NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

JOÃO MIGUEL SILVA DE LIMA

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

ESTUDO E DESENHO DE ABORDAGEM AUTO-ORGANIZADA PARA SISTEMAS DE CONTROLO DISTRIBUÍDOS

MESTRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Universidade NOVA de Lisboa
Novembro, 2021

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

ESTUDO E DESENHO DE ABORDAGEM AUTO-ORGANIZADA PARA SISTEMAS DE CONTROLO DISTRIBUÍDOS

JOÃO MIGUEL SILVA DE LIMA

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Orientador: Doutor André Dionísio Bettencourt da Silva Parreira Rocha
Professor Auxiliar
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri:

Presidente: Doutora Anikó Katalin Horváth da Costa
Professora Auxiliar, Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Vogal: Doutor José António Barata de Oliveira
Professor Associado com Agregação, Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Convidado: Doutor André Dionísio Bettencourt da Silva Parreira Rocha
Professor Auxiliar, Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

MESTRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Universidade NOVA de Lisboa
Novembro, 2021

Estudo e Desenho de Abordagem Auto-organizada para Sistemas de Controlo Distribuídos

Aos meus pais.
Porque nunca teria conseguido aqui chegar sem vocês.

Agradecimentos

Esta dissertação marca o final de uma longa viagem através de inúmeros desafios, incertezas, infelicidades, alegrias e muitos precalços, mas que com o contributo de várias pessoas, foi possível cá chegar. Quero agradecer a todas estas pessoas que me ajudaram ou contribuíram para que conseguisse chegar ao fim desta viagem.

Especialmente ao meu orientador, Professor Doutor André Rocha, gostaria de lhe agradecer todo o apoio, orientação, paciência, empenho e motivação que me forneceu durante a execução e preparação da dissertação. Sem si não teria conseguido acabar a dissertação e ainda estaria bloqueado e perdido nos problemas que surgiram.

Obrigado a todos os amigos que estiveram sempre presentes. Especialmente ao André Cuco, Bernardo Lima, Bruno Silva, Daniel Caroço, Diogo (Arranja), Fábio Lopes, Gonçalo Queiroz, João Costa, João Macau, Miguel Lourenço, Nuno Martins, Pedro Corista, Pedro Costa, Pedro Garcia e Rui Trindade. Sem os momentos que passámos juntos, picardias, aventuras e apoio nem teria conseguido acabar a licenciatura, quanto mais o mestrado.

E claro, um eterno agradecimento a toda a minha família, mas principalmente aos meus pais, Aventino Lima e Ana Paula Lima, à minha irmã, Ana Lima, a minha avó Maria Odete e sua irmã Maria José, por todo o amor, partilha, companhia, generosidade e apoio incondicionais em todos os dias desta viagem atribulada. Também quero agradecer aos meus restantes avós, nomeadamente avó Deolinda, avô João e avô Miguel, que embora já não se encontrem presentes, fizeram parte desta viagem e na qual nunca foram, nem nunca serão esquecidos. Foi graças a todos vós que iniciei esta viagem, e foi por vocês que consegui acabá-la. A minha gratidão é eterna e considero-me uma pessoa sortuda por ter nascido e criado nesta família.

Obrigado a todos, do fundo do meu coração por tudo.

"It does not matter how slowly you go as long as you do not stop." (Confucius)

Resumo

Com os avanços tecnológicos nos últimos anos e com o aparecimento da Internet, a indústria teve de se revolucionar e adaptar a um novo mundo com um mercado extremamente volátil e competitivo, dando origem ao que conhecemos hoje como indústria 4.0. Com a Indústria 4.0 algumas tecnologias mais conhecidas como, por exemplo, a computação em nuvem, Big Data e inteligência artificial, ganharam importância. No meio destas emergiram os sistemas ciber-físicos, tecnologia que proporciona aos sistemas físicos serem monitorizados, controlados e integrados através de meios computacionais, que serve de base a muitas das tecnologias que encontramos na indústria 4.0.

Para implementar esta tecnologia no mundo industrial, muitos investigadores e projetos têm-se baseado numa tecnologia proveniente da Inteligência Artificial, os Agentes. Na indústria são utilizados sistemas multi-agente. Estes sistemas frequentemente possuem capacidades de auto-organização, que se devem à aplicação de diversos algoritmos. A maioria das arquiteturas que encontramos implementadas para sistemas distribuídos auto-organizados, utilizam um reduzido número de algoritmos e são construídas em torno desses algoritmos, sendo extremamente intolerantes à flexibilidade de utilizar outros algoritmos.

Neste trabalho propomos uma arquitetura para um sistema de controlo distribuído auto-organizado para manufactura, com a capacidade de se conseguirem integrar vários algoritmos diferentes. Esta arquitetura é baseada em agentes inteligentes e está dividida em três camadas responsáveis por: controlo e reprodução do chão da fábrica; armazenamento do estado do chão da fábrica e trajetos de produtos; optimização.

A abordagem generaliza os problemas que se encontram em sistemas industriais de produção e tenta apresentar uma base para servir de guia na construção de um sistema ciber-físico de produção que requeira flexibilidade nos algoritmos de optimização. Esta abordagem apresenta robustez, modularidade, flexibilidade e revela capacidades de auto-organização.

De forma a comprovar o comportamento e as capacidades desta arquitetura, foram realizados alguns testes através da simulação de um chão de uma fábrica e posteriormente tiradas as conclusões sobre os resultados obtidos.

Palavras-chave: Sistema Ciber-Físico de Produção, Sistema Multi-Agente, Auto-Organização, Algoritmos Optimização

Abstract

With technological advances in recent years and the emergence of the Internet, the industry had to revolutionise and adapt to a new world with an extremely volatile and competitive market, giving rise to what we know today as Industry 4.0. With Industry 4.0 some better-known technologies, such as cloud computing, Big Data and artificial intelligence, gained importance. In the midst of these, the cyber-physical systems emerged, technology that provides physical systems to be monitored, controlled and integrated through computational means, which is the foundation for many of the technologies found in Industry 4.0.

To implement this technology in the industrial world, many researchers and projects have relied on a technology derived from Artificial Intelligence, the Agents. In industry, several agents are generally used, leading to multi-agent systems. These systems often have self-organisation capabilities, which are due to the application of man-made algorithms. Most architectures we have found implemented for self-organised distributed systems use a small number of algorithms and are built around these algorithms, being extremely intolerant to the flexibility of using other algorithms.

In this work we propose an architecture for a distributed self-organised control system for manufacturing, with the ability to integrate several different algorithms. This architecture is based on intelligent agents and is divided in three layers responsible for: shop floor control and reproduction to virtual world; storage status of the shop floor and product paths; optimisation.

The approach generalises the problems encountered in industrial production systems and attempts to present a basis to serve as a guide in the construction of a cyber-physical production system that requires flexibility in the optimisation algorithms. This approach presents robustness, modularity, flexibility and reveals self-organisation capabilities.

In order to prove the behaviour and capabilities of this architecture, some tests were performed through the simulation of a factory floor and then conclusions were drawn on the results obtained.

Keywords: Cyber-Physical Production System, Multi-Agent Systems, Self-Organization, Optimization Algorithms

Índice

Índice de Figuras

Índice de Tabelas

Índice de Listagens

Glossário

admissível Uma função heurística é dita admissível se, a estimativa que ela calcular, nunca ultrapassar o custo mínimo real de chegar ao objectivo

consistente Uma função heurística é dita consistente se as suas estimativas forem sempre inferiores ou iguais à soma da estimativa do custo até ao objectivo de qualquer um nó, mais o custo de chegar até esse nó

debugging Processo de encontrar e reduzir/corrigir defeitos ou erros

grafos pesados É uma representação abstracta de um conjunto de objectos e das relações existentes entre eles. É definido por um conjunto de nós ou vértices, e pelas ligações ou arestas, que ligam pares de nós. Como as ligações possuem um peso ou custo associado, denomina-se de grafo pesado

middleware Software de computador que fornece serviços para outros softwares além daqueles que eles já possuem

script Programa escrito com uma sequência de instruções que o computador realizará num sistema, que automatiza a execução de tarefas que seriam executadas, uma de cada vez.

Siglas

ACO Ant Colony Optimization
AID JADE Agent Identifier

BCO Bee Colony Optimization
BMS Bionic Manufacturing Systems

CPPS Cyber-Physical Production Systems
CPS Cyber-Physical Systemss

DBCP Database Connection Pool
DF Directory Facilitator
DFD DFAgentDescription
DTO Data Transfer Object

EPS Evolvable Production Systems

FA Firefly Algorithm
FF Fractal Factory
FIPA Foundation for Intelligent Physical Agents
FMS Flexible Manufacturing Systems

HMS Holonic Manufacturing Systems

I4.0 Indústria 4.0
IEEE Institute of Electrical and Electronics Engineers
IoT Internet of Things

JADE JAVA Agent DEvelopment Framework
JDBC Java Database Connectivity

MAS Multi-Agent Systems

PA Product Agent
PID Controladores Proporcional Integral Derivativo
PLC Controladores Lógicos Programáveis
PSO Particle Swarm Optimization

RA Resource Agent
RMS Reconfigurable Manufacturing Systems

SD ServiceDescription
SQL Structured Query Language

1 Introdução

Este capítulo faz uma contextualização do trabalho proposto. É apresentada uma visão muito geral dos principais tópicos relevantes, nomeadamente Indústria 4.0, sistemas ciber-físicos e sistemas multi-agente. Também é apresentada a questão e hipótese que originou este trabalho. Por fim, é feita uma pequena descrição de como o documento está organizado.

1.1 Motivação e Hipótese

1.1.1 Motivação

Nas últimas décadas o desenvolvimento tecnológico tem sido colossal, e com o aparecimento da Internet, a sua evolução tem sido cada vez mais rápida. A tecnologia tem um papel essencial na vida das pessoas e afeta o modo como vivemos. Atualmente, a interação entre o mundo real e o virtual é cada vez maior e mais comum. Todos os anos é possível assistir a avanços em factores de desempenho, processamento, redução no tamanho e custos, tornando-a mais acessível a um número cada vez mais crescente de pessoas, caminhando para uma globalização digital/tecnológica.

Devido a estes avanços tecnológicos e à globalização que impulsionam a disseminação de ideias, culturas e o movimento de pessoas e bens, a Indústria teve de se revolucionar e adaptar a este novo mundo com um mercado extremamente volátil e competitivo, dando origem à quarta revolução industrial, conhecida como *I4.0*. Esta revolução visa a implementação e adaptação das novas tecnologias aos processos de produção, de modo a melhorar a eficiência, automatização, sustentabilidade e reduzir os custos associados.

Os princípios básicos determinados que sustentam cenários da I4.0 são [1, 2]:

- Interoperabilidade – é a capacidade de todos os equipamentos, numa dada planta fabril, conseguirem comunicar uns com os outros através de redes e semânticas predefinidas.

- Virtualização – significa que o sistema ciber-físico é capaz de monitorizar um sistema físico real, não virtual. Ou seja, com a ajuda de sensores, é criada uma cópia da fábrica para o mundo virtual onde toda a informação e estados de cada sistema é reproduzida e partilhada.

- Descentralização – significa que o controlo do sistema está distribuído (sistemas de controlo distribuído) e têm habilidade para tomarem decisões de forma autónoma e inteligente sem a intervenção de um operador humano. No entanto, para garantir estabilidade, evitar falhas críticas e manter a qualidade, em alguns sistemas, é necessária a implementação de uma entidade que regule e visualize o sistema como um todo em qualquer altura.

- Capacidade de resposta em tempo real – para que um sistema seja organizado e capaz de se organizar (auto-organizado), é importante que a informação recolhida por sensores e demais, seja imediatamente processada e analisada. Assim, a velocidade de resposta de um sistema influencia, consequentemente, o tempo que o sistema demora a reagir a uma falha e descobrir uma alternativa ou solução, ou seja, quanto mais rápido for a resposta do sistema a uma falha, melhor será a sua estabilidade e habilidade para evitar um erro de maior criticalidade.

- Orientação ao serviço/produto – a volatilidade do mercado tem levado a uma demanda cada vez maior de produtos individualizados e às vezes completamente personalizados. Esta mudança onde o comprador é que impõe as condições do produto, implica que as empresas optem por um novo método de negócio, que passa em vender os seus serviços e os de outros sócios ou parceiros, para que o resultado seja o produto proveniente de todas a especificações pedidas pelo cliente.

- Modularidade – capacidade de adaptação à inserção ou remoção de módulos para alterar/atualizar ou adicionar novos serviços. Este princípio é denominado de *Plug& Play* e rege, normalmente, por software e interfaces comuns a todos os aparelhos, possibilitando a sua identificação e uso imediatos.

- Adaptação ao humano – algo que não é de grande relevância para muitos, mas que é de grande importância para o futuro, é incluir o ser humano e não o descartar, considerando-o como uma peça essencial do sistema. Novos sistemas de manufactura devem seguir as necessidades do ser humano e não o contrário. A 'desumanização' pode trazer impactos sociais e económicos no futuro, onde a máquina substitui o homem e seguimos as necessidades da máquina.

A I4.0 impulsionou o estudo e desenvolvimento de algumas tecnologias essenciais como a computação em nuvem, Big Data, Inteligência Artificial, Internet of Things (IoT), etc., mas há uma que está associada a quase todas essas tecnologias, que não existira sem as outras (nomeadamente a IoT), é um alicerce para a I4.0 e é o principal tema abordado nesta dissertação. Os Sistemas Ciber-Físicos, denominado na literatura como Cyber-Physical Systemss (CPS). Esta é a tecnologia que proporciona sistemas físicos serem monitorizados, controlados e integrados através de meios computacionais[3–5]. Esta tecnologia acarreta vários desafios técnicos aos níveis de design, construção e verificação que apresentaremos à frente.

Para implementar esta tecnologia no mundo industrial, muitos investigadores e projetos têm-se baseado numa tecnologia proveniente da Inteligência Artificial, os *Agentes*. O conceito de *agente* está relacionado à inteligência artificial na forma em que ele representa um programa de software inteligente e autónomo. Na indústria, geralmente são usados vários agentes, dando à origem de Sistemas Multi-Agente ou MAS [6]. Esta tecnologia é uma ferramenta poderosa que proporciona aos sistemas de produção formas de se proteger de falhas, aumentar a eficiência, flexibilizar/adaptar, organizar, entre outras [7–9].

1.1.2 Hipótese

Existem sistemas capazes de se organizar automaticamente, estes sistemas na indústria geralmente utilizam arquiteturas que usam a tecnologia MAS e apresentam a habilidade de conseguir alterar a sua estrutura e/ou as suas funções em resposta a perturbações exteriores ao sistema. Estes elementos de auto-organização são capazes de manipular ou organizar outros elementos no sistema, de modo a que todo ele chegue a um ponto estável. Estes comportamentos são o resultado da aplicação de algoritmos, sendo que alguns representam ou foram inspirados em comportamentos de organização provenientes da natureza[10].

Embora existam vários algoritmos de otimização, a maioria das arquiteturas ou abordagens já implementadas para sistemas distribuídos auto-organizados, apenas utilizam um numero muito reduzido de algoritmos. Estas arquiteturas são construídas de raiz já com esses algoritmos em mente e são extremamente inflexíveis caso se pretenda alterar algum dos algoritmos utilizados. Ou seja, por exemplo e supondo que queremos aplicar uma arquitetura num problema qualquer, se quisermos utilizar uma arquitetura que já existe e que utiliza o algoritmo X, mas tivermos conhecimento de outro algoritmo Y que seria muito mais eficaz para o nosso problema, eventualmente chegaremos á conclusão de que, se quisermos realmente utilizar o algoritmo escolhido, muito provavelmente teremos de construir outra arquitetura de raiz em torno desse algoritmo Y. Por conseguinte, também não existe nenhuma norma que sirva de guia para construir sistemas deste tipo.

Esta inflexibilidade por parte das arquiteturas no que concerne o algoritmo utilizado, leva-nos á seguinte questão/hipótese:

Que arquitetura multi agente genérica exequível poderemos desenhar, de forma a implementar um sistema de controlo distribuído auto-organizado para manufactura, onde possamos recorrer de diferentes algoritmos, consoante os requisitos do sistema?

