*, considerando as restrições impostas pelo formato GML. Assim, o início do ficheiro terá a seguinte apresentação:

graph
[

No que diz respeito à escrita dos nós e das ligações do sistema, esta utiliza para esse efeito dois métodos implementados na classe *FileOperations*, *writeNodes(String node)* e *writeEdges(String source, String target, Long weight)*.

Inicialmente, a estrutura dos nós (*LinkedList nodes*) é percorrida através de um ciclo *For-each* e por cada nó iterado é invocado o método *writeNodes(String node)*. Este método é responsável por escrever no ficheiro todos os nós do sistema seguindo as especificações do GML, ou seja, o nó será escrito no ficheiro de acordo com o seguinte formato:

node
[
identificador do nó (nome do agente)
]

Após a escrita dos nós segue-se a escrita das ligações (*edges*). A estrutura das ligações é percorrida e, mais uma vez recorrendo a um ciclo *For-each*, por cada ligação iterada invoca-se o método *writeEdges(String source, String target, Long weight)*. Este método escreve no ficheiro todos os *edges* capturados seguindo também um formato específico.

Os *edges* do sistema serão escritos no ficheiro GML de acordo com a seguinte estrutura:

edge
[
source (emissor da mensagem)
target (recetor da mensagem)
weight (peso da mensagem)
]

Terminada a escrita de todos os nós e ligações, o ficheiro GML termina através do método *writeFooterGMLFile()*, também implementado na classe *FileOperations*, que tem como objetivo colocar no final do mesmo o carater especial *]*, indicando que o grafo representante da rede de agentes se encontra concluído.

A tradução dos dados num ficheiro GML é então concluída através da alteração do valor da variável *booleana done*, esquematizada na Figura 4.17, responsável pelo controlo do comportamento do método *SaveGMLFile()*. Após a alteração do seu valor para *true* o *SimpleBehaviour* termina e, a partir desse momento, o ficheiro GML fica disponível para posterior utilização.

Depois da criação do ficheiro GML, o seu armazenamento é realizado na diretoria "**src\Sniffer\resources**"da ferramenta *JadeSniffer* e a sua descrição é denotada por "**agents_network.gml**".

Capítulo 5

Validação e Resultados

Com o propósito de efetuar uma validação e uma verificação adequada à aplicabilidade e à eficiência da ferramenta, recorreu-se à inferência da rede de um EPS.

Neste contexto, na secção 5.1 é efetuada uma breve descrição e caraterização do sistema utilizado nos testes e na validação da ferramenta tendo em conta o paradigma onde este se integra. Na secção 5.2 é apresentado o caso de teste utilizado na integração do sistema utilizado e da ferramenta. Na secção 5.3 são apresentados os resultados obtidos na aplicação do caso de teste. Por último, é efetuada a discussão dos resultados na secção 5.4.

5.1 Sistema para Demonstração da Ferramenta

A cooperação entre as entidades que abstraem os componentes físicos dos sistemas de manufatura evolutiva de automação é dominante nos paradigmas que emergem nos domínios da manufatura [RFMB14].

Assim, com vista ao alcance de uma manufatura cada vez mais ágil, a conceção de sistemas capazes de lidar com a incerteza e ambientes de alto dinamismo torna-se cada vez mais necessária [RFMB14].

5.1.1 Sistema Evolutivo de Manufatura

A essência do sistema testado na validação da ferramenta *JadeSniffer* centraliza-se na simulação de uma adaptação mecatrónica do *Firefly Algorithm* (FA) apresentado em [Yan09].

Conceitos como a auto-organização, a coerência e a emergência são conceitos dominantes nos sistemas modernos de manufatura. A auto-organização e a coerência asseguram a integridade lógica e estrutural do sistema em situações de mudanças e cenários inesperados. A emergência é fundamental para conduzir o sistema ao resultado pretendido ao longo do seu tempo de execução. Pois, sendo a emergência o fenómeno originado pela complexidade das interações estabelecidas entre os componentes do sistema, esta permite compreender de que modo é que estas interações influenciam o seu desempenho [FRA$^+$14].

A elevada complexidade que estes sistemas apresentam devido ao seu elevado número de componentes e à sujeição destes a ambientes incertos e de constantes mudanças constitui um objeto de estudo inovador [FRA$^+$14].

Esta inovação prende-se na semelhança entre os comportamentos dos sistemas modernos de manufatura e os comportamentos dos sistemas biológicos. Neste sentido, o sistema em análise foca-se no desenvolvimento e na investigação de aproximações bio-inspiradas, aplicadas ao controlo e à operação dos sistemas modernos de manufatura [FRA$^+$14].

Mediante a simulação do EPS considerado pretende-se adicionar ao sistema a dinâmica necessária para compreender a relação existente entre as várias partes do sistema e de que forma essa relação poderá influenciar o seu comportamento global [FRA$^+$14].

Assim, para atingir o nível de agilidade desejado num ambiente de elevado dinamismo, o conhecimento global do sistema encontra-se distribuído pelos vários módulos que o constituem e um mecanismo de controlo auto-organizado, inspirado em conceitos biológicos, permite gerir toda a operação do sistema [FRA$^+$14].

5.1.2 Caraterização do Sistema de Teste

Os pirilampos são entidades que dependem diretamente dos sinais bioluminescentes caraterísticos da sua espécie. Estes sinais são emitidos mediante flashes rítmicos de curta duração e são maioritariamente utilizados para atrair os machos que integram a espécie, as potenciais presas ou até para afastar possíveis predadores [FRA+14].