Em ambientes industriais de produção, sabemos que os sistemas distribuídos baseados em multi-agentes (MAS) na sua quase totalidade, contêm sempre dois tipos de agentes: agentes que controlam o material e/ou produto e agentes para controlar as entidades que processam ou transformam os produtos e/ou materiais. Também sabemos que estes sistemas comunicam geralmente com o mundo físico de forma padronizada através de PLCs e PIDs, ligados aos sensores e atuadores das respectivas máquinas. O que falta

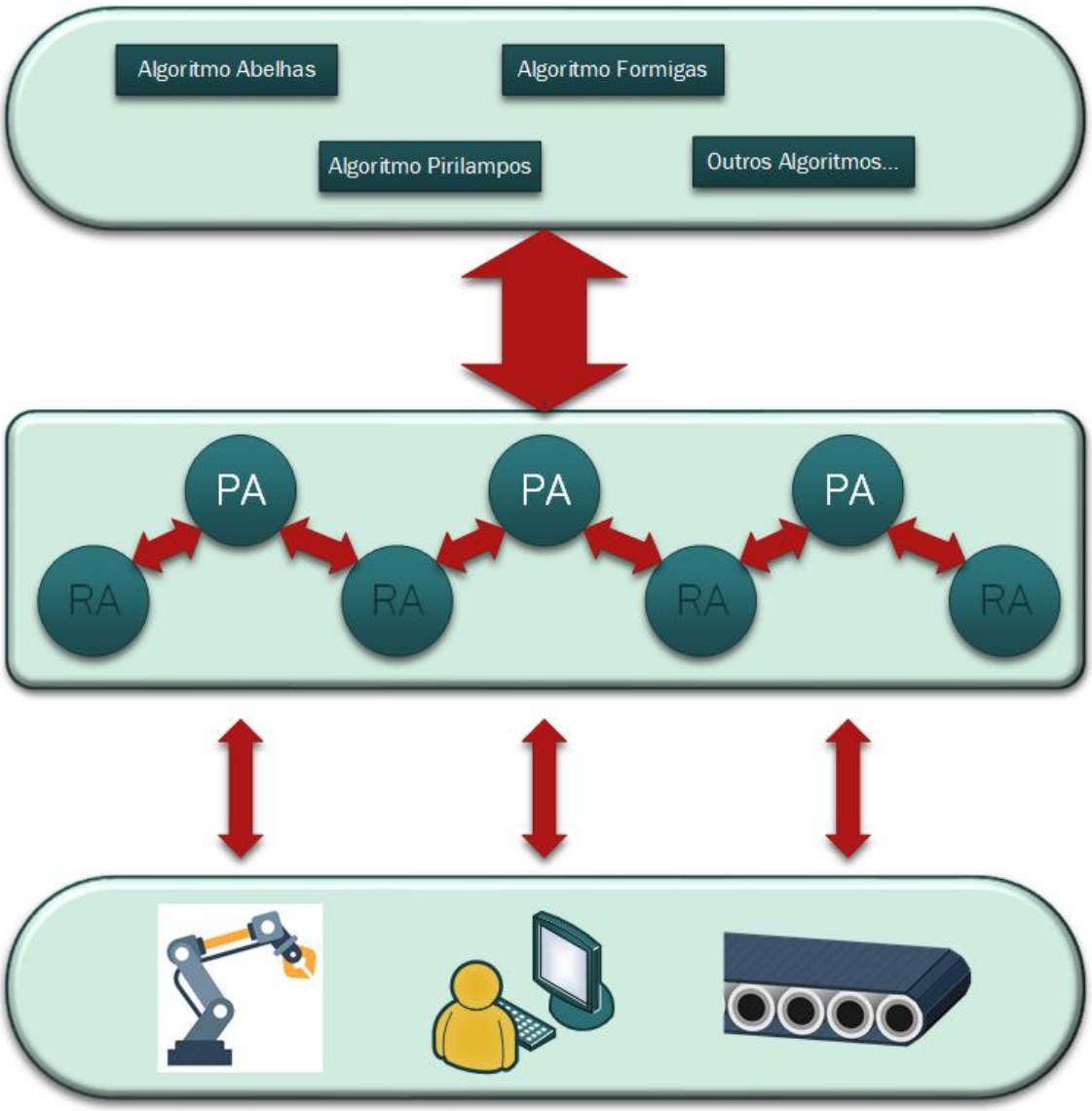

Figura 1.1: Diagrama da estrutura do MAS com Resource Agent(RA) e Product Agent (PA)

descobrir são os parâmetros de entrada comuns de entre todos os algoritmos e tentar criar uma ou várias saídas no sistema que sejam relevantes ao maior número possível de algoritmos. Queremos então encontrar uma arquitetura geral para Sistemas Ciber-Físicos de Produção, conhecidos como Cyber-Physical Production Systems (CPPS) na literatura, que torne flexível a escolha dos algoritmos a utilizar (Figura 1.1). Esta separação dos algoritmos das restantes entidades do MAS para uma camada independente, deverá conceder a tal flexibilidade desejada nos algoritmos.

1.2 Delineamento do Documento

Após uma pequena introdução aos principais tópicos, da questão e hipótese deste trabalho, os capítulos subsequentes estão organizados de forma seguinte.

O segundo capítulo apresenta uma revisão literária sobre variados temas considerados importantes para o desenvolvimento deste trabalho. É apresentada a evolução histórica, o aparecimento de sistemas ciber-físicos e o emergir dos mais recentes paradigmas na indústria e manufactura.

No terceiro capítulo estão presentes alguns dos conceitos mais importantes que suportam este trabalho. Designadamente, agentes, sistemas multi-agente, auto-organização e algoritmos bio-inspirados.

No quarto capítulo, é feita uma versão preliminar da arquitetura que soluciona o problema colocado.

O quinto capítulo será focado na implementação de um exemplo prático da arquitetura e irá entrar mais em detalhe relativamente às entidades que a compõem, suas interações e comportamentos.

No sexto capítulo são realizados alguns testes à arquitetura com algoritmos diferentes para problemas idênticos. São depois exibidos, analisados e comentados os resultados obtidos, realçando as diferenças entre algoritmos.

Por fim, no sétimo e último capítulo é feita uma revisão crítica ao trabalho realizado e são apontados os principais pontos importantes a reter da dissertação. De seguida mostra-se alguns desafios que ficaram em aberto para futuros trabalhos.

2 Estado de Arte

Neste capítulo é apresentada uma revisão literária sobre os variados temas considerados importantes para o desenvolvimento deste trabalho. Primeiramente é feita uma apresentação da evolução histórica na indústria, depois abordamos o emergir dos mais recentes paradigmas na indústria e manufactura. Por fim o aparecimento de sistemas ciber-físicos.

2.1 Indústria 4.0

Nas primeiras três revoluções industriais, os humanos criaram tecnologias mecânicas, eléctricas e de informação, que levaram a uma otimização na produção e processos industriais. A primeira revolução melhorou a eficiência através do uso do poder hídrico e máquinas a vapor; a segunda revolução trouxe a eletricidade e a produção em massa através das linhas de montagem; na terceira revolução surgiu a implementação da automação com o uso de novas tecnologias em eletrónica e informação. A quarta revolução industrial está a emergir, sendo liderada pela recente tecnologia dos sistemas ciber-físicos para integrar o mundo real no virtual como indústria do futuro (Figura 2.1).

A Alemanha desenvolveu um plano estratégico para facilitar a conversão da indústria 3.0 para a 4.0. Os pontos principais deste plano, em [12], são de modo geral: a construção de uma rede ciber-física, o foco no estudo e desenvolvimento de temas relacionados com as fábricas e produção inteligentes, realização de três integrações e alcançar oito objetivos de planeamento.

A três integrações são: a horizontal, a vertical e de ponta-a-ponta (end-to-end). A integração horizontal refere-se à integração entre os recursos e a rede de informação, para alcançar a cooperação entre empresas e tornar possível o serviço em tempo real. A integração vertical implica a interligação dos sistemas numa fábrica inteligente e manufactura customizada como alternativa à tradicional. Integração ponta-a-ponta significa a integração de toda a cadeia industrial e integração entre empresas para criar o máximo de customização.

Os oito objetivos de planificação que formam os básicos para alcançar a I4.0, são: uniformização de sistemas e construção de uma arquitetura de referência; gestão eficiente; estabelecer uma infraestrutura compreensiva e fiável; segurança; organização e desenvolvimento de trabalho; treino e formação dos trabalhadores; estabelecer uma estrutura de trabalho regulatória; melhorar a eficiência dos recursos.

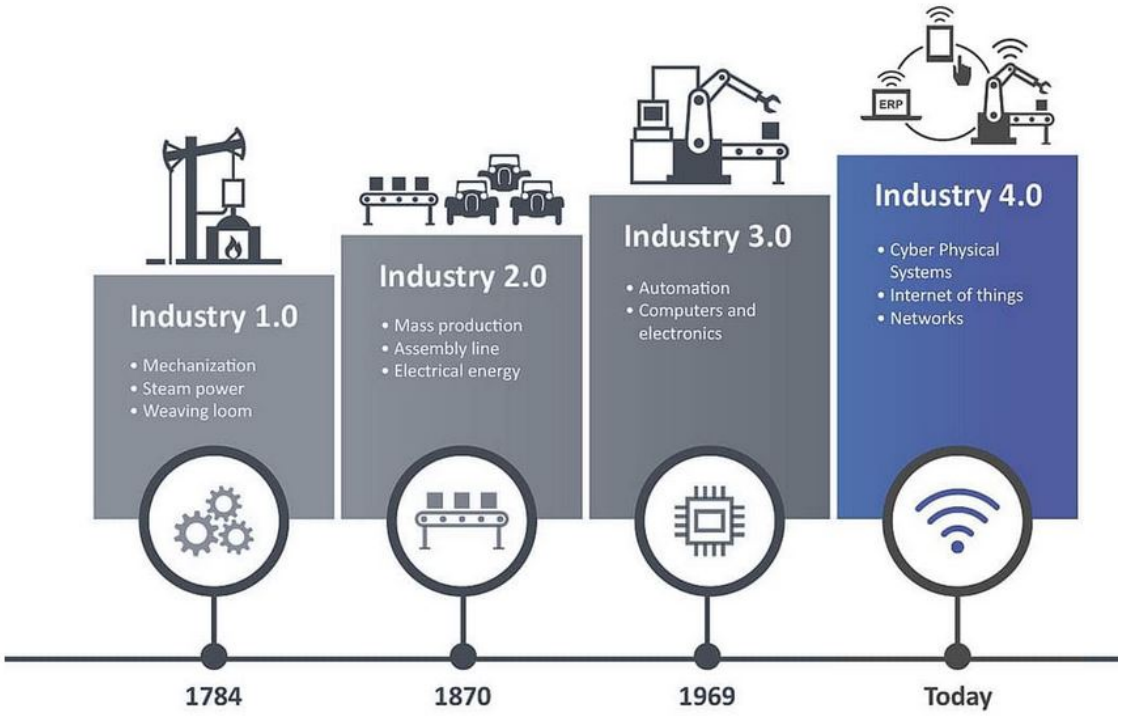

Figura 2.1: Revoluções da indústria de [11]

Resumindo a estratégia alemã da I4.0, está focada no desenvolvimento da manufactura inteligente recorrendo ao uso da tecnologia dos CPS, descentralização da produção, produtos customizados, sustentabilidade e aumentar tanto a acessibilidade, como o número de utilizadores.

2.2 Paradigmas

Segundo S. J. Hu e colegas [13] e A. Azwan Abdul Rahman [14], de forma a sustentar a competitividade em mercados dinâmicos, são necessários novos designs de sistemas de produção. Durante os últimos dois séculos a produção industrial tem revolvido entre vários paradigmas de produção. O primeiro paradigma foi o de produção artesanal, em que o produto era requisitado pelo cliente diretamente e criado através de métodos manuais. Este tipo de produção estava confinado a pequenas áreas geográficas e não era escalável. Passado pouco mais de um século, o aparecimento de linhas de montagem deu início à produção em massa, que possibilitou a produção em larga escala de produtos. Este sistema de produção era denominado de linha de produção dedicada. Nas últimas décadas a competição global, a alta procura por produtos variados e personalizados, originaram novos e complexos desafios de customização em massa, exigindo novas formas de produção (Figura 2.2).

Portanto, são necessárias novas formas de negócio apoiados na colaboração entre fornecedores e clientes, onde os paradigmas mais antigos não conseguem responder aos

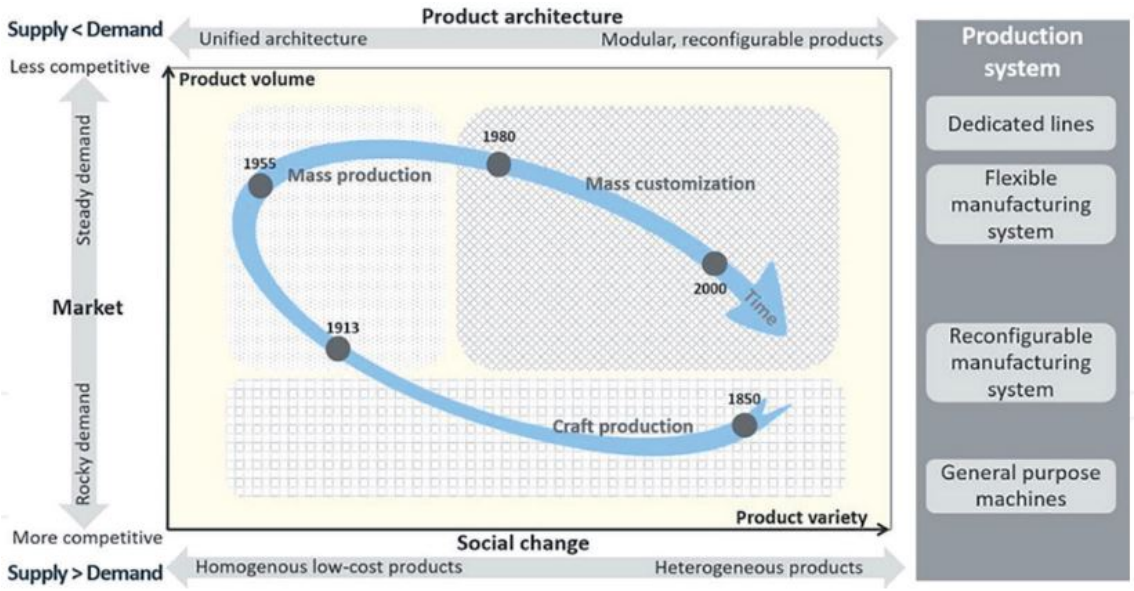

Figura 2.2: Revolução dos sistemas de produção de [14]

novos desafios [15]. Alguns paradigmas recentes são capazes de lidar com estes requisitos e fornecem tanto flexibilidade, como adaptabilidade aos sistemas de produção. Alguns desses novos paradigmas são discutidos nas próximas subsecções.

2.2.1 Sistemas Manufactura Flexível

As primeiras definições de flexibilidade na manufactura são baseadas na noção da adaptação a incertezas. Flexibilidade pode ser vista como a capacidade do sistema mudar e assumir várias posições ou estados em resposta à mudança de requisitos com pequenas penalizações no tempo, custo e performance [16, 17].

Nos Sistemas de Manufactura Flexível (Flexible Manufacturing Systems (FMS)) as companhias predefinem um certo número de processos de produção dentro dos limites das capacidades do sistema. Com só uma configuração, é possível as companhias ativarem de forma rápida variados modelos de produtos, ou seja, os recursos de hardware são os mesmos, mas o software que gere os processos é programável, possibilitando a integração de várias ferramentas para gerar produtos diferentes.

No entanto, quando requisitos inesperados aumentam, a adaptabilidade dos FMS é constrangida por limitações e por problemas de sincronização. Os FMS não estão preparados para alterações estruturais e como consequência, não conseguem responder a flutuações abruptas do mercado [14].

2.2.2 Sistemas Manufactura Reconfiguráveis

Os Sistemas de Manufactura Reconfiguráveis (Reconfigurable Manufacturing Systems (RMS)) são vistos como um desafio visionário para empresas de manufatura. Eles usam

equipamento modular como blocos de construção para realizar as funcionalidades requisitadas do sistema para a produção. Os RMS são o resultado do elevado custo de reconfiguração dos FMS. Os RMS diferenciam-se das linhas de produção dedicada e FMS, pela sua capacidade de ajustar a estrutura do sistema, adaptação e escalabilidade a pedidos variados. Os ajustes estruturais podem ocorrer nos níveis do sistema, máquina ou ambos. RMS é um paradigma de sistemas de manufactura com bom custo-benefício quando é necessário a adaptação a frequentes mudanças. Reduz os custos através do design de um sistema de produção para todas as partes da estrutura, providenciando a flexibilização necessária a todas para a produção dos componentes em conjunto. Garantindo-lhe a habilidade de produzir uma grande variedade de produtos em ambientes de elevado performance [14, 16]. Para além disto, em RMS a integração de novas tecnologias e/ou funções é possível e rápido.

Além dos benefícios económicos, os recentes requisitos de sustentabilidade ambiental e de saúde a ter consciência na manufactura, a reconfiguração é vista como uma característica vital para empresas de produção [18].

Uma arquitetura existente que controla o chão de uma fábrica e se foca na reconfiguração rápida do sistema é a CoBASA[19] através do uso de multi-agentes.

2.2.3 Sistemas Manufactura Biológicos/Biónicos

Segundo A. Tharumarajah em [20], os organismos biológicos exibem comportamentos autónomos, com relações hierárquicas. Um bom exemplo e resumo é dado por A. Tharumarajah: "os órgãos de um ser vivo agem autonomamente enquanto coordenam as suas ações e mantêm estabilidade. Os órgãos, por sua vez, são constituídos por células e suportam as formas de vida a que pertencem. Os Sistemas Manufactura Biológicos/Biónicos (Bionic Manufacturing Systems (BMS)) [21] são inspirados de tais conceitos biológicos e propõem conceitos para executar propriedades essenciais de sistemas de manufactura" [20].

Estruturalmente, a célula é a unidade base que compreende todas as outras partes de um sistema biológico. As células são basicamente similares, mas têm funções diferentes e são capazes de várias operações. As células operam através da troca de informação química com o seu ambiente envolvente. É esta troca de informação química com outras células que vai fazer o comportamento dela própria e de outras alterar. A coordenação entre as células é executada através de enzimas. O outro nível de regulação é feito pelas hormonas que são segregadas pelas células e transportadas por fluidos corporais até outras partes onde exercem ações fisiológicas especificas. O objectivo principal de um grupo de células em cooperação é fazer os órgãos a que estão associadas trabalhar (ex. coração). Enquanto um conjunto de órgãos têm como objectivo de tornar o organismo vivo (ex. humano).

A. Tharumarajah diz que nas unidades de manufactura:

> *"...as células de trabalho são inseridas numa empresa como recursos internos de*

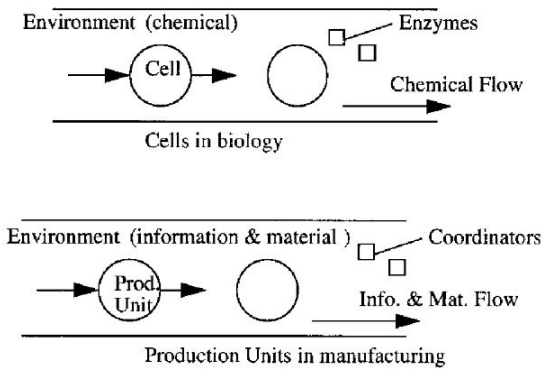

Figura 2.3: Semelhanças entre sistemas biológicos e de manufactura de [20]

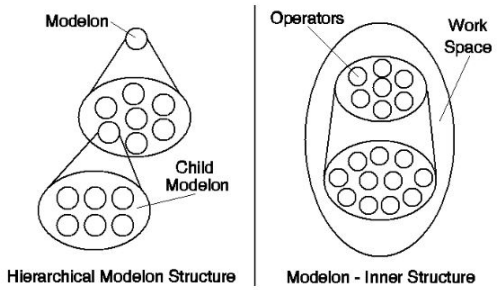

Figura 2.4: Estrutura de um modelon de [23]

> *produção (trabalhadores, máquinas, etc.) com capacidades distintas, mas, através de cooperação, capazes de alcançar o objectivo da unidade (produto intermediário e acabado) alterando estados se necessário. Matéria prima e informação de controlo circulam de forma predefinida através de interfaces. O, então, material processado e informação são enviados de novo pelos interfaces respectivos, até um ambiente onde se decide que direção devem tomar de destino (outra célula de trabalho, armazém, etc.). Unidades de coordenação e supervisão têm como missão coordenar e supervisionar o trabalho entre elementos internos das células e unidades no sistema de manufactura."* [22].

As propriedades expostas dos sistemas biológicos, apresentam várias semelhanças com a operação de unidades de manufactura (Figura 2.3).

Em adição às semelhanças operacionais, o processo de crescimento de formas de vida, também tem algum potencial de aplicação em sistemas de manufactura. Formas de vida são criadas através da repetição da divisão de células. Estas novas células desenvolvem-se em diferentes órgãos com funcionalidades distintas. Isto é possível através da informação de ADN que é replicada nas células divididas. Do mesmo modo, modelos de funções de manufactura podem ser criadas através do fornecimento de informação para desenvolver as funções requisitadas dos componentes divididos. Estas propriedades são usadas para propor conceitos e modelos de suporte. Um destes elementos principais é o modelon. Um modelon é composto por outros modelons de nível mais baixo (uma hierarquia de modelons filhos), operadores (acções enzimáticas), e memória comum ou ambiente (um espaço onde é trocada informação entre modelons) (Figura 2.4). A propagação de informação é feita através de um processo auto-organizado, onde modelons de níveis superior passam informação aos do nível inferior. Finalmente, entidades supervisoras ou operadoras (como enzimas) ficam responsáveis pela regulação e controlo das interações dos filhos e pais. Estes impõem regras de estruturação de modo a executar as tarefas e alcançar os objetivos do sistema [20, 22, 23].

2.2.4 Fábrica Fractal

Os conceitos de fractal têm origem na matemática e teoria do caos e indicam novas ideias para lidar com a inflexibilidade e rigidez das organizações atuais. Fábrica Fractal (Fractal Factory (FF)) é um sistema aberto e a sua principal característica é semelhança entre os seus pequenos componentes, conhecidos como entidades ou unidades fractal. Uma unidade fractal possui as seguintes características [20, 22, 23]:

- **Auto-organização** - não necessita de intervenção exterior para se reorganizar. Cada unidade arranja a sua estrutura interna baseando-se em critérios previamente atribuídos;
- **Similaridade** - uma unidade fractal é idêntica a outras fractais, mas a sua estrutura interna pode ser diferente.
- **Auto-otimização** - continuamente procura pela melhor performance.

Fractais agem como unidades independentes para atingir os seus objetivos, enquanto resolvem conflitos através de cooperação e o processo é iterativo à medida que alterações são impostas. No entanto, para que o objectivo principal do sistema seja atingido, a coerência do objectivo deve ser mantida através da cooperação e interação com outras unidades. Este sistema de objectivo funciona através da coordenação de fractais que ocupam ambos os níveis hierárquicos adjacentes ou o seu próprio. Numa FF uma organização predefinida não existe. Qualquer unidade fractal possui os seus próprios recursos com capacidades estáticas e um sistema de informação eficiente que fornece os dados necessários para a manufactura de produtos e alocação de recursos operacionais. Estas características criam um ambiente muito dinâmico dentro da unidade que torna possível o trabalho na presença de constantes mudanças na estrutura empresarial e reação rápida a requisitos exteriores.

2.2.5 Sistemas Manufatura Hólonicos

O paradigma de Sistemas de Manufatura Hólonicos (Holonic Manufacturing Systems (HMS)) é baseado nos conceitos estabelecidos por Arthur Koestler em 1967, ao tentar definir a natureza híbrida das estruturas de seres vivos e grupos sociais. O hólon é a base da teoria de HMS. Em [23], Christo e Cardeira dizem que:

> *"...na natureza, cada sistema tem as suas partes e faz parte de algo maior. Um hólon é um todo porque é constituído por subunidades (outros hólons) e ao mesmo tempo é uma subunidade de um sistema maior. As propriedades que se opõem no todo/parte são reflectidas nos atributos de autonomia e cooperação dos hólons. Todo o sistema pode ser considerado um hólon, desde a partícula ao universo. Por outras palavras, qualquer unidade num sistema é feita de unidades básicas e são parte de um todo."*

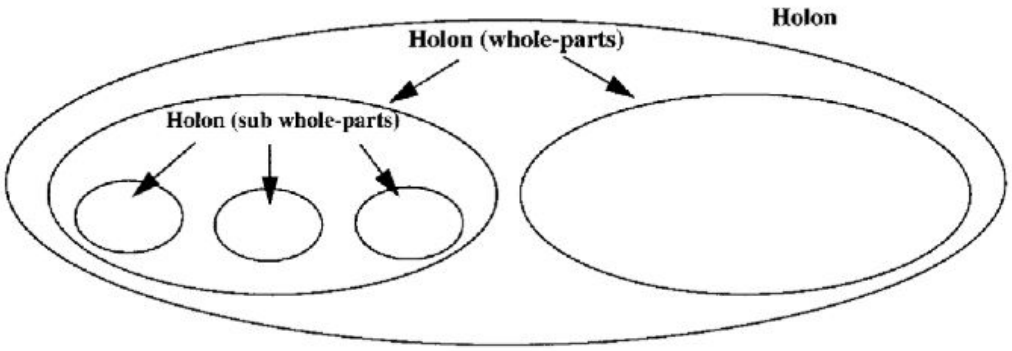

Figura 2.5: Sistema hólonico de [20]

Podemos ver uma representação deste sistema na Figura 2.5.

Criado sob o programa de investigação internacional de sistemas inteligentes de manufactura (IMS), o consórcio de HMS definiu alguns conceitos aplicados ao ambiente de manufactura [24]:

- **Hólon** – o bloco de construção de um sistema de manufactura, com comportamentos autónomo e cooperativo, capaz de transportar, armazenar e/ou validar informação e objetos físicos. Adicionalmente, um hólon pode conter parte de outro hólon;
- **Autonomia** – a entidade é capaz de criar e controlar a execução dos seus próprios planos e/ou estratégias;
- **Cooperação** – interação entre duas ou mais entidades em que se estabelecem a execução mútua de planos aceites;
- **Holarquia** – sistema hierarquicamente organizado de hólons autónomos que cooperam para um objectivo comum;
- **HMS** – uma holarquia que integra toda a atividade de manufactura, desde a ordem do pedido até produção e marketing para realizar a agilidade da empresa manufatora;
- **Atributos hólonicos** – atributos de uma entidade que a tornam num hólon. O mínimo é autonomia e cooperação.

Com estes conceitos, os HMS resultam em sistemas e subsistemas descentralizados e distribuídos [20–22, 24].

Baseado neste paradigma, existem múltiplas arquiteturas que foram sendo propostas, alguns exemplos são:
> Arquitetura PROSA[25] do inglês *Product-Resource-Order-Staff Architecture*, que utiliza três hólons na sua estrutura para controlar a logística (order holons), planeamento (product holons) e recursos (resource holons);

> Arquitetura ADACOR[26] de *ADAptive holonic COntrol Architecture*, uma arquitetura de controlo descentralizada, mas que considera também a centralização de modo a aumentar a otimização global do sistema. Nesta arquitetura cada hólon representa um recurso físico industrial ou entidade logística (Product Holons (ProdH), Task Holons (TH), Operation Holons (OpH) and Supervisor Holons (SupH));
> Arquitetura ORCA[27]. Nesta arquitetura um hólon optimizador global controla outros hólons optimizadores locais de níveis inferiores. Um mecanismo de comutação, que permite hólons locais decidir, é accionado se for acontecer uma perturbação e proíbe a aplicação de um planeamento predefinido. Este comutador permite que a arquitetura se adapte a perturbações, mas sem forma de retorno;
> Arquitetura ADACOR2[28]. Uma evolução da arquitetura ADACOR com o objectivo de fazer o sistema evoluir dinamicamente para configurações descobertas, e não alternar apenas entre duas configurações estacionárias e transientes. O resto da arquitetura é idêntica a ADACOR;
> Arquitetura POLLUX[29]. Esta arquitetura foca-se no mecanismo de adaptação da arquitetura, utilizando parâmetros de governação que podem alargar ou constringir o comportamento de hólons de níveis inferiores em relação a perturbações observadas por níveis superiores. Na POLLUX o número de comutações, comparando com a arquitetura ORCA, é grande. Cada decisão de uma comutação é justificada pelo incremento de performance relativamente às outras.

Os fundamentos deste paradigma têm sido utilizados, na grande maioria, como base de sistemas de multi-agentes (MAS).

2.2.6 Sistemas de Produção Evolutivos

O paradigma dos Sistemas de Produção Evolutivos (Evolvable Production Systems (EPS)) é baseado nos princípios dos HMS e BMS e é visto como uma visão alargada de sistemas evolutivos de montagem (EAS - Evolvable Assembly Systems), oferecendo ambientes de manufactura auto-organizados e adaptáveis. Os EPS usam sistemas complexos na natureza como metáfora para a sua contínua necessidade de adaptação a ambientes em constante mudança. No sentido biológico, o desenvolvimento é um processo relativamente rápido, que permite organismos individuais de fazer pequenas adaptações dependendo das suas condições. A evolução, por outro lado, refere-se a modificações mais significativas num espaço de tempo maior, à capacidade de evoluir com o ambiente, de geração em geração. Neste sentido, a evolução apoia-se na habilidade de sistemas complexos sofrerem alterações de relevância variadas, desde adaptações pequenas a transformações mais importantes.

Nos EPS a modularidade num nível de granularidade fina, a interligação entre módulos e uma aproximação de controlo baseados no paradigma de multi-agentes é fundamental. Ou seja, se numa linha de montagem os vários componentes que a compõem puderem ser ligados e desligados, podemos dizer que possui uma granularidade fina, caso apenas

os módulos se possam desligar ou ligar, então são de granularidade mais grossa. Existem dois princípios fundamentais em EPS [15, 30]:

1. O design do produto mais inovativo só pode ser alcançado se não se verificarem constrangimentos no processo de produção. O procedimento de seleção do processo subsequente, totalmente independente, pode então resultar numa metodologia do sistema de produção ótima;

2. Sistemas sobre condições dinâmicas precisam de evoluir, ou seja, precisam de ter a capacidade inerente de evoluir para aceitar os novos ou diferentes conjuntos de requisitos.

De modo similar ao paradigma HMS, o atual paradigma EPS providencia a base para algumas arquiteturas de multi-agentes como o caso da arquitetura IDEAS [31] que se inspira em sistemas evolutivos para auto-organização.

2.3 Sistemas Ciber-Físicos

Denomina-se de Sistemas Ciber-Físicos, sistemas computacionais colaborativos que representam e estão interligados com processos decorrentes no mundo físico. Computadores e redes embutidas, controlam e monitorizam os processos físicos, normalmente com ciclos de análise e resposta, onde os processos físicos afetam as computações e vice versa [32]. Como já foi dito antes, os CPSs são atualmente vistos como um pilar importante na indústria 4.0, mas na forma de Sistemas Ciber-Físicos de Produção (CPPS) e são o motivo de várias investigações para a automação em fábricas mundialmente [5].

CPPS são, à semelhança do CPSs, elementos autónomos, cooperativos e sub-sistemas que estão interligados aos vários níveis da produção. Estes processos abrangem desde o nível das máquinas até aos níveis mais altos da produção como as redes logísticas ou administração. A pirâmide de automação tradicional numa fábrica é constituída por cinco níveis hierárquicos: nível do campo/chão da fábrica (base da pirâmide); nível de controlo de processos; supervisão; gestão industrial; administração e planeamento empresarial.

Os CPPSs parcialmente partem da pirâmide da automação tradicional. Os níveis de controlo e do chão da fábrica/campo ainda existem e incluem os Controladores Lógicos Programáveis (PLC)s e Controladores Proporcional Integral Derivativo (PID)s perto dos processos, de modo a garantir o melhor desempenho dos ciclos de controlo, enquanto nos restantes níveis da hierarquia é adotada uma forma mais descentralizada e distribuída no seu funcionamento, uma qualidade que diferencia os CPPSs.

Os CPPSs são ainda uma tecnologia relativamente recente, e é essencial definir estruturas e metodologias como guias para a sua implementação.