Tal como os pirilampos que emitem sinais de luz padronizados com a finalidade de atrair indivíduos da mesma espécie, o *shop-floor* pode ser definido como um ambiente onde as várias partes precisam de ser atraídas aos recursos para conseguir executar determinado processo ou compor um produto final [RFMB14].

Nestes termos, efetuando uma analogia entre os comportamentos que estas espécies apresentam e o *shop-floor* industrial, destacam-se as caraterísticas principais do sistema em descrição: modelos, atratividade e vizinhança.

As Figuras 5.1, 5.2 e 5.3 pretendem ilustrar as caraterísticas do sistema *supra* citadas e pertencem ao trabalho de investigação efetuado no desenvolvimento desta abordagem. O trabalho em questão encontra-se apresentado em [FER13].

Modelos

Como no ambiente de produção do EPS é necessário atrair as várias partes/peças aos recursos do sistema, à semelhança do que acontece nas populações de pirilampos que atraem as entidades da mesma espécie, surgiu o conceito de modelo [RFMB14].

Cada modelo define o tipo de peças que um recurso precisa de atrair para conseguir executar uma habilidade específica (*skill*). A execução da *skill* irá resultar no produto final. A soma de todas as peças corresponde às peças suportadas pelo respetivo recurso [RFMB14].

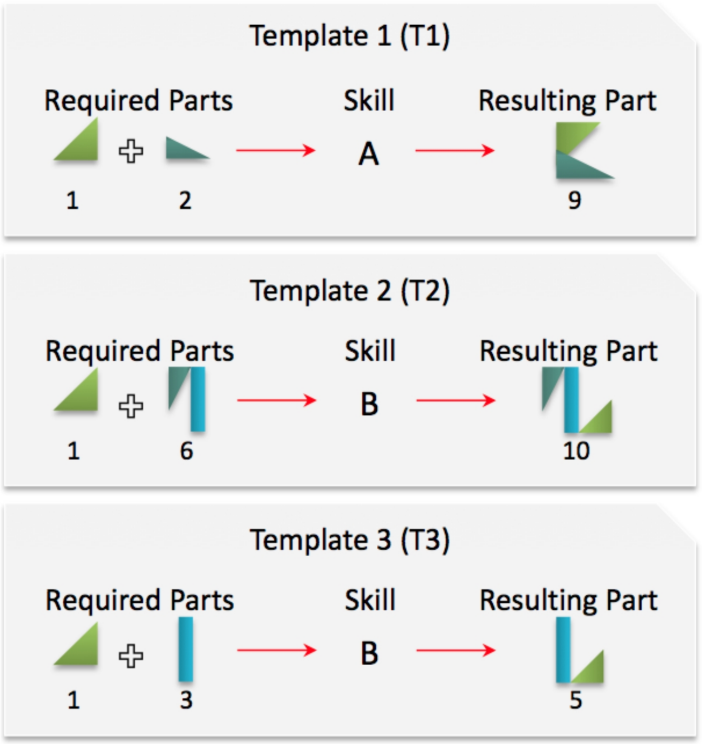

Figura 5.1: Exemplificação de Possíveis Modelos, retirado de [FER13].

Não é obrigatório que um modelo esteja sempre associado à atração de várias peças. Ou seja, nas operações de adição de valor, as várias peças são atraídas de forma a formar um conjunto de peças resultante, nas operações que não adicionam valor a peça não é atraída a nenhuma outra peça [FRA+14].

Para controlar a produção, os modelos são ativados ou desativados de acordo com as ordens pendentes. Por exemplo, se uma ordem de cem produtos de determinado tipo é recebida, os modelos necessários à produção daquele tipo de produto são ativados até que o sistema de produção cumpra os objetivos impostos por essa ordem [FRA+14].

Atratividade

Para traduzir o comportamento das entidades individuais em interações auto-organizadas e coerentes surgiu a necessidade de formalizar o conceito de atratividade também característico do sistema [FRA+14].

Independentemente deste conceito ser usualmente dependente da distância ao componente atrator, nesta abordagem assume-se que a atratividade é constante dentro de uma área de atração [RFMB14]. Por conseguinte, cada recurso do sistema tem tantas áreas independentes de atração como peças necessárias ao conjunto dos vários modelos que o recurso pode processar [FRA+14].

Se uma peça se encontra no interior de uma área de atração e essa área coincide com o tipo da peça, a peça é atraída em direção a esse recurso específico, caso contrário, a peça tende a retomar o seu percurso aleatório [FRA+14].

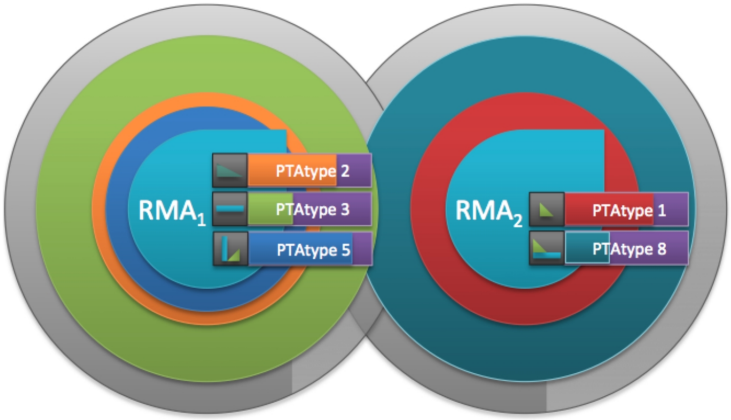

Figura 5.2: Áreas de Atração de dois Recursos Distintos, retirado de [FER13]. A área de atração do recurso da esquerda das peças de tipo 2 é representada pela cor laranja, das peças de tipo 3 pela cor verde e das peças de tipo 5 pela cor azul. A área de atração do recurso da direita das peças de tipo 1 corresponde à cor vermelha e das peças de tipo 8 à cor azul. A área de vizinhança entre os dois recursos encontra-se a cinzento.