J. Lee, Bagheri, and Kao [33] apresentam uma estrutura com cinco níveis que fornece uma orientação de todos os passos para o desenvolvimento de um CPPS, denominada

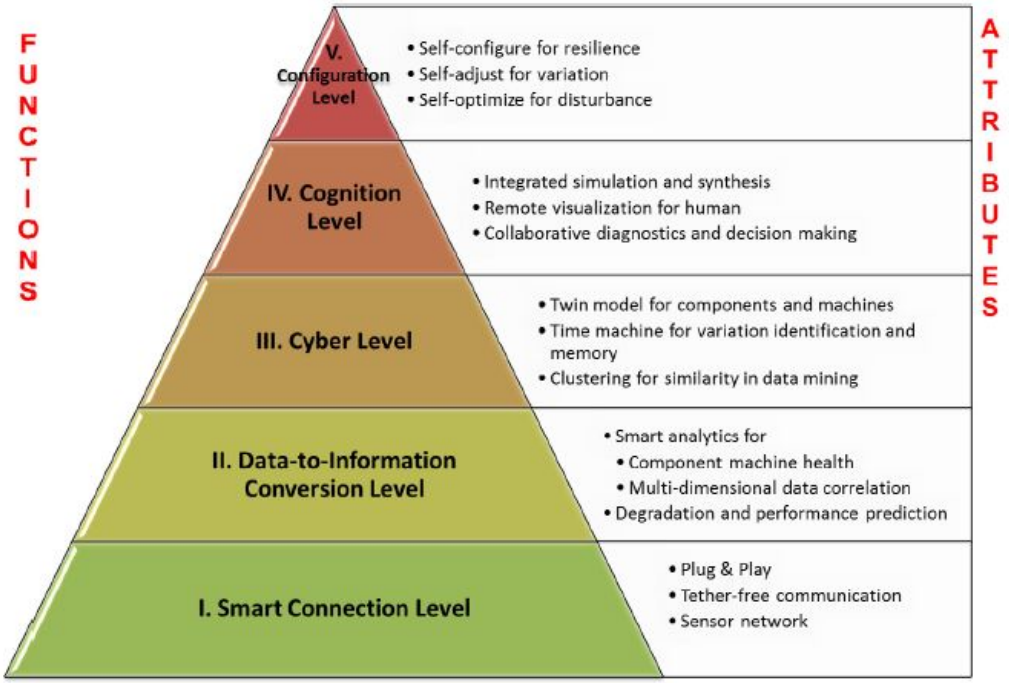

Figura 2.6: Arquitetura 5C para construção de CPPS de [33]

de arquitetura 5C (Figura 2.6). Esta arquitetura define um plano de trabalho para a construção do CPPS desde a aquisição de dados até à criação de valor.

O plano de trabalho começa no primeiro nível mais abaixo e sobe progressivamente até ao quinto nível no topo da pirâmide (Figura 2.6):

1. **Smart Connection** - A aquisição de dados corretos e precisos é o primeiro passo. Dois factores importantes a ter em consideração são: considerando os vários tipos de dados, é necessário um método fiável para gerir a sua aquisição e transferência para o servidor central; seleção dos sensores apropriados para o tipo de trabalho e especificidade da máquina.

2. **Data-to-Information Conversion** - Dos dados adquiridos, no primeiro nível, temos de extrair informação útil. Para além disto, podemos ainda, através de vários cálculos, fazer com que as máquinas possam realizar autonomamente um prognóstico do seu estado.

3. **Cyber Level** - Serve como ponto central para todos os dados de todas as máquinas. Extrapola mais informações desses dados para comparar e prever o comportamento das máquinas.

4. **Cognition Level** - Neste nível é criado conhecimento de todo o sistema e é necessária uma correta apresentação desse conhecimento adquirido para suportar decisões a ser executadas por utilizadores especializados.

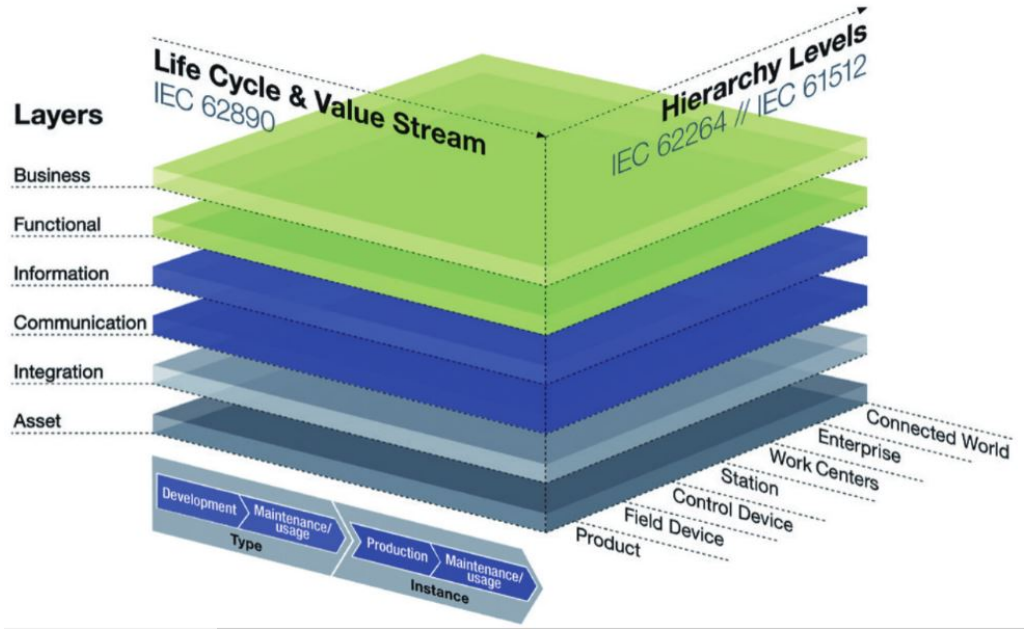

Figura 2.7: Modelo para arquitetura de referência para I4.0 (RAMI 4.0) de [36]

5. **Configuration Level** - O nível de configuração recebe informação dos espaços cibernéticos aos físicos e atua como controlo supervisionado, para embutir configuração e adaptação automática nas máquinas. Este patamar também age como um sistema de controle de resiliência, que aplica as decisões corretivas e preventivas tomadas no nível cognitivo ao sistema monitorizado.

Outra metodologia, com o propósito de criar uma visão comum em como abordar o desenvolvimento de CPPS, é conhecida como RAMI 4.0.

RAMI 4.0 é uma abreviação de Reference Architectural Model for Industrie 4.0, que significa Modelo de Arquitetura de Referência para indústria 4.0. Tal como a arquitetura 5C, a RAMI 4.0 fornece uma estrutura de como desenvolver um sistema distribuído. Esta possui três eixos coordenados que representam todos os aspectos cruciais da I4.0, subdividindo-os em grupos mais pequenos, todos interligados (Figura 2.7)[34–36].

Os três eixos contidos na RAMI 4.0 são:

- **Hierarchy Levels** - níveis hierárquicos, baseado na norma IEC 62264 do padrão internacional para integração de sistemas de controlo. Estes níveis representam todos os componentes presentes no ambiente industrial, com inclusão dos produtos e a conexão com serviços e IoT, rotulado de "Connected World".

- **Life Cycle & Value Stream** - representa o ciclo de vida dos produtos e instalações, baseado na norma IEC 62890 que define os princípios básicos para gestão do ciclo de vida dos sistemas e componentes usados nos processos industriais de medição, controlo e automação.

- **Layers** - As seis camadas descrevem a decomposição da máquina às suas propriedades, representando vários níveis de abstração:

    1. **Asset Layer** - Esta camada representa todos os componentes físicos no mundo real;
    2. **Integration Layer** - Nesta camada é onde se encontra a "digitalização"de todos os componentes presentes na camada anterior (Asset Layer);
    3. **Communication Layer** - Esta camada é responsável pela comunicação entre as duas camadas vizinhas, Integration e Information, através de protocolos de comunicação padrão;
    4. **Information Layer** - Adquire informação de todo o sistema, garantindo a integridade de todos os tipos de dados;
    5. **Functions Layer** - Responsável pelo controlo do sistema através de funções e regras;
    6. **Business Layer** - Define e altera as regras que o sistema tem de seguir, mostra o comportamento do sistema em tempo real, gere recursos e modelos de negócio.

3 Conceitos de Suporte

Este capítulo tem como objectivo dar contextualização e apresentar alguns dos conceitos mais importantes que suportam este trabalho. Designadamente, agentes, sistemas multi-agente, auto-organização e otimização através de algoritmos com origem em comportamentos biológicos.

3.1 Sistemas Multi-Agente

A definição de agente não é consensual e existem diversas definições, no entanto podemos definir uma descrição que respeite a maioria. Em [37] encontramos a definição de que um agente é uma entidade independente capaz de resolver problemas (implementada em hardware, software ou uma mistura dos dois) que respeitam as seguintes propriedades:

- **Autonomia** - os agentes devem conseguir realizar a maioria das suas tarefas sem a intervenção direta de outros agentes ou humanos. Devem também possuir um certo grau de controlo sobre as suas próprias ações e estado;

- **Sociabilidade** - os agentes devem ser capazes de interagir, quando acharem apropriado, com outros agentes e humanos de modo a resolverem o seu problema e ajudar os outros nas suas atividades. Isto implica que os agentes tenham, no mínimo, uma maneira de comunicar os seus requisitos a outros e também um mecanismo interno que decida sobre as apropriadas interações sociais (tanto a iniciar como a responder interações);

- **Responsividade** - os agentes devem percepcionar o seu ambiente envolvente (que pode ser o mundo físico, um utilizador, grupo de agentes, a Internet, entre outros. . .) e responder de forma célere a alterações que possam vir a ocorrer-lhe;

- **Proatividade** - os agentes não devem simplesmente agir em resposta ao seu ambiente, devem exibir comportamentos oportunos, direcionados ao objectivo e tomar iniciativa, quando apropriado.

Em adição a estas condições necessárias, são propostas outras características desejáveis. Estas incluem:

- **Adaptabilidade** - a habilidade de um agente modificar o seu comportamento ao longo do tempo, em resposta a alterações de condições no seu ambiente, ou pela obtenção de conhecimento sobre a resolução do seu problema;
- **Mobilidade** - a habilidade de o agente mudar a sua localização física de modo a melhorar a sua resolução do problema;
- **Veracidade** - a assunção de que o agente não comunica de propósito informação falsa, ou seja, não mente;
- **Racionalidade** - a assunção de que o agente agirá de forma a atingir o seu objectivo e não o contrário em que o previne/evita sem razão.

Um sistema com vários agentes ou também conhecido como sistema multi-agente (MAS) deriva do campo da inteligência artificial distribuída, sendo caracterizada pela descentralização e execução paralela de atividades através de agentes [38]. O MAS oferece uma alternativa no design de sistemas complexos baseados na descentralização de funções, providenciando modularidade, flexibilidade, robustez, adaptabilidade e reconfigurabilidade. Especialmente no cenário da indústria, o aparecimento de agentes industriais focam-se na introdução da inteligência artificial que pode ser aplicada na automação de dispositivos, sistemas e infraestruturas, contribuindo efetivamente para a criação e interação de sistemas e componentes ciber-físicos [9, 38, 39].

3.2 Auto-Organização

Ideias modernas sobre auto-organização começaram com a fundação cibernética nos anos 1940s. W. Ross Ashby, H. von Foerster e N. Wiener, entre outros, contribuíram para uma primeira compreensão. Mais tarde, o conceito foi adotado em física e atualmente está presente na maioria das ciências naturais. Muitos sistemas têm vindo a ser identificados com possuindo aspectos de auto-organização, embora uma clara definição ainda esteja em falta [10].

Segundo Heylighen em [40], *"auto-organização significa que uma estrutura funcional aparece e se mantenha espontaneamente. O controlo necessário para concretizar isto deve ser distribuído através de todos os componentes que participam. Se fosse centralizado num módulo ou subsistema, então, em princípio, com a remoção deste módulo o sistema perderia a sua organização"*. Estes sistemas são intrinsecamente robustos e capazes de amortecer a deterioração dos sistemas face a erros criticos. Adaptando-se e aprendendo a lidar com futuros problemas.

Um aspecto importante na engenharia de CPS, é ter em consideração os vários comportamentos sociais e biológicos para desenvolver auto adaptação e evolução em sistemas complexos. A biologia oferece uma enorme quantidade destes mecanismos, simples, mas poderosos, para lidar com ambientes complexos. A ideia é levar estes mecanismos a

resolver problemas complexos existentes na manufactura. Neste tipo de ambientes, os comportamentos e padrões resultantes são mais complexos que a soma dos comportamentos individuais. A complexidade também advém da sensibilidade às condições iniciais, conhecido como efeito borboleta, e das interações não lineares que envolvem cooperação. Neste sentido, o comportamento emergente é difícil de prever, tornando-se importante garantir que comportamentos ou propriedades indesejadas e imprevistas ocorram. Também é importante controlar o nervosismo do sistema para evitar que ele trabalhe num estado caótico [39].

Podemos extrapolar da maioria das definições algumas características comuns de auto-organização, são elas [41–43]:

- **Sem controlo externo explícito** - Uma propriedade imprescindível nestes sistemas auto-organizados que devem ser independentes de qualquer controlo externo para se organizar. Ou seja, o sistema deve ser capaz de se organizar sem recorrer a qualquer interferência exterior e baseando-se apenas em decisões internas;

- **Controlo distribuído** - Num sistema auto-organizado baseado na tecnologia MAS, o controlo é distribuído por todo o sistema, com todas as partes a contribuir uniformemente para a organização do sistema. Cada indivíduo cinge-se a comportamentos simples e possui uma visão limitada pela natureza local das suas interações. No entanto esta propriedade não é de todo mandatória, visto que podemos encontrar sistemas com a presença de um controlo central;

- **Ordem global através de interações locais** - A habilidade de chegar a uma ordem global (estável) através do emergir das interações de dentro do sistema. Todas as entidades que compõem estes sistemas, estão correlacionadas entre si, onde algumas, através de propagação local, impõem a reconfiguração aos seus vizinhos, que por sua vez, forçam também a reconfiguração nos seus vizinhos e por aí adiante. Resultando na expansão da ordem local até à global. Geralmente este fenómeno emergente resulta destas interações locais, possibilitando o sistema de operar sem qualquer controlo central;

- **Robustez** - Como já foi mencionado acima, os sistemas auto-organizados são robustos, no sentido em que a ordem global é relativamente pouco sensível a perturbações e erros. Tal resiliência deve-se à redundância da organização distribuída do sistema. Para além de o comportamento se degradar de forma não abrupta, certos erros e perturbações podem ter um impacto favorável na sua organização;

- **Múltiplos equilíbrios** - Sistemas não lineares são tipicamente caracterizados por terem mais do que uma solução (estados estáveis de equilíbrio), portanto existe um conjunto de configurações para onde o sistema pode convergir. Estes estados podem ser máximos locais ou o máximo global;

- **Dinâmica longe do equilíbrio** - Um sistema auto-organizado é um processo dinâmico, onde o decorrer do tempo apresenta um incremento na sua ordem. Sistemas longe do equilíbrio são mais susceptíveis a alterações no ambiente, no entanto conseguem ser mais dinâmicos e capazes de reagir. Isto proporciona ao sistema a capacidade de produzir uma grande variedade de regulações com diversas intensidades, levando o sistema a diversas configurações estáveis. Tipicamente, o equilíbrio do sistema resulta na interação entre respostas positivas e negativas;

- **Não-linearidade** - Na maioria dos sistemas tradicionais baseados em métodos matemáticos, os efeitos nos sistemas são proporcionais às suas causas. Nos sistemas auto-organizados tal relação linear entre causa e efeito não existe. Pequenas flutuações podem implicar grandes variações no comportamento geral, enquanto grandes flutuações podem nem ter quase nenhumas repercussões. Esta característica promove a complexidade, onde o comportamento geral não é simplesmente subentendido pela observação de cada um dos componentes individuais.

3.3 Algoritmos Bio-Inspirados

A pesquisa em algoritmos bio-inspirados tem sido desenvolvida em várias áreas à bem mais de meio século. Existem algoritmos para vários propósitos e não têm todos o mesmo objectivo. Cada vez se tem usado mais estes algoritmos para resolver problemas complexos em engenharia e a manufactura é um deles. Muitos deles são usados para resolver problemas de otimização, mas são capazes de muito mais do que isso. Existem centenas de algoritmos de otimização, no entanto, devido á natureza estocástica da maioria dos problemas na manufactura e o facto de nos apoiarmos na tecnologia de multi-agentes, os algoritmos que se utilizaram serão baseados em sistemas colectivos. Estes são algoritmos de otimização estocástica que através da cooperação entre entidades de uma população, são capazes de explorar e definir soluções, até encontrar uma solução que seja ótima.

A natureza é extremamente abundante em sistemas colectivos naturais, composto por várias entidades com comportamentos interativos muito simples e que em grupos acabam por dar origem a comportamentos muito complexos e adaptáveis. Estes comportamentos de variadas espécies têm sido alvo de inspiração para o desenvolvimento de inúmeros algoritmos bio-inspirados. Existem muitos algoritmos de sistemas colectivos e mais recentemente têm começado a aparecer algoritmos híbridos, onde são combinados dois ou mais algoritmos para resolver o mesmo problema. Mas para evitar o acréscimo de complexidade, estes serão deixados de parte e apenas se utilizaram os normais. Alguns dos algoritmos que poderão vir a ser utilizados na implementação, entre outros conhecidos de otimização, são apresentados nas subsecções seguintes.

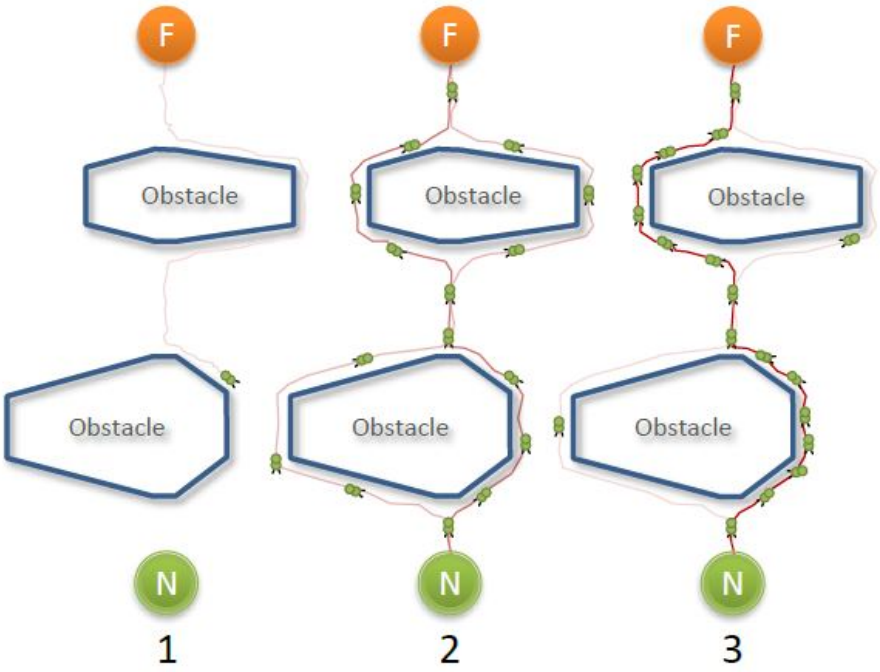

Figura 3.1: Caminho mais curto gerado através do ACO de [43]

3.3.1 Otimização Colónia de Formigas

A otimização da colónia de formigas (ACO) é uma meta-heurística capaz de resolver complexos problemas combinatórios, proposta por M. Dorigo em 1991[44]. Este algoritmo é inspirado na procura de alimento das formigas, que produz um rasto de feromonas eficiente até á comida. O algoritmo explora uma mecânica similar para resolver problemas de otimização através da implementação de uma heurística aleatória que faz decisões probabilísticas tendo em conta o rasto de feromonas e informações de heurísticas disponíveis[45]. O mecanismo que explica este processo de procura chama-se estigmetria[46]. A estigmetria é um mecanismo de comunicação indireto através de modificações no ambiente criado pelos indivíduos, neste caso feromonas. As formigas conseguem descobrir o caminho mais curto entre o seu ninho e a comida através da deposição de feromonas enquanto andam, criando um trilho de feromonas de variadas intensidades. O caminho escolhido é baseado numa decisão probabilística da intensidade das feromonas: quanto mais forte o trilho, maior a probabilidade de seguir esse caminho. Na figura 3.1 [43], depois da formiga descobrir alimento, começa a largar feromonas no caminho de volta ao ninho (1). Quando outras formigas detectam o trilho de feromonas, tendem a segui-lo conforme a concentração de feromonas. Consequentemente, o nível de concentração de feromonas aumenta no percurso mais curto por ser utilizado mais vezes (2). Essa atração das feromonas acaba por reforçar o trilho mais curto e surge o melhor caminho (3). A evaporação das feromonas garante a adaptação a eventuais mudanças, pelo facto de que o caminho só se mantém se for continuamente utilizado[43, 47].

O primeiro algoritmo de formigas, "Sistema de Formigas" (Ant System - AS), foi desenvolvido nos anos 90s por Dorigo e outros [48] e testado com êxito no benchmark conhecido como o Problema do caixeiro-viajante. A meta heurística ACO foi desenvolvida para generalizar o método de resolução de problemas combinatórios por aproximação de soluções, através do comportamento genérico das formigas. O ACO é constituído três funções principais[47]:

- ***AntSolutionsConstruct*** - esta função realiza o processo de construção da solução, onde as formigas artificiais andam através de estados adjacentes ao problema de acordo com uma regra de transição, construindo iterativamente soluções.
- ***Pheromone Update*** - faz as atualizações dos trilhos de feromonas. Pode atualizar os trilhos apenas quando tem soluções completas, ou então a cada iteração. Para além do reforço do trilho de feromonas, também inclui uma evaporação das feromonas no trilho. A evaporação faz com que as formigas se "esqueçam" daquele caminho e procurem novos.
- ***DeamonActions*** - um passo opcional no algoritmo, que envolve aplicar atualizações adicionais numa perspectiva global (que não existe no mundo natural). Este pode incluir, por exemplo, um reforço às feromonas da melhor solução gerada.

3.3.2 Otimização Colónia de Abelhas

O algoritmo de otimização da colónia de abelhas (BCO) é um algoritmo baseado em agentes proposto por Teodorovic para resolver problemas complexos de combinatória [49, 50].

No início todas as abelhas se encontram na colmeia. Durante o processo de procura, cada abelha realiza uma série de movimentos locais, construindo de forma incremental uma parcial solução. Às soluções parciais, são adicionadas outras componentes de soluções até se obter uma ou mais soluções satisfazíveis. Este processo repete-se um número de vezes pré-definido até se obterem algumas soluções parciais. Quando as abelhas voam pelo espaço de procura, realizam um caminho de ida e um caminho de volta. No caminho de ida elas constroem várias partes de soluções, através da combinação da exploração individual e informação colectiva passada. Depois realizam o caminho de volta para a colmeia. Dentro da colmeia, todas as abelhas participam nos processos de decisão. As abelhas batedoras (Scout bees) realizam uma dança em frente de toda a colónia, trocando informações da qualidade das soluções parciais encontradas. Baseando-se na qualidade gerada pelas soluções parciais, cada abelha decide se abandona a solução parcial criada e continua a construção da sua solução ou então dança e recruta outras abelhas para a solução parcial criada. Dependendo da qualidade da solução gerada, cada abelha tem um certo nível de lealdade ao percurso que guia para a solução parcial prévia. Novos caminhos de ida e volta são gerados e novos processos de decisão são feitos até se encontrar

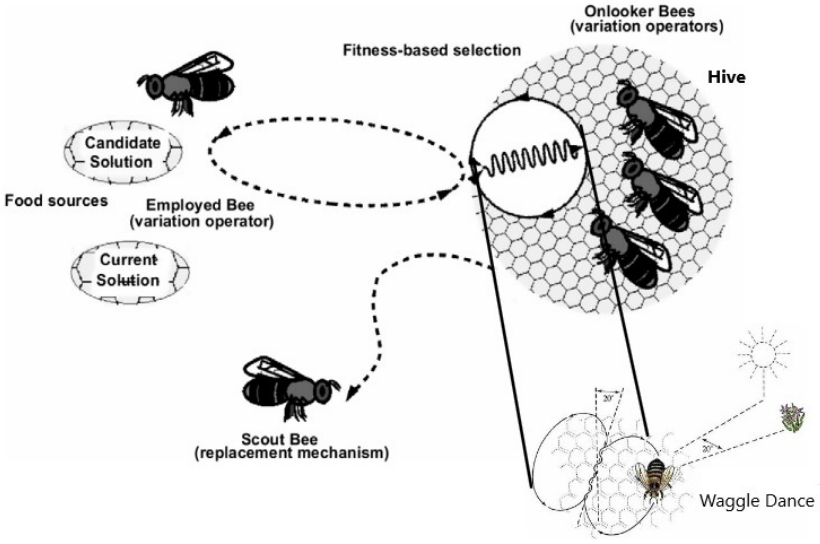

Figura 3.2: Esquema do funcionamento de BCO de [51]

uma ou mais soluções satisfazíveis, ou um critério de paragem for satisfeito (Figura 3.2) [43, 47].

3.3.3 Algoritmo Pirilampos

O algoritmo dos pirilampos (FA) foi proposto por Yang [52] e é uma heurística baseada em enxames para otimização de tarefas inspirada no piscar dos pirilampos. O algoritmo constitui um procedimento iterativo de população, com inúmeros agentes (os pirilampos) a resolver um dado problema de forma simultânea. Os agentes comunicam uns com os outros através do brilho bioluminescente, que lhes permite explorar com melhor eficácia o espaço de funções de custo comparativamente à procura aleatória distribuída. Este algoritmo de otimização baseia-se na assunção de que uma solução possa ser assumida na forma de um agente (pirilampo), que brilha proporcionalmente à qualidade da sua solução ao problema. Essa intensidade do brilho no piscar do pirilampo atrai outros pirilampos para essa solução, tornando a procura do espaço ainda mais eficiente (Figura 3.3)[47, 53].

Três características do piscar são:

1. Todos os pirilampos são unissexo e irão se deslocar na direção dos mais brilhantes/atraentes;

2. O grau de atracão de um pirilampo é proporcional ao seu brilho. O seu brilho também pode decrescer à medida que a distancia aumenta. Se não existir nenhum pirilampo mais brilhante, o pirilampo move-se de forma aleatória;

3. A intensidade do brilho no pirilampo é determinada pelo seu valor da função objectivo, que indica a qualidade da sua potencial solução.

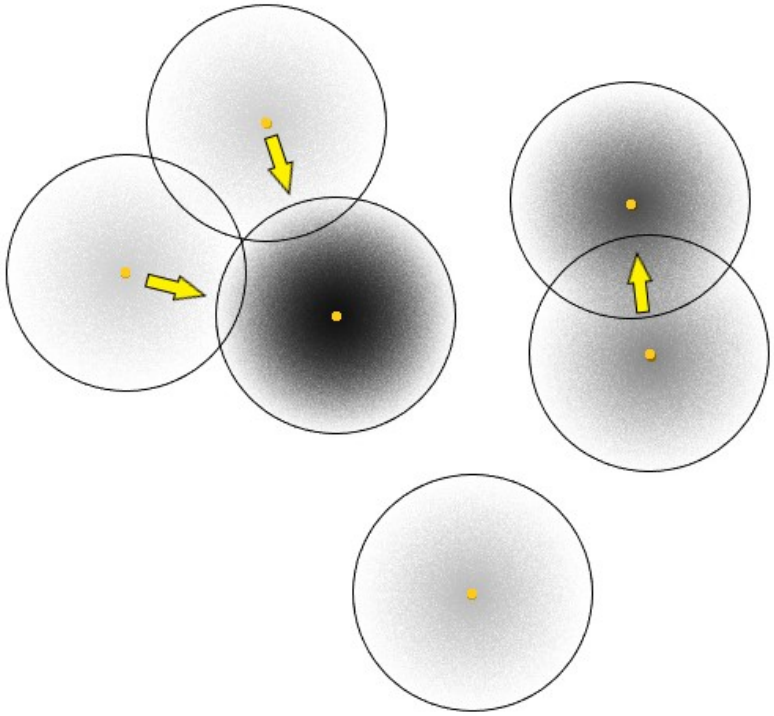

Figura 3.3: Exemplo do comportamento entre pirilampos no FA

Existem algumas aplicações na manufactura que usam este algoritmo, embora ainda seja relativamente recente. A arquitetura BIOSOARM [54] é um exemplo que replica o comportamento dos pirilampos para se auto-organizar. Onde as peças são atraídas para unidades de processamento num raio predefinido, através da requisição de transporte. O nível de atração destas unidades de processamento está dependente do número de peças que existem já lá armazenadas, mais o número de peças que se estão a deslocar para elas.

3.3.4 Algoritmo otimização de Partículas

O algoritmo de otimização das partículas (PSO) foi desenvolvido em 1995 por Kennedy e Eberhart [55], que se baseia no comportamento que se verifica em grupos de peixes e pássaros. Este algoritmo é um dos mais conhecidos e é o responsável pela criação e estudo de muitos novos trabalhos em torno da inteligência em grupo, ou mais conhecida como Swarm Intellegence.

Este algoritmo procura por soluções no espaço de estados através do ajuste das trajetórias de agentes individuais, chamados partículas, através da soma de vectores de posição de outras soluções. Cada partícula é atraída para a melhor posição global atual e a sua melhor posição até ao momento, introduzindo pequenas variações aleatórias. Na figura 3.4 podemos observar um exemplo de como uma partícula se movimenta.

A localização inicial das partículas deve ser distribuída uniformemente pelo espaço

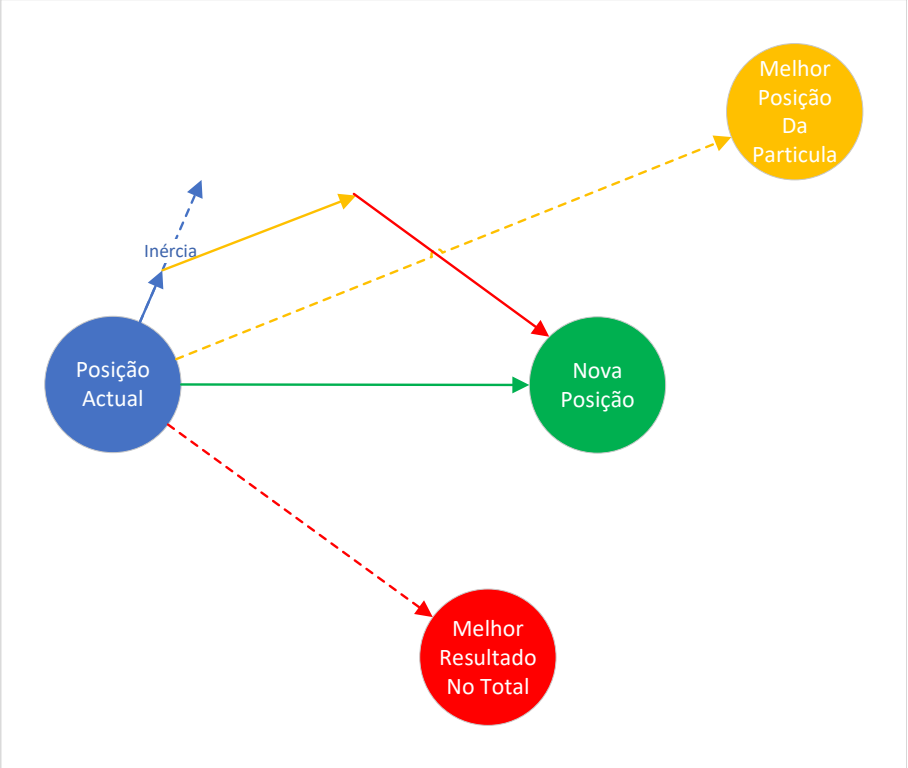

Figura 3.4: Exemplo do movimento de uma partícula em PSO

de estados, para que a procura abranja o maior espaço possível, isto é principalmente relevante para problemas multi-modais [56].

Existem várias aplicações na manufactura com MAS, que usam estes tipos de algoritmos para resolver problemas complexos. O algoritmo ACO é utilizado por W. Xiang e H.P.Lee em [57], para orientar os agentes de produção na realização das suas tarefas para os devidos recursos, através do uso das feromonas e garantir que estes cumprem a sua devida ordem de execução. Os algoritmos BCO em [58], por c.Chong e outros, e o PSO em [59], por Tsung-Lieh Lin e outros, são ambos utilizados para optimizar a designação de tarefas entre vários recursos, recorrendo aos modelos de procura das abelhas e conjuntos de partículas, respectivamente. Por último, para o FA, um exemplo é o BIOSOARM [54], que já foi apresentado antes.

Existem muitas mais aplicações e abordagens com estes e outros algoritmos na manufactura, mas em todos eles, os algoritmos que se podem utilizar estão limitados e/ou são impossíveis de se alterar sem refazer a arquitetura de novo.