Apesar do fato de não existir nenhuma noção de gradiente dentro das áreas de atração, o raio de cada área de atração varia de modo independente e de acordo com o tamanho da fila de cada tipo de peça em particular. A área de atração expande ou retrai consoante a procura relativa das várias peças [FRA+14].

Vizinhança

Ao contrário do que acontece na interação que ocorre na atração entre as várias peças e os recursos, a interação entre vizinhanças apenas ocorre entre recursos [RFMB14].

Cada recurso tem uma área de vizinhança correspondente. Consoante o local físico onde este se encontra, as áreas de vizinhança que os recursos intersetam são consideradas vizinhas [RFMB14].

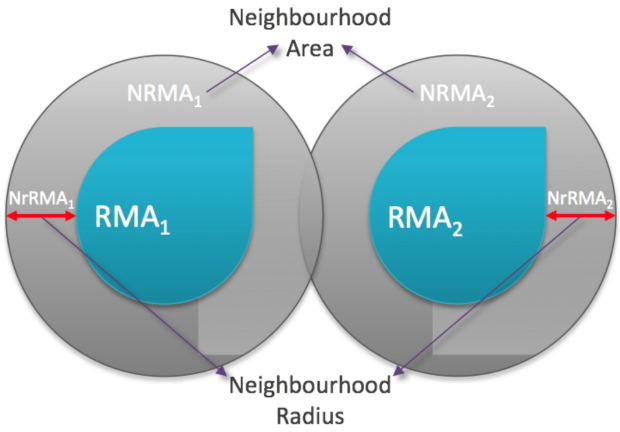

Figura 5.3: Interação de Vizinhanças entre Recursos, retirado de [FER13]. A área de vizinhança entre os dois recursos encontra-se representada pela cor cinzenta.

Embora as interações entre as vizinhanças não estejam diretamente relacionadas com o mecanismo de atração dos pirilampos, a influência que a vizinhança tem no sistema pode ser vista como uma analogia às condições do ambiente envolto na comunidade dos pirilampos [FRA+14].

Por exemplo, por um lado, se o ambiente é nebuloso o pirilampo terá pouca visibilidade e, consequentemente, será atraído pelo parceiro mais brilhante dentro de uma região pequena e limitada. Por outro lado, se o ambiente for nítido, o pirilampo será novamente atraído pelo parceiro mais brilhante mas agora numa área mais abrangente onde existem mais hipóteses de uma reprodução com melhores caraterísticas genéticas [FRA+14].

Desta forma, assim que uma peça entra na área de atração de um recurso, o seu campo de visibilidade é expandido com o intuito de abranger os vizinhos do respetivo recurso. Por sua vez, a peça é atraída ao recurso "mais brilhante"ou ao que se encontra mais próximo e que é compatível com ela [FRA+14].

Assim, mesmo que o recurso não seja compatível com a peça que entrou dentro da sua área de atração, existe sempre a hipótese dos seus vizinhos serem compatíveis com ela [RFMB14].

5.2 Caso de Teste

O caso de teste considerado na exploração e na validação da ferramenta foi concretizado com base na integração do EPS apresentado em 5.1.1 e da ferramenta *JadeSniffer*.

Através da utilização da ferramenta na captura das inúmeras interações do EPS foi possível inferir a rede de agentes do sistema selecionado. O que possibilitou um estudo detalhado relativamente à dinâmica das interações e à topologia da rede do EPS.

No caso de teste foram considerados diversos modelos. Os vários recursos foram instanciados e implantados no sistema a fim de simular um ambiente destinado à montagem de bicicletas [RFMB14].

A seguinte Tabela 5.1 revela a diversificação existente no sistema em estudo no que diz respeito aos variados tipos de modelos considerados.

Tabela 5.1: Modelos Considerados no Caso de Teste, adaptada de [RFMB14].

Modelo	Peças Necessárias	Habilidade	Peças Resultantes	Tipo do RMA
T1	1, 1, 10	A	17	WA1
T2	1, 1, 14	A	18	WA1
T3	1, 1, 15	A	19	WA1
T4	1, 1, 16	A	20	WA1
T5	2, 2, 11	A	25	WA1
T6	2, 2, 22	A	26	WA1
T7	2, 2, 23	A	27	WA1
T8	2, 2, 24	A	28	WA1
T9	4, 10	B1	14	SA1
T10	4, 15	B1	16	SA1
T11	4, 17	B1	18	SA1
T12	4, 19	B1	20	SA1
T13	5, 11	B2	22	SA2
T14	5, 23	B2	24	SA2
T15	5, 25	B2	26	SA2
T16	5, 27	B2	28	SA2
T17	7, 10	C1	15	HA1
T18	7, 14	C1	16	HA1
T19	7, 17	C1	19	HA1
T20	7, 18	C1	20	HA1
T21	8, 11	C2	23	HA2
T22	8, 22	C2	24	HA2
T23	8, 25	C2	27	HA2
T24	8, 26	C2	28	HA2

O plano representativo da estrutura dos diferentes recursos do EPS encontra-se ilustrado na seguinte Figura 5.4. Os diferentes recursos do sistema correspondem a cada um dos quadrados coloridos que se encontram desenhados na figura.

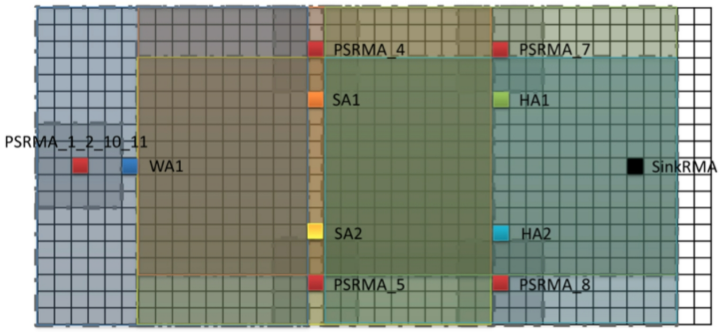

Figura 5.4: Plano Estrutural do Sistema Evolutivo Considerado nos Testes à Ferramenta, extraído de [RFMB14].

Os quadrados vermelhos correspondem aos alimentadores das várias peças essenciais à montagem do produto final. Portanto, cada um destes alimentadores tem como principal objetivo introduzir no sistema as peças que vão sendo requisitadas, durante a execução do sistema, pelos recursos que se encontram mais próximo [RFMB14].

No caso do quadrado preto representado, este representa um encaminhador no sistema. O encaminhador tem a responsabilidade de conduzir todos os produtos finais ao seu destino final [RFMB14].

No âmbito dos testes efetuados, foram considerados oito tipos de peças básicas necessárias à composição de bicicletas de dois tipos distintos. A Figura 5.5 que se segue pretende esquematizar a produção dos dois tipos de bicicletas tendo em conta as peças necessárias às montagens finais.

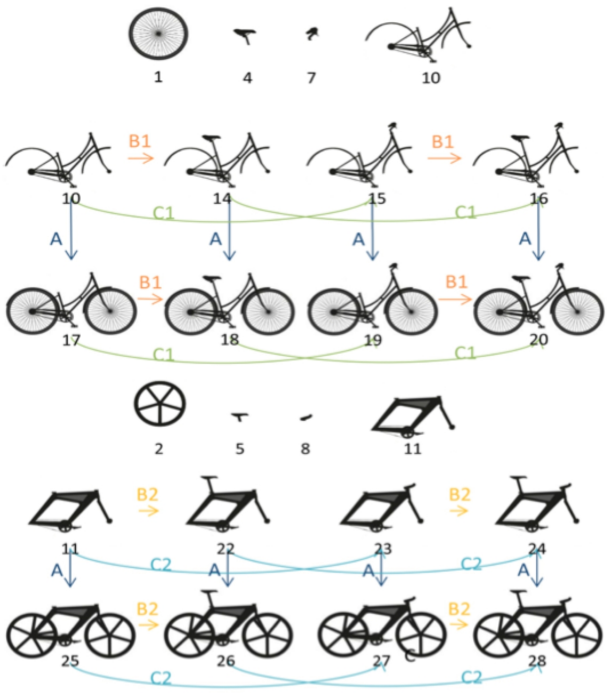

Figura 5.5: Montagem de Bicicletas e suas Partes Constituintes, retirado de [RFMB14].

Face ao exposto, para obter produtos finais de tipos diferentes, cujas representações na Figura 5.5 correspondem às montagens 20 e 28, é necessário processar determinado número de peças básicas em cada uma das montagens [RFMB14].

No processamento inicial das peças que constituem cada produto final considera-se que os movimentos associados às mesmas exibem um caráter livre e aleatório até que sejam atraídas pelos recursos que lhes correspondem. Assim, ambos os tipos de bicicletas 20 e 28 requerem a execução de três processos sequenciais e hierarquizados [RFMB14].

O primeiro processo é comum à produção dos dois tipos de bicicletas e resume-se à execução da habilidade A por parte do recurso WA1 [RFMB14].

Para a bicicleta do tipo 20, o segundo processo é executado pelo recurso SA1 através da habilidade B1 e o terceiro processo pelo recurso HA1 mediante a habilidade C1.

No caso da bicicleta de tipo 28, o segundo processo é executado pelo recurso SA2 através da habilidade B2 e por fim, o terceiro processo é da responsabilidade do recurso HA2 através da execução da habilidade C2.

Perante a análise paralela da Figura 5.5 e da Tabela 5.1 é possível validar a correspondência entre a produção de cada um dos produtos finais com os recursos e habilidades associadas.

Devido à distribuição do conhecimento pelos vários recursos do sistema e devido à agilidade que o sistema apresenta na resolução de problemas, as várias peças podem ser eficazmente montadas seguindo os três processos *supra* citados e descritos na Figura 5.5 [RFMB14].

Em suma, o cenário descrito foi utilizado como objeto de teste ao desempenho da ferramenta através da inferência da rede do sistema. Os objetivos de produção considerados resumiram-se à produção de 10 bicicletas do tipo 28 e 40 bicicletas do tipo 20 [RFMB14].

5.3 Resultados

Ainda que o objetivo deste trabalho não seja a validação do EPS selecionado, os resultados obtidos mostram claramente o comportamento do sistema. Neste sentido, o sistema utilizado permitiu validar a arquitetura e a ferramenta implementada e os resultados obtidos representam um método viável à explicação do comportamento do sistema e das relações ocultas entre os componentes que o integram.

Após a inferência da rede de agentes do EPS importou-se o ficheiro GML gerado pela ferramenta para o *Gephi* que, por sua vez, possibilitou a visualização da rede do sistema observado.