4 Arquitectura

Neste capítulo apresentamos uma versão preliminar de uma possível abordagem a uma arquitetura que soluciona o problema colocado no inicio do documento. Esta abordagem irá recorrer das tecnologias e alguns conceitos que foram previamente descritos

4.1 Visão Geral da Arquitetura

A arquitetura que se propõem é constituída por três camadas. A camada inferior representa o chão da fábrica e todos os sistemas e equipamentos que lá se encontram. As outras duas camadas são constituídas por um MAS e pelo algoritmo de otimização instanciado.

A camada central é a principal e mais importante na arquitetura e onde se encontra o MAS. É nesta camada que, através de sensores e interfaces, se sabe o estado atual da linha de produção, ou seja, que estações e produtos existem, quantas são, onde estão, o que fazem e, no caso de ser um produto, o que querem fazer ou qual o seu objectivo. Esta camada encontra-se em permanente comunicação com o chão da fábrica, devido às constantes trocas de informação entre os agentes e os seus respectivos sistemas físicos. É também aqui que se encontram armazenados os melhores percursos que os produtos deverão seguir de modo a executar as suas ações(*skills*), até atingirem o seu objectivo. Estes caminhos encontram-se na forma mais optimizada descoberta até ao momento e são o resultado da interação desta camada com a seguinte, acima desta.

Acima do MAS, encontramos a camada que possui os algoritmos de otimização e que confere neste trabalho a particularidade da flexibilidade na escolha de algoritmos. Esta camada é responsável pela constante otimização do sistema. Os algoritmos acedem ao estado atual do sistema, recorrendo à camada inferior, e calculam quais os melhores percursos para uma dada ação a executar pelos agentes de produtos presentes naquela camada. Estes caminhos depois de calculados, são depois transferidos e armazenados na camada inferior para que os agentes lhes possam aceder.

De forma a ajudar a compreender a estrutura e comportamento da arquitetura que se pretende desenvolver, são exibidas as duas seguintes Figuras 4.1 e 4.2. Na Figura 4.1 podemos observar um diagrama da arquitetura geral do sistema, onde se encontram representadas as três camadas de que é composta e alguns exemplos dos dados genéricos que são transferidos entre elas. A Figura 4.2, mostra um diagrama de sequência com as

interações generalizadas entre as camadas que compõem a arquitetura, ao longo da sua execução.

Ao longo do resto deste capítulo, entraremos mais em detalhe sobre as entidades, interações e comunicações entre e dentro de cada camada. No entanto, antes disso, definiremos alguns conceitos a ter em consideração, de forma a entender melhor o funcionamento da arquitetura.

4.2 Definição de *Skill*

Como foi dito antes e se pode verificar na Figura 4.1, existem dois agentes que representam os recursos (RA) e os produtos (PA). Os PA's possuem um plano de ações (*Skills*) a serem realizadas sobre o produto a que estão associados para que se obtenha o produto final desejado. Os RA's, por outro lado, contêm a informação das *Skills* que estes conseguem executar sobre determinados produtos. Estas *Skills* só podem ser executadas pelos recursos (RA) e um recurso poderá executar várias *Skills* diferentes [60].

Quando é introduzido um produto no sistema, ou seja um PA, este vai requisitar a execução ordenada de várias *Skills* para que ele chegue ao produto desejado. O PA desloca-se e requisita então a um RA capaz de executar a *Skill* em questão, e aguarda pela confirmação e execução da mesma pelo RA. Após receber o produto processado, o PA remove a *Skill* do seu plano e passa à seguinte, repetindo o mesmo processo até chegar ao fim.

4.3 Plug and Produce

Esta arquitetura também possui a capacidade de que os vários recursos possam ser removidos, ou acrescentados outros, durante a execução. Os RA's e PA's são uma representação de um componente físico e existem tantos quantos os presentes no mundo real, com uma relação de um para um, ou seja, cada RA ou PA está associado a um único componente.

O facto de o sistema ser composto por módulos independentes, para além de fornecer flexibilidade e reconfigurabilidade ao sistema de produção, confere-lhe também robustez face a erros e avarias, por exemplo, se algum recurso for removido ou por alguma razão sofrer avaria, o agente do produto irá procurar por outro recurso para onde se dirigir e realizar a *skill* requerida.

4.4 Arquitectura MAS

Como foi dito anteriormente, o MAS é responsável pelo controlo lógico de alto-nível e pela abstração de todos os componentes que compõem o sistema de produção. Normalmente verifica-se que arquiteturas para sistemas de produção são construídas de raiz já com um algoritmo em mente e são extremamente inflexíveis caso se o pretenda alterar. Esta arquitetura tem a particularidade da otimização se realizar fora da camada onde

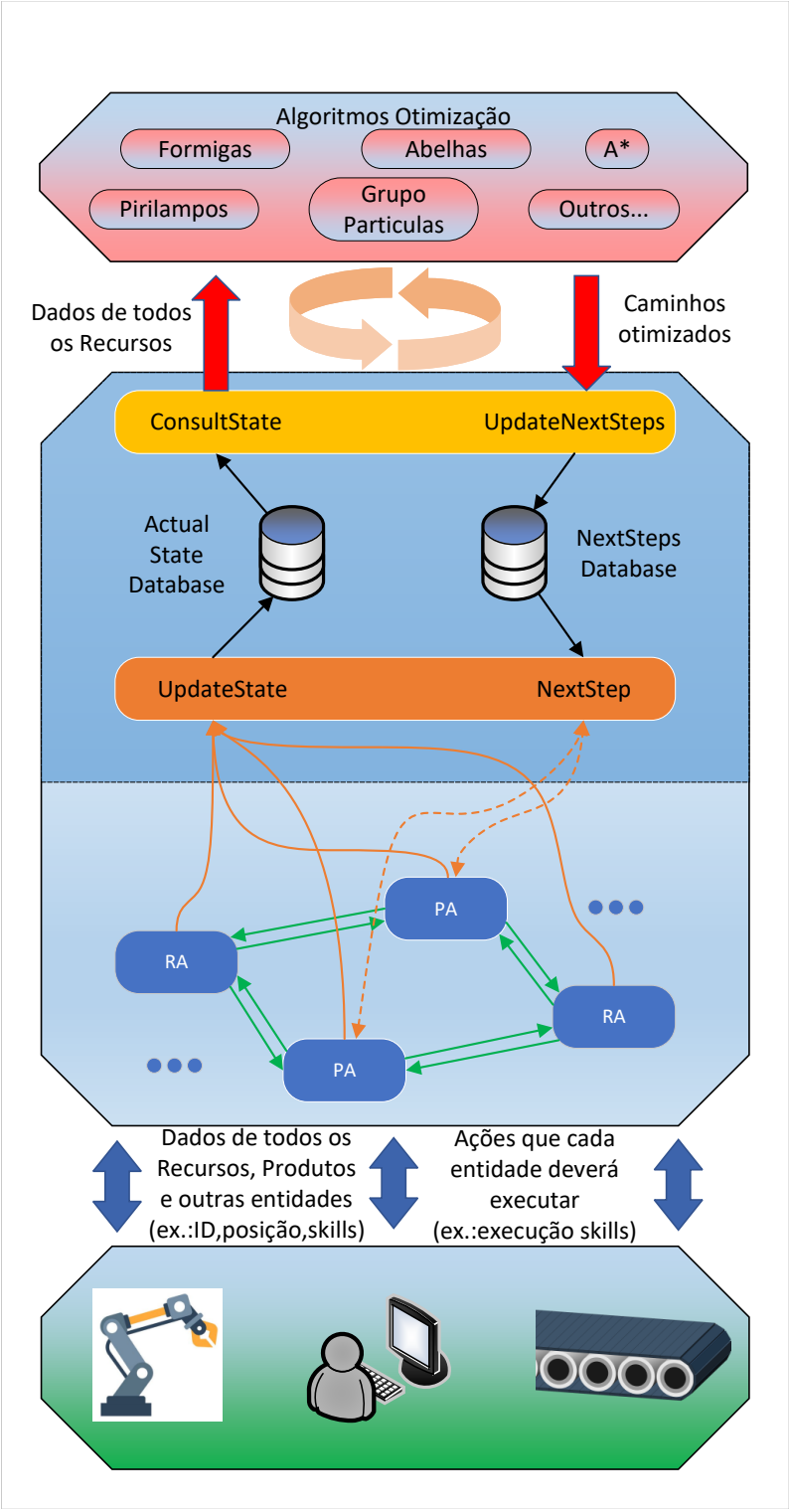

Figura 4.1: Diagrama geral da arquitetura proposta

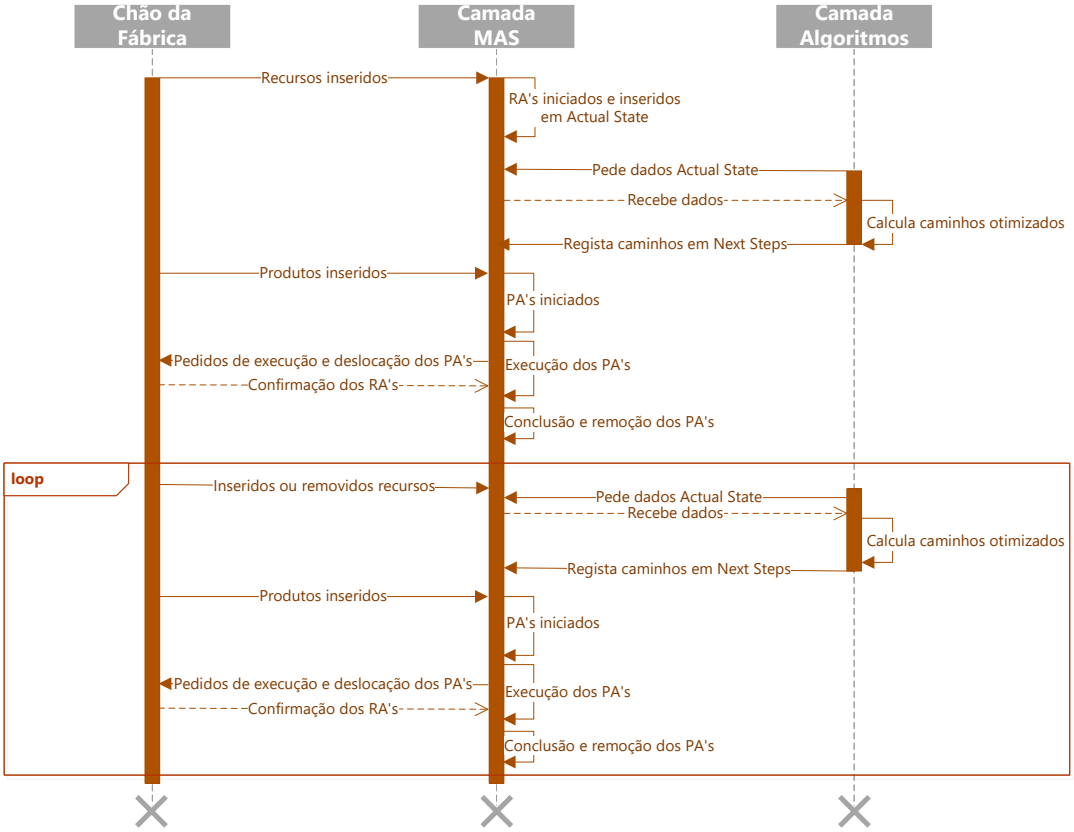

Figura 4.2: Diagrama de sequência representando as interações entre as camadas da arquitetura

se encontra o MAS, dando oportunidade de se poder utilizar diferentes algoritmos de otimização.

O MAS está entre as camadas do chão da fábrica e dos algoritmos de otimização, com os quais troca informações de forma diferente. Estas diferenças, o que os constitui e o modo como funcionam, ou se comportam, são apresentados de uma forma genérica nas subsecções seguintes.

4.4.1 Camada do Chão da Fábrica

No chão da fábrica encontramos todos os recursos da fábrica e as peças ou materiais de produção. Geralmente, para cada um destes componentes encontram-se conectados um PLC, PID e/ou vários sensores. Estes componentes são o que possibilitam a esse recurso ou produto/material, de ter uma representação virtual no MAS (camada seguinte).

Quando um produto ou um recurso é ligado ao sistema, este dará origem, respectivamente, a um PA ou RA. As informações relativas ao produto ou recurso para passar à camada do MAS podem ser muitas e variadas, mas para evitar o acréscimo de complexidade e manter a arquitetura o mais genérica possível, apenas serão considerados os

aspectos essenciais para o seu normal funcionamento. Tanto para o recurso como para o produto, as informações que irão ser transmitidas à camada seguinte do MAS são:

- **Recurso**
    1. **Identificação** - Geralmente um nome ou código único para identificação do recurso;
    2. **Posição** - Coordenadas que permitam situar o recurso no chão da fábrica;
    3. **Skills** - Uma lista com as várias ações que o recurso pode exercer sobre um produto.
- **Produto**
    1. **Identificação** - Nome ou código único associado ao produto;
    2. **Posição** - Coordenadas da posição atual do produto;
    3. **Tasks** - Uma lista ordenada com as várias *Skills* a executar.

Esta camada encontra-se continuamente em comunicação com os seus respectivos agentes na camada do MAS, de modo a refletir no mundo físico as ações que lhes são solicitadas.

4.4.2 Camada MAS

Esta é a camada central onde podemos encontrar o MAS. É aqui que se encontram as duas entidades essenciais para qualquer sistema de manufactura (recursos e produtos), é onde se gere toda a produção, ou seja, para que recurso se deve dirigir certo produto para realizar as *skills* e também onde se registam e monitorizam todas as posições, movimentos, ações e alterações dos vários agentes presentes no sistema.

Esta camada esta dividida em duas zonas, uma que contém os agentes (a mais claro) e outra de integração que contém serviços e bases de dados (mais escura), como podemos ver na Figura 4.3. Nas subsecções seguintes entraremos mais em detalhe sobre as entidades que compõem cada uma das zonas, começando pelos agentes e acabando nos serviços e bases de dados.

4.4.2.1 Agente Produto

Um Agente do Produto ou Product Agent (PA), é a representação virtual de um produto físico. Este agente é responsável pela gestão de todas as operações de manufactura necessárias de um produto na linha de produção.

Quando um produto físico é inserido no sistema, dá-se a criação de um PA, requerendo as informações mínimas, já citadas anteriormente, a respeito da sua identificação, posição e *Tasks*, sendo esta última informação não mais do que uma lista com várias *Skills* ordenadas.

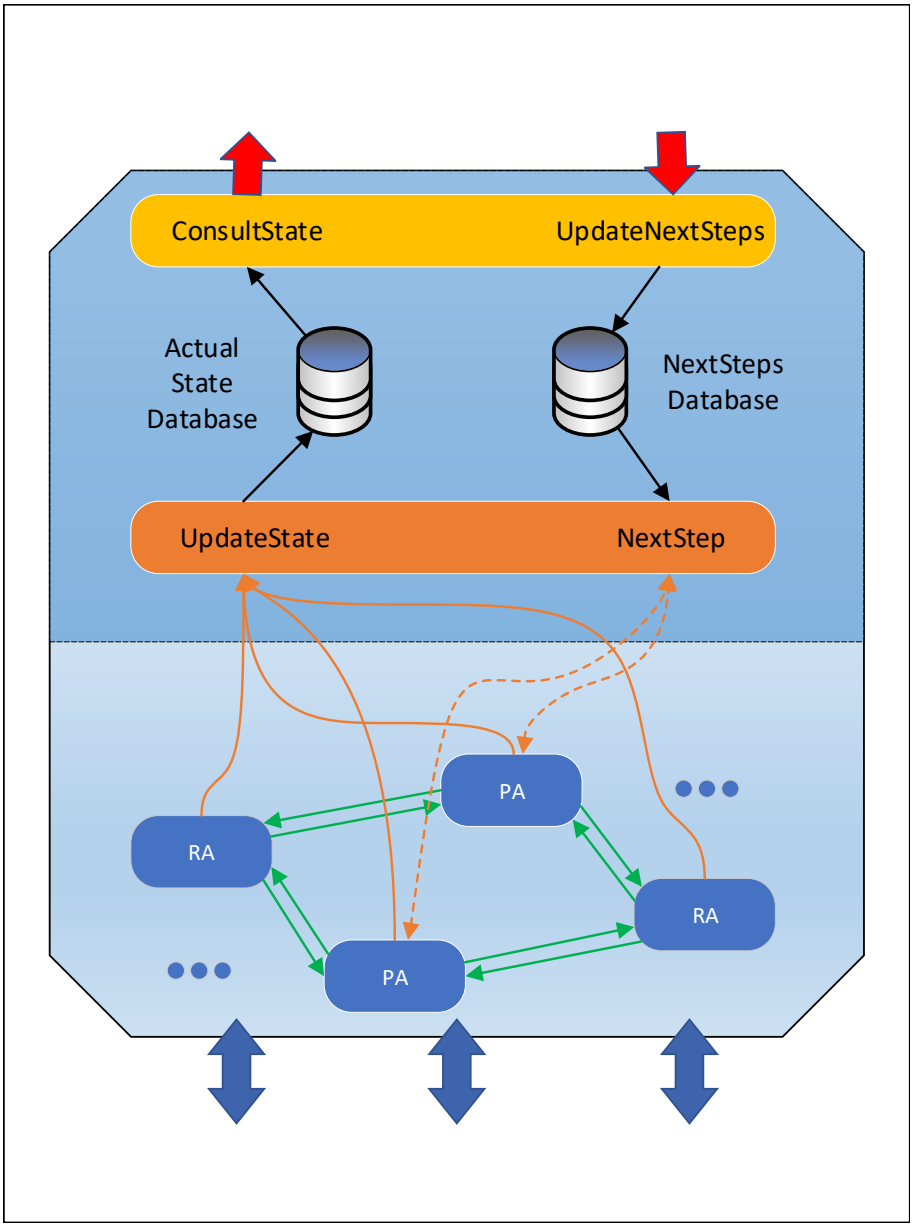

Figura 4.3: Diagrama da camada central que contém o MAS

Para que um PA possa funcionar corretamente e concluir a sua tarefa, ou seja, conseguir chegar a um produto final, é necessário existirem recursos capazes de processar as operações necessárias da sua lista.