A seguinte Figura 5.6 mostra a rede dos agentes estruturada de acordo com a organização dos nós (agentes) no cenário de produção já mencionado.

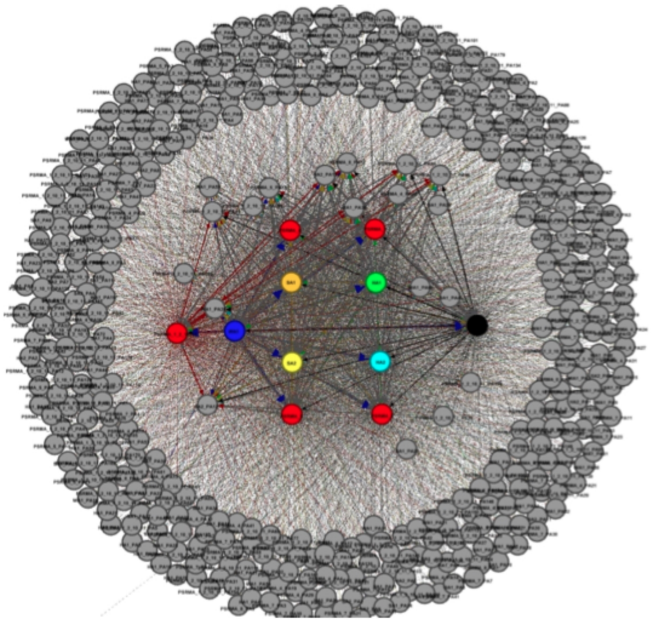

Figura 5.6: Estrutura da Rede do Sistema Inferido, retirado de [RFMB14].

Como propriedades do estudo especializado em análise e observação de redes, que visam o enriquecimento do conhecimento interno das mesmas, são consideradas relevantes à análise dos resultados obtidos o Grau de Entrada/Saída dos Nós, o Grau Médio e a Intensidade das Ligações.

5.3.1 Grau de Entrada/Saída dos Nós

Em cada nó existe uma distinção entre as suas ligações de entrada e as suas ligações de saída. As ligações de entrada são obtidas através do cálculo do grau de entrada (*indegree*) e as de saída através do cálculo do grau de saída (*outdegree*).

Para alcançar a estrutura de rede evidenciada na Figura 5.6 recorreu-se à distribuição dos graus dos nós do sistema.

Os nós que exibiram os maiores valores de grau de entrada, correspondem às entidades gestoras e controladoras do sistema de montagem e encontram-se localizados no centro da estrutura 5.6. No caso dos nós que revelaram os maiores valores de grau de saída, estes encontram-se posicionados na periferia da estrutura e correspondem, tal como era de esperar, às várias peças necessárias à montagem das bicicletas [RFMB14].

Como tal, tornam-se evidentes as duas comunidades emergentes no sistema deduzido. Sendo estas, as entidades centrais da estrutura representada, consideradas as autoridades do sistema e que correspondem aos recursos do sistema, e as restantes, as várias peças separadas das bicicletas.

A distribuição dos graus apresentada pelos nós do sistema foi expectável. Por um lado, os recursos que recebem a maioria das mensagens com pedidos para execução das suas habilidades apresentaram um elevado valor nas suas ligações de entrada, por outro lado, as peças constituintes de cada um dos produtos finais, vistas como clientes dos recursos, apresentaram o maior número de ligações de saída devido às várias mensagens que enviam aos recursos durante a execução do sistema.

5.3.2 Grau Médio

Recorrendo ao cálculo do número de ligações pertencentes a cada nó, dado pela propriedade do Grau Médio, obteve-se um valor de 21,676.

É pertinente justificar o valor obtido, considerando para esse efeito, a sua relação direta com o fato de as várias peças das bicicletas constituirem a maior comunidade existente no sistema e, por conseguinte, serem elas as principais entidades contribuintes ao valor em causa.

5.3.3 Intensidade das Ligações

Restringindo agora a análise efetuada aos nós que apresentaram um valor de grau de entrada acima de 22, obtém-se a seguinte estrutura representada na Figura 5.7 [RFMB14].

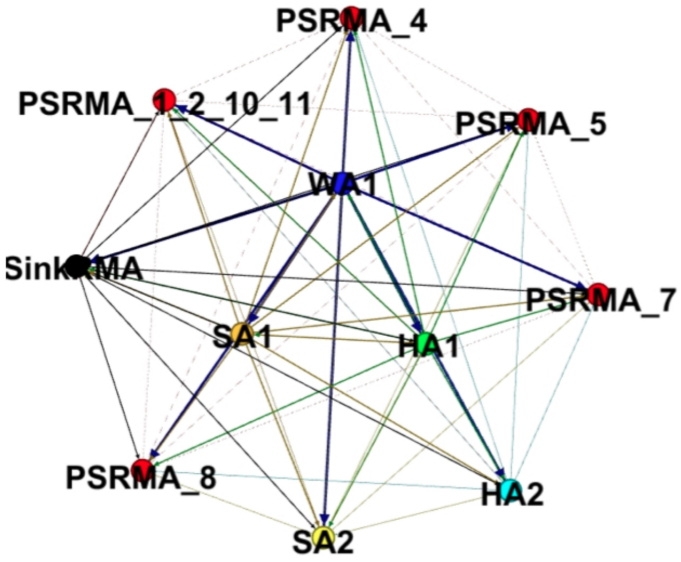

Figura 5.7: Rede das Principais Entidades de Execução (Grau de entrada acima de 22), retirado de [RFMB14].

Com o auxílio da Figura 5.7 é possível constatar através da espessura das linhas que interligam os nós da rede e que representa o peso das ligações, que alguns agentes interagem mais do que outros. Esta variação nas interações entre os agentes permite deduzir o padrão visível que acompanha a produção dos dois tipos de bicicletas [RFMB14].

Tal como já mencionado, o recurso WA1 é utilizado na montagem das duas bicicletas durante a execução do primeiro processo de produção. Sendo considerado como uma das principais entidades do sistema, fato que se justifica mediante o seu elevado valor de grau de entrada, foi possível apurar nos testes efetuados que este recurso respondeu a um total de 3684 pedidos de execução.

Após a execução do primeiro processo da produção, cada peça recorre a diferentes recursos consoante o tipo da bicicleta que representa. Posto que as ordens de montagem foram dadas em quantidades diferentes para dois tipos de bicicletas (10 bicicletas do tipo 28 e 40 bicicletas do tipo 20), é esperado que alguns recursos do sistema apresentem mais solicitações do que outros [RFMB14].

Deste modo, a distribuição das interações por ligação entre os agentes do sistema (peso das ligações entre os agentes) apresentou valores entre 1 e 399.

Com a finalidade de estudar e observar com um pouco mais de detalhe a influência do número de interações no sistema inferido, considerou-se conveniente efetuar uma análise adicional à rede do sistema. A Figura 5.8 pretende exibir esta análise e, consequentemente, permitir a retirada de mais algumas conclusões associadas à complexidade do sistema utilizado.

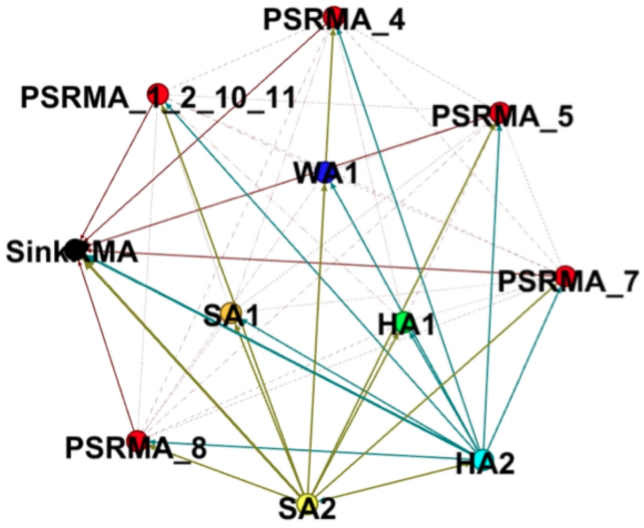

Figura 5.8: Processamento Adicional da Rede Inferida (Grau de entrada acima de 22 e peso das ligações abaixo de 93), retirado de [RFMB14].

Verifica-se através da observação da figura anterior que os recursos SA2 e HA2, responsáveis pela produção das 10 bicicletas de tipo 28 de entre as 50 que se pretendem produzir na totalidade, exibem um número diminuído de interações [RFMB14].

Considerando as várias interpretações quanto ao estudo dos sistemas mediante as suas perspetivas de rede, a ferramenta implementada permitiu fornecer uma visualização cumulativa da atividade associada ao sistema com base nas interações que ocorreram entre os seus agentes constituintes [RFMB14].

Neste contexto, a complexidade do EPS considerado foi de extrema importância nos testes ao desempenho da ferramenta. A validação da ferramenta envolveu um total de 472 agentes que interagiram entre si 10231 vezes.

Relativamente ao tempo de vida dos agentes, é de ressalvar que no decorrer da execução do sistema em tempo real, não se encontravam mais de 40 agentes ativos em simultâneo.

5.4 Discussão de Resultados

A simulação do sistema apresentado em 5.1 permitiu validar a ferramenta num ambiente cujas caraterísticas e comportamentos funcionais exibem elevados níveis de complexidade tendo em conta as inúmeras interações que ocorrem entre as suas entidades e o ambiente de incerteza onde as mesmas executam as suas tarefas.

A inferência da rede de agentes do EPS permitiu desvendar várias caraterísticas que influenciam o desempenho do sistema. O desmascarar destas caraterísticas forneceu o conhecimento necessário para melhorar o sistema considerado, e outros que sigam as mesmas abordagens, no sentido de encarar com agilidade as transformações sentidas nos seus ambientes de produção.

No âmbito dos MAS, as situações emergentes nos ambientes reais de produção podem sofrer alterações contextuais mesmo que as suas arquiteturas de referência sejam as mesmas. A calibração do ambiente de produção e a consequente validação do mesmo, constituem os pontos de maior desafio [RFMB14].

O EPS mostrou ser extremamente sensível aos vários cenários que surgiram no decorrer da sua execução. Pelo que, foi possível verificar que a constante competição das várias peças de montagem pelos vários recursos poderá implicar variações significativas no seu desempenho [RFMB14].

Embora os agentes sejam implementados com vista ao alcance de um objetivo global com determinados padrões associados, as condições locais do ambiente onde estes objetivos tendem a ser cumpridos pode influenciar drasticamente a execução dos comportamentos das entidades do sistema [RFMB14].

Se os agentes são providos de caraterísticas como a autonomia e cooperação, é relevante desenvolver métodos de observação e validação que nos permitam compreender a influência das variações dos seus comportamentos na organização interna do sistema e, por conseguinte, no desempenho global do mesmo [RFMB14].

Em suma, a ferramenta desenvolvida demonstrou ser uma metodologia viável e compatível com todos os MAS independentemente do grau de complexidade ou de dinamismo que estes apresentem [RFMB14].