As *skills* são extraídas uma a uma e de forma ordenada da lista e só são removidas após confirmação de um recurso da sua execução bem sucedida. Sempre que se extraí uma *skill* da lista, o agente PA requisita a um serviço qual o melhor percurso a tomar até ao recurso capaz de executar essa *skill*. Após receber o melhor percurso, o PA dirige-se ao RA indicado. Depois de chegar ao recurso, o PA faz um *request* ao RA para executar a *skill* sobre o produto. A seguir à aceitação e sucesso na execução da *skill* pelo recurso, o PA remove essa *skill* da sua lista e avança para a seguinte, repetindo os processos anteriores até chegar ao produto final, em outros termos, até não haver mais nenhuma *skill* a executar na lista. Para melhor compreensão de todos estes processos temos o fluxograma retratado na Figura 4.4.

4.4.2.2 Agente Recurso

Um Agente de um Recurso ou Resource Agent (RA), é criado sempre que se liga ao sistema uma entidade, presente no chão da fábrica, que tenha a capacidade de executar ações sobre produtos, como por exemplo, perfurar, soldar, lixar, entre outras.

Como já foi mencionado antes e à semelhança do PA, para criar este agente são necessárias, no mínimo, as informações relativas à sua identificação, posição e lista de *skills* que ela é capaz de executar, e deverá pelo menos conter uma *skill*. Quando este é criado, ele automaticamente regista-se na base de dados **Actual State** com todas as suas informações.

O RA é normalmente menos complexo que um PA e só comunica com duas entidades. Quando é inserido no sistema, este comunica com um serviço para notificar da sua localização e *skills* que é capaz de executar. Sempre que houver alguma alteração, tanto da sua localização, como das *skills* que ele executa, este terá de comunicar essas alterações ao serviço. A outra única comunicação que o RA realiza, é com um qualquer PA que lhe requisite a execução de uma *skill*. O RA deverá comunicar a execução da *skill* indicada e no fim da sua execução, terá que responder com o seu sucesso ao PA. Estes processos estão retratados no fluxograma da Figura 4.5.

4.4.2.3 Bases de Dados e Serviços

Nesta zona encontramos serviços e bases de dados que servem como intermédio, ou camada de integração, para a comunicação padrão com a camada dos algoritmos (Figura 4.3 parte escura).

Existem dois serviços, um que comunica com os agentes e outro que comunica com os algoritmos. Ambos estes serviços têm acesso a duas bases de dados, em que uma armazena o estado atual do sistema, **Actual State**, que contém todas as informações dos agentes existentes, e a outra, **Next Steps**, que contém todos os caminhos otimizados entre recursos para dadas *skills*.

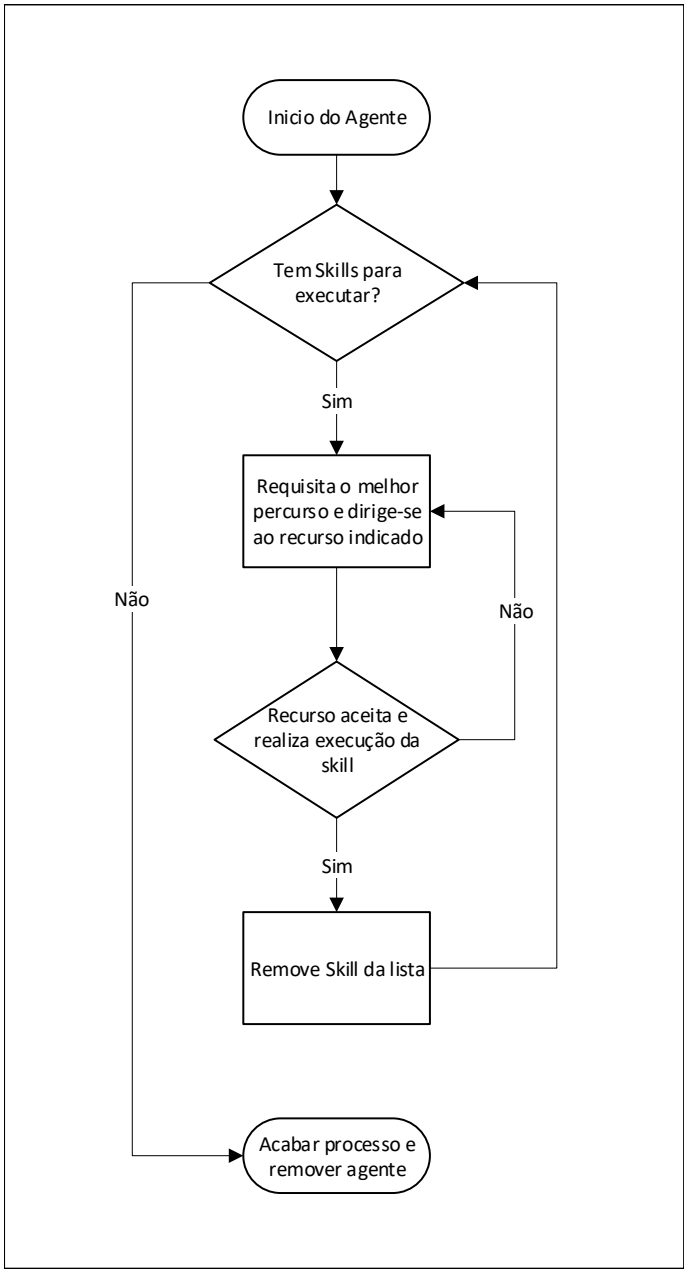

Figura 4.4: Fluxograma representativo do comportamento de um PA

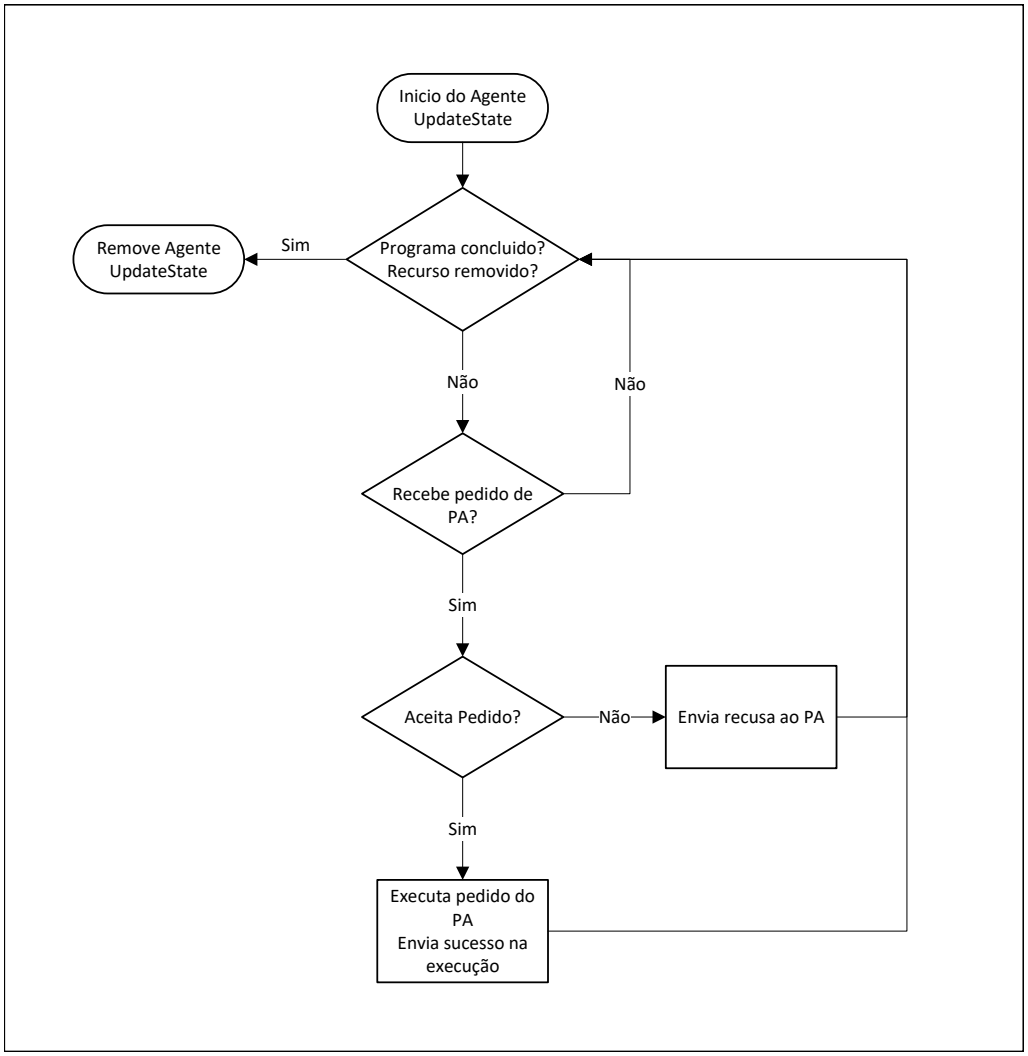

Figura 4.5: Fluxograma representativo do comportamento de um RA

O serviço que comunica com os agentes apenas pode escrever na base de dados do estado atual, através de uma função **UpdateState**, e ler a dos caminhos com a função **NextStep**. O serviço que comunica com a camada dos algoritmos apenas pode ler o estado atual, através da função **ConsultState**, e escrever na base de dados dos caminhos os percursos calculados pela camada superior, através da função **UpdateNextSteps**. Na Figura 4.6 encontramos uma representação de como é feita a comunicação dos algoritmos com a camada MAS.

Estes serviços e base de dados permitem que a camada superior dos algoritmos se tenha apenas de preocupar com otimização e também traz outro nível de robustez ao sistema, pois no caso de haver uma completa remoção da camada superior, o sistema é capaz de continuar a funcionar normalmente, com a única desvantagem de não poder utilizar novos recursos que possam ser inseridos durante o tempo em que a camada dos algoritmos não existe.

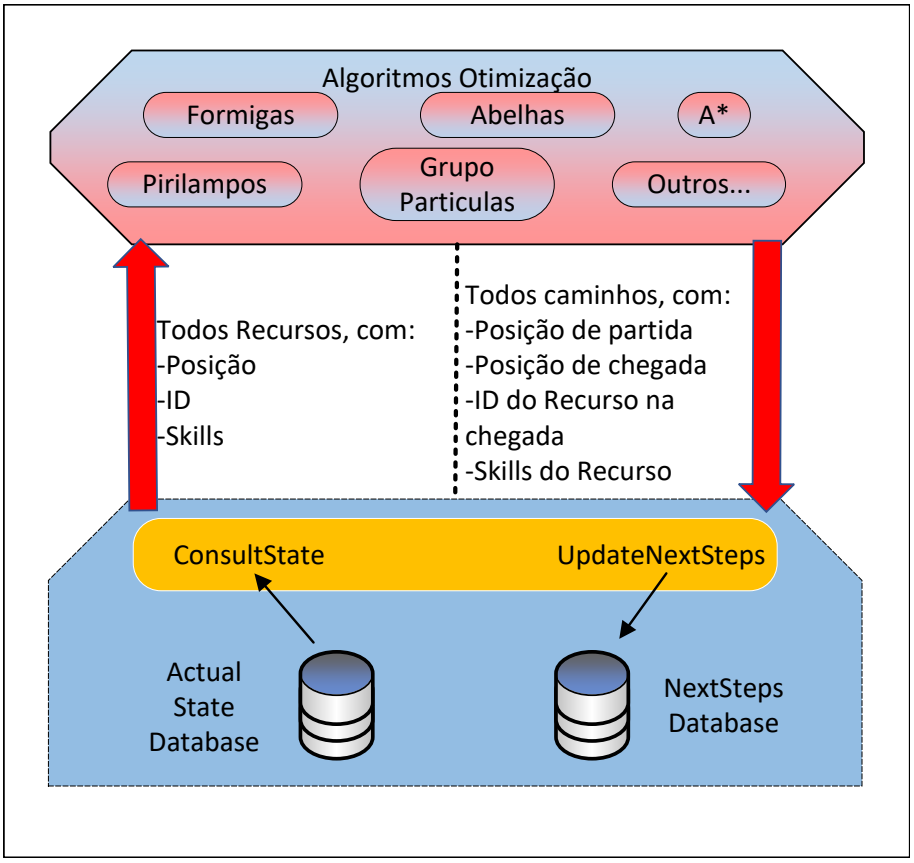

Figura 4.6: Diagrama representativo das comunicações entre as camadas do algoritmo e MAS

4.4.3 Camada dos Algoritmos

A camada dos algoritmos é a última camada da arquitetura. Nesta camada podemos encontrar vários algoritmos de otimização e escolher um para optimizar o problema.

Após se selecionar um algoritmo, este requisita á camada do MAS todos os dados existentes do estado atual do sistema, através da função **ConsultState** do serviço lá contido. O algoritmo por sua vez, irá pegar na informação sobre os recursos existentes de que necessita e calculará todos os itinerários optimizados possíveis entre recursos, de forma que, a partir de um recurso, o produto se possa dirigir a outro qualquer recurso que execute a sua próxima *skill*.

Quando o algoritmo finaliza o seu processamento e chega a uma lista com todos os itinerários optimizados, este chama o serviço da camada MAS que executa a função para criar ou atualizar a base de dados **UpdateNextSteps**. Na Figura 4.6 encontramos a representação deste comportamento com alguns dos dados que são enviados.

A frequência com que o algoritmo realiza a otimização do sistema fica a critério do engenheiro que quiser implementar esta arquitetura. Alguns exemplos que podem funcionar poderão optar pela realização cíclica de otimizações, intervalados de uma certa quantidade predefinida de tempo, ou então, sempre que for detectado uma alteração ou

inserção de um recurso.

5 Implementação

O capítulo seguinte apresenta e descreve todos os passos que foram executados para a implementação de uma arquitetura com base no que foi mencionado no capítulo anterior. O capítulo está dividido em várias secções que discutem toda a implementação desde a criação da ontologia, até aos algoritmos de otimização usados para teste. Esta arquitetura MAS foi desenvolvida em JAVA com recurso ao JADE, desenvolvido por Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood e outros, em 2007 [61].

5.1 Ontologia

O primeiro passo na implementação passou pela criação da ontologia da arquitetura. Uma ontologia é um modelo de dados onde estão descritos todos os conceitos dentro da arquitetura e suas relações, uma forma de representação de conhecimento. Para este efeito recorreu-se ao programa *Protégé*, que para além de permitir a criação de todos as entidades, suas propriedades e estabelecer todas as relações entre si, é capaz de gerar o código em *JAVA* da ontologia criada.

Na Figura 5.1 podemos observar as várias classes de que é composta e entre elas também podemos encontrar setas que representam as relações existentes. As setas a azul indicam que uma entidade, aquando da sua criação, pertence a uma das classes a que se encontra ligado, ou seja, todos os recursos estão contidos na classe *Resource*, todos os produtos estão em *Product* e todas as habilidades/executáveis estão no interior de *Skill*.