A geração da rede de agentes do EPS por meio da ferramenta desenvolvida foi indispensável à caraterização quantitativa e qualitativa do sistema considerado. A análise da rede no *Gephi* possibilitou esta caraterização através da aplicação das várias métricas tipicamente utilizadas nos domínios do estudo e análise de redes [RFMB14].

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Conclusões

Os vários paradigmas emergentes nos domínios dos novos sistemas de produção de manufatura têm atingido níveis de complexidade que dificultam cada vez mais a monitorização e a previsão do comportamento destes sistemas.

A utilização da tecnologia de agentes na implementação destes sistemas tem sido amplamente utilizada e a forma como os agentes reagem aos vários comportamentos emergentes e em que medida é que as suas interações afetam o desempenho global do sistema constituem tópicos de investigação de grande desafio.

As ferramentas tradicionais que possibilitam a visualização e o estudo de MAS não apresentam o desempenho esperado em sistemas onde a carga de mensagens é intensa. A troca excessiva de mensagens entre os agentes poderá influenciar negativamente o desempenho da plataforma multiagente e, por conseguinte, o desempenho do sistema.

Como tal, as bases que cimentaram o desenvolvimento da ferramenta apresentada apoiaram-se neste desafio e permitiram implementar uma nova ferramenta que soluciona as limitações das tradicionais ferramentas de *Sniffiing*. Esta ferramanta contribui significativamente para a aquisição de conhecimento no âmbito dos MAS em contextos mecatrónicos.

A ferramenta desenvolvida tem a capacidade de capturar as mensagens que ocorrem no ambiente MAS sem perder mensagens independentemente da dinâmica ou composição do sistema. O processamento da informação angariada após a execução do MAS permite agilizar os processos que ocorrem em tempo real na plataforma multiagente de forma a que o desempenho da plataforma e do próprio sistema não sejam comprometidos.

Nos testes ao desempenho e à eficácia da ferramenta pôde concluir-se que esta reage bastante bem em ambientes de elevado dinamismo e de interações constantes. As simulações ao sistema de teste (EPS), que encenaram a produção de dois tipos de bicicletas em quantidades diferentes, revelaram que o sistema pode ser bastante sensível aos vários comportamentos emergentes.

A ferramenta assegurou a visualização da rede de agentes do EPS e possibilitou o estudo e a observação detalhada da totalidade das interações entre as entidades do sistema num ambiente que envolveu uma carga de mensagens intensa. O estudo e a observação do EPS permitiram, por sua vez, compreender de que modo é que as variações no ambiente de produção afetam o desempenho exibido pelos agentes durante o cumprimento das suas tarefas.

6.2 Trabalho Futuro

Após a realização do trabalho apresentado considera-se relevante melhorar a ferramenta desenvolvida de forma a que esta passe a suportar um maior número de funcionalidades que possibilitem a visualização e a análise da rede de agentes.

Estas funcionalidades poderão ser funcionalidades de visualização, de cálculo das diversas medidas de rede e até de segmentação da rede em períodos de tempo. Desta forma, será possível apreciar a evolução do sistema no domínio do tempo.

Para isso, poderá integrar-se na arquitetura interna da ferramenta algumas bibliotecas que o *Gephi* disponibiliza aos seus utilizadores. A integração destas bibliotecas na ferramenta permitirá que esta passe a conter muitas das funcionalidades disponíveis no *Gephi* excluindo a necessidade da utilização de ferramentas especializadas em análise de redes após a inferência da rede do sistema.

Na sequência da concretização deste trabalho foi elaborado um artigo que permitiu a exposição da ferramenta desenvolvida, [RFMB14]. O artigo foi apresentado na conferência INDIN (*International Conference on Industrial Informatics*) em Julho de 2014 [IND14].

Referências Bibliográficas

[BB03] Steve Brown and John Bessant. The manufacturing strategy-capabilities links in mass customisation and agile manufacturing–an exploratory study. *International Journal of Operations & Production Management*, 23(7):707–730, 2003.

[BC06] Radu F Babiceanu and F Frank Chen. Development and applications of holonic manufacturing systems: a survey. *Journal of Intelligent Manufacturing*, 17(1):111–131, 2006.

[BCG07] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[BHJ+09] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[CPR03] F Bellifemine G Caire, A Poggi, and G Rimassa. Jade. a white paper, 2003.

[ElM05] Hoda A ElMaraghy. Flexible and reconfigurable manufacturing systems paradigms. *International journal of flexible manufacturing systems*, 17(4):261–276, 2005.

[FER13] JOÃO DIAS FERREIRA. *Bio-Inspired Self-Organisation in Evolvable Production Systems*. PhD thesis, KTH School of Engineering and Management, SE-100 44 Stockholm, SWEDEN, September 2013.

[FIP14] Fipa communicative act library specification. Available: http://www.fipa.org/specs/fipa00037/SC00037J.html, 2014.

[FRA+14] João Dias Ferreira, Luis Ribeiro, Hakan Akillioglu, Pedro Neves, Antonio Maffei, and Mauro Onori. Characterization of an agile bio-inspired shop-floor. In *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*, pages 1–7. IEEE, 2014.

[FRN+12] João Ferreira, Luis Ribeiro, Pedro Neves, Hakan Akillioglu, Mauro Onori, and José Barata. Visualization tool to support multi-agent mechatronic based systems. In *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*, pages 4372–4377. IEEE, 2012.

[FS11] Regina Frei and Giovanna Di Marzo Serugendo. Concepts in complexity engineering. *International Journal of Bio-Inspired Computation*, 3(2):123–139, 2011.

[GEP14] Gephi - the open graph viz platform. Available: https://gephi.org/, 2014.

[Him97] Michael Himsolt. Gml: A portable graph file format. *Html page under http://www. fmi. uni-passau. de/graphlet/gml/gml-tr. html, Universität Passau*, 1997.

[IND14] 12th ieee international conference on industrial informatics. Available: http://indin2014.ece.ufrgs.br/index.html, July 2014.

[JAD14] Java agent development framework (jade). Available: http://jade.tilab.com/, 2014.

[KHJ+99] Yoram Koren, Uwe Heisel, Francesco Jovane, Toshimichi Moriwaki, G Pritschow, G Ulsoy, and H Van Brussel. Reconfigurable manufacturing systems. *CIRP Annals-Manufacturing Technology*, 48(2):527–540, 1999.

[KTŠS08] Jirı Kubalık, P Tichỳ, R Šindelár, and RJ Staron. Agents clustering within multi-agent system by means of multiobjective iterative algorithm. In *19th Eur. Meeting Cybern. Syst. Res*, volume 1, pages 555–560, 2008.

[KTSS10] Jiri Kubalik, Pavel Tichy, Radek Sindelar, and Raymond J Staron. Clustering methods for agent distribution optimization. *Systems, Man, and Cyber-*

netics, Part C: Applications and Reviews, IEEE Transactions on, 40(1):78–86, 2010.

[Lei09] Paulo Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, 2009.

[MN05] Charles M Macal and Michael J North. Tutorial on agent-based modeling and simulation. In *Proceedings of the 37th conference on Winter simulation*, pages 2–15. Winter Simulation Conference, 2005.

[MUK00] Mostafa G Mehrabi, A Galip Ulsoy, and Yoram Koren. Reconfigurable manufacturing systems: key to future manufacturing. *Journal of Intelligent Manufacturing*, 11(4):403–419, 2000.

[MVK06] László Monostori, József Váncza, and Soundar RT Kumara. Agent-based systems for manufacturing. *CIRP Annals-Manufacturing Technology*, 55(2):697–720, 2006.

[NFO+13] Pedro Neves, João Ferreira, Mauro Onori, Luis Ribeiro, and José Barata. Prospection of methods to support design and configuration of self-organizing mechatronic systems. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, pages 3854–3861. IEEE, 2013.

[NROB14] Pedro Neves, Luis Ribeiro, Mauro Onori, and José Barata. Performance assessment in self-organising mechatronic systems: A first step towards understanding the topology influence in complex behaviours. In *Technological Innovation for Collective Awareness Systems*, pages 75–84. Springer, 2014.

[OB09] Mauro Onori and José Barata. Evolvable production systems: mechatronic production equipment with process-based distributed control. In *9th IFAC Symposium on Robot Control: SYROCO 2009*. IFAC, 2009.

[OLBH12] Mauro Onori, Niels Lohse, Jose Barata, and Christoph Hanisch. The ideas project: plug & produce at shop-floor level. *Assembly automation*, 32(2):124–134, 2012.

[OOVSB03] Elth Ogston, Benno Overeinder, Maarten Van Steen, and Frances Brazier. A method for decentralized clustering in large multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 789–796. ACM, 2003.

[RB11] Luis Ribeiro and José Barata. Prospecting tools for mechatronic multiagent-based systems. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 369–374. IEEE, 2011.

[RBCO10] Luis Ribeiro, José Barata, Gonçalo Cândido, and Mauro Onori. Evolvable production systems: an integrated view on recent developments. In *Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology*, pages 841–854. Springer, 2010.

[RCB+11] Luis Ribeiro, G Candido, Jose Barata, S Schuetz, and A Hofmann. It support of mechatronic networks: A brief survey. In *Industrial Electronics (ISIE), 2011 IEEE International Symposium on*, pages 1791–1796. IEEE, 2011.

[RFMB14] Luis Ribeiro, João Dias Ferreira, Catarina Moura, and José Barata. A network inference tool for jade-based systems. In *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*. IEEE, 2014.

[SN99] Weiming Shen and Douglas H Norrie. Agent-based systems for intelligent manufacturing: a state-of-the-art survey. *Knowledge and information systems*, 1(2):129–156, 1999.

[STŠM07] Raymond J Staron, Pavel Tichỳ, Radek Šindelář, and Francisco P Maturana. Methods to observe the clustering of agents within a multi-agent system. In *Holonic and Multi-Agent Systems for Manufacturing*, pages 127–136. Springer, 2007.

[Tha96] A Tharumarajah. Comparison of the bionic, fractal and holonic manufacturing system concepts. *International Journal of Computer Integrated Manufacturing*, 9(3):217–226, 1996.

[TSS+06] Pavel Tichy, Petr Slechta, Raymond J Staron, Francisco P Maturana, and Kenwood H Hall. Multiagent technology for fault tolerance and flexible control. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(5):700–704, 2006.

[UHFV00] Kanji Ueda, Itsuo Hatono, Nobutada Fujii, and Jari Vaario. Reinforcement learning approaches to biological manufacturing systems. *CIRP Annals-Manufacturing Technology*, 49(1):343–346, 2000.

[VBWV+98] Hendrik Van Brussel, Jo Wyns, Paul Valckenaers, Luc Bongaerts, and Patrick Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in industry*, 37(3):255–274, 1998.

[WJ95] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152, 1995.

[Yan09] Xin-She Yang. Firefly algorithms for multimodal optimization. In *Stochastic algorithms: foundations and applications*, pages 169–178. Springer, 2009.