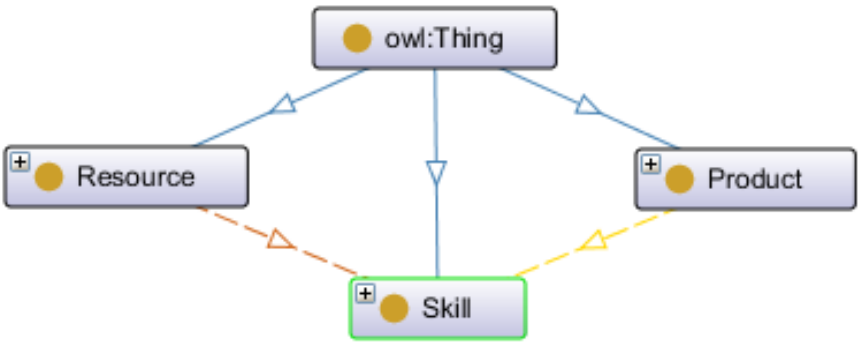

Figura 5.1: Representação gráfica da ontologia utilizada

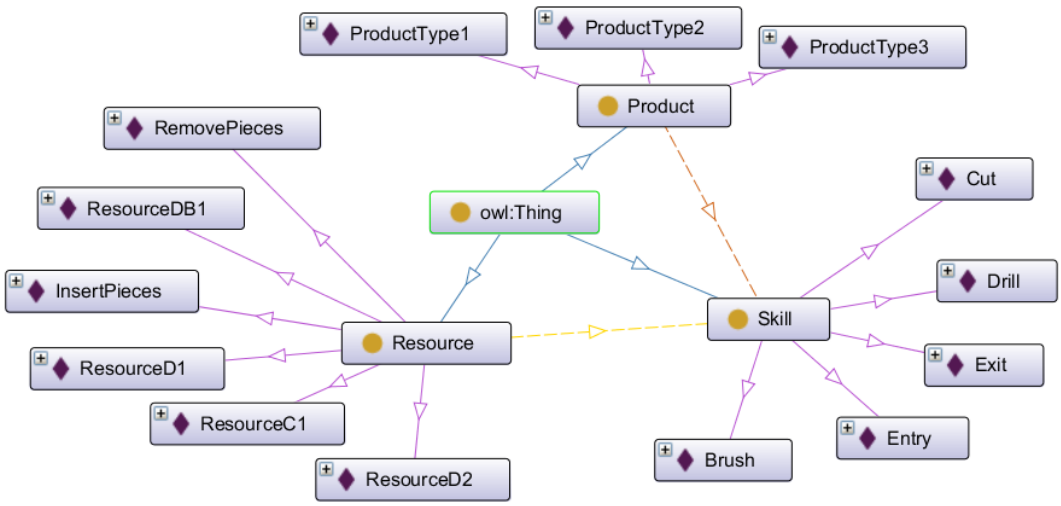

Figura 5.2: Representação gráfica da ontologia expandida com algumas entidades criadas

As setas remanescentes a tracejado indicam uma relação entre classes. Neste caso em concreto, a seta entre *Resource* e *Skill* significa que os recursos executam *skills*, enquanto a seta de *Product* para *Skill* indica que os produtos "consomem"*skills* 5.2. Dentro de cada classe podemos encontrar as entidades que as compõem com as suas propriedades, como, por exemplo, a descrição do produto ou recurso, as *skills* que consomem ou executam e suas posições.

Cada entidade possui dois tipos de propriedades:

- Propriedades do objeto - contém as ações que estas entidades executam sobre as outras. Como por exemplos, o recurso *InsertPieces* descreve que executa a *skill Entry* e o produto *ProductType1* diz que consome as *skills*: *Entry*, *Brush* e *Exit*;
- Propriedades de dados - contém todos os dados da respectiva entidade, ou seja, a sua descrição, identificação, posição (no caso dos produtos e recursos), capacidade ou quantidade (no caso dos produtos e recursos) e prioridade(somente nas *skills*).

Após a criação da ontologia o programa *Protégé* cria o ficheiro *myOntology.owl*, que é a ontologia em si, e de seguida gera ficheiros em *JAVA* para uso no projeto. Estes ficheiros possibilitaram a implementação de funções para criar e manipular entidades, ou seja, produtos, recursos e *skills*.

5.2 Bases de Dados

Nesta secção iremos descrever a implementação da camada central da arquitetura, onde encontramos as bases de dados com as quais os agentes e algoritmos irão interagir. Como já foi previamente dito, são necessárias duas bases de dados. Para tal foram criadas duas bases de dados em Structured Query Language (SQL), armazenadas localmente, com

recurso ao programa XAMPP, um pacote com os principais servidores open-source criado pelo projeto *Apache Friends*[62], que possibilitou a criação de um servidor local MariaDB.

As duas bases de dados têm funções distintas. A base de dados **Actual State**, tal como pelo nome indica, serve para armazenar o estado atual do chão da fábrica, ou seja, contém toda as informações relativamente aos recursos e produtos existentes na fábrica (Figura 5.3). A outra base de dados **Next Steps**, contém os caminhos entre os recursos, optimizados pelos algoritmos, para o uso pelos produtos (Figura 5.4).

#	Nome	Tipo
1	ID	char(30)
2	type	char(10)
3	position	char(10)
4	skills	char(60)
5	quant_capacity	int(11)

Figura 5.3: Estrutura interna da base de dados Actual State

#	Nome	Tipo
1	Unique_ID	int(11)
2	position_initial	char(10)
3	position_end	char(10)
4	resource_id_end	char(30)
5	skill_to_execute	char(60)

Figura 5.4: Estrutura interna da base de dados Next Steps

O dados presentes em cada base de dados têm as seguintes funções/significados:

- **Actual State**
  1. **ID** - contém o nome do recurso/produto que é único (ex.: ProductType1_0, ResourceDB3, ...);
  2. **type** - Permite distinguir se os dados estão associados a um produto ou um recurso;
  3. **position** - Coordenadas no formato (x,y), que permitem situar o recurso/produto no chão da fábrica;
  4. **skills** - Uma linha com as *skills* que o recurso pode exercer sobre um produto, ou no caso de ser um produto, as *skills* que se pretendem realizar por ordem de execução;
  5. **quant_capacity** - Informa a capacidade de um recurso ou a quantidade de produto a produzir.
- **Next Steps**
  1. **Unique_ID** - Código único associado ao caminho, que é atribuído automaticamente assim que é inserido na base de dados;
  2. **position_initial** - Coordenadas iniciais do caminho, isto é, coordenadas de onde se inicia o trajeto do produto até ao recurso;

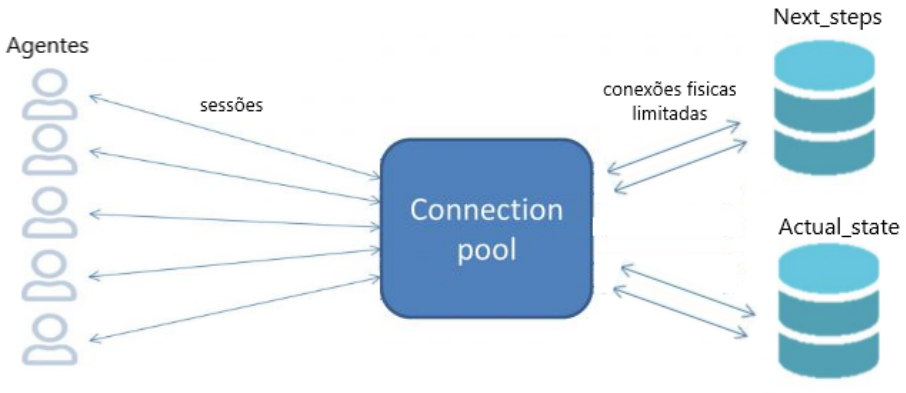

Figura 5.5: Representação básica de como funciona uma Connection Pool

3. **position_end** - Coordenadas finais do caminho, onde o recurso de encontra;
4. **resource_id_end** - Nome do recurso para onde se dirige o produto (igual ao ID que se encontra na base de dados Actual State);
5. **skill_to_execute** - Linha com as várias *Skills* possíveis de executar.

Para comunicar com as bases de dados, foram criadas conecções com recurso à tecnologia Java Database Connectivity (JDBC) adicionando o componente de serviços *pooling* Database Connection Pool (DBCP) fornecido pela *Apache Commons*[63]. Estabelecer conecções JDBC, a nível de custo de recursos é muito caro, principalmente neste caso, em que esta camada central poderá ser acedida por múltiplos serviços ao mesmo tempo (agentes). Neste tipo de situações, o uso de uma "piscina de conecções"(**connection pool**) pode ter vantagens muitos significativas a nível de desempenho, recorrendo da técnica de *pooling* sobre essas conecções (Figura 5.5).

Connection pooling significa que as conecções são reutilizadas sempre que uma conexão é requisitada, em vez de serem criadas novas ou recriadas. Para facilitar a sua reutilização é mantida em memória uma certa quantidade limitada de conecções físicas para as bases de dados, a que chamamos de **connection pool**, e é mantida por um módulo de *pooling*, proveniente do DBCP. O *pooling* das conecções que a *Apache Commons* fornece é dividido em vários pequenos recursos, ou seja, é "multi-thread"e corre em segundo plano.

Para se obter uma conexão às bases de dados, usa-se o objeto **DataSource**. O objeto **DataSource** é que se responsabiliza pela realização do *pooling* às conexões existentes, em que, caso haja uma conexão disponível, devolve essa conexão da "piscina"e caso contrário, cria uma nova conexão. A reutilização das conexões não implica alteração nenhuma no código normal utilizado pela aplicação ou serviços para realizar conexões, visto que estas conexões reutilizadas se comportam da mesma forma que conexões físicas novas ás bases de dados. Quando a aplicação ou serviço acaba de realizar as tarefas de que necessita através da conexão, esta é encerrada e adicionada ou devolvida à "piscina"de conexões

para ser reutilizada por outras aplicações ou serviços. No anexo I podemos encontrar as configurações utilizadas para os **DataSource**'s das duas bases de dados. Na arquitetura implementada, foram definidas para que no máximo existissem cem conecções, simultaneamente, por base de dados, que perfaz um total de duzentas conecções que podem existir em paralelo.

A classe *DBConnection* é uma das classes mais importantes na arquitetura. Esta é a biblioteca responsável por todas as interações, configurações e execução das conexões com as bases de dados. É utilizada por todos os agentes e pela camada de algoritmos, sempre que estes tenham de ler, escrever, apagar ou procurar dados em ambas as bases de dados (Figura 5.6).

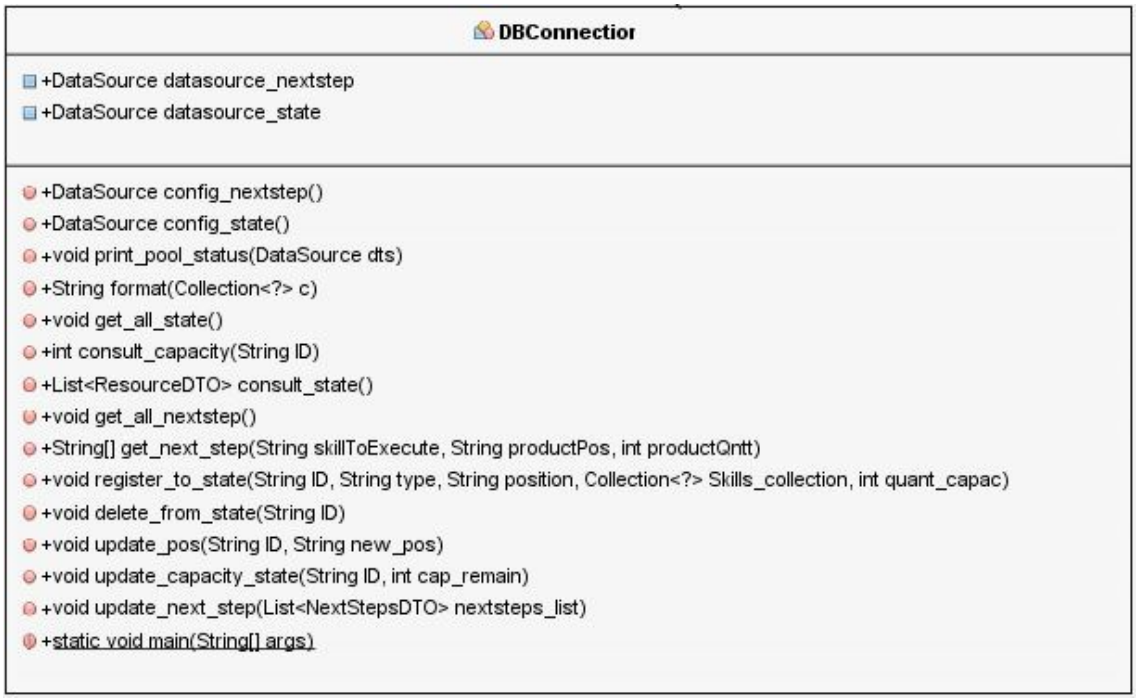

Figura 5.6: Diagrama da Classe *DBConnection*

A classe *Constants_DTOs* é uma pequena classe que contém dois Data Transfer Object (DTO)s e uma função auxiliar para conversão de coordenadas. Estes DTOs são utilizados para transferir e receber informação das bases de dados (Figura 5.7).

5.3 Serviços Complementares

O MAS foi desenvolvido na linguagem de programação JAVA, através do IDE NetBeans, com recurso ao framework JADE. Este framework ajuda a simplificar a implementação deste tipo de sistemas, através da integração de software que fornece novos serviços e estende as capacidades do NetBeans, denominado de *middleware*. O JADE cumpre com as especificações padrão de comunicações Foundation for Intelligent Physical Agents (FIPA)

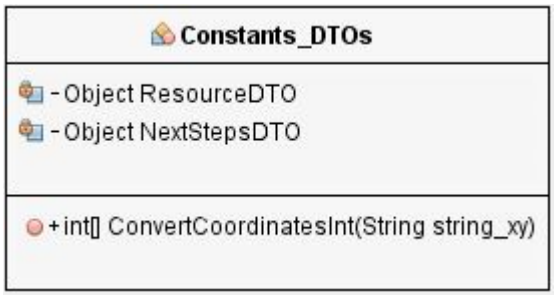

Figura 5.7: Diagrama da Classe *Constants_DTOs*

[64] e oferece um conjunto de ferramentas gráficas para ajudar durante as fases de debugging, lançamento e execução dos agentes. Nas subsecções seguintes, serão apresentados e descritos os serviços acima mencionados, com maior pormenor, entre outros fornecidos pelo JADE e as classes complementares que foram criadas para facilitar algumas funções do MAS.

5.3.1 Comunicação FIPA

FIPA é um conjunto de protocolos padrão de comunicação aceite pelo Institute of Electrical and Electronics Engineers (IEEE), uma associação profissional para engenharia eletrotécnica, no ano de 2005, que promove a tecnologia baseada em sistemas multi-agentes e a sua integração com tecnologias já existentes [64].

O JADE fornece a implementação de alguns protocolos de interação do FIPA, mas não todos. Os principais são: **FIPA-CONTRACT-NET**, **FIPA-QUERY** e **FIPA-REQUEST**. Nesta arquitetura unicamente foi utilizado um protocolo baseado no **FIPA-REQUEST**. Este é usado entre dois agentes, onde um deles requer do outro a execução de uma ação, ou seja, uma *skill*.

Para que este protocolo funcione, um agente, ao qual lhe chamamos de **Initiator**, deve enviar um pedido a outro, denominado de **Participant**, sob o formato de um **request**. Este último processa o **request** e decide sobre a sua realização, devolvendo uma resposta na forma de **refuse** e tornando a variável **refused** a verdadeiro, se a a execução da ação for refutada. De outra forma, a variável que se torna verdadeira será **agreed** e, caso seja requerido, envia uma resposta no formato **agree** (opcional).

Após aceitação do **request** (ou seja, variável **agreed** a verdadeiro), o agente **Participant** executa a ação solicitada. De seguida, depois da realização, ou não, da ação, o agente deve comunicar com o **Initiator** recorrendo a uma das seguintes mensagens[65]:

- **failure** - Se falhar na execução do pedido;
- **inform-done** - Caso a execução do pedido tenha sido bem sucedida e apenas é pretendido a comunicação do seu sucesso, ou seja, de que está feito;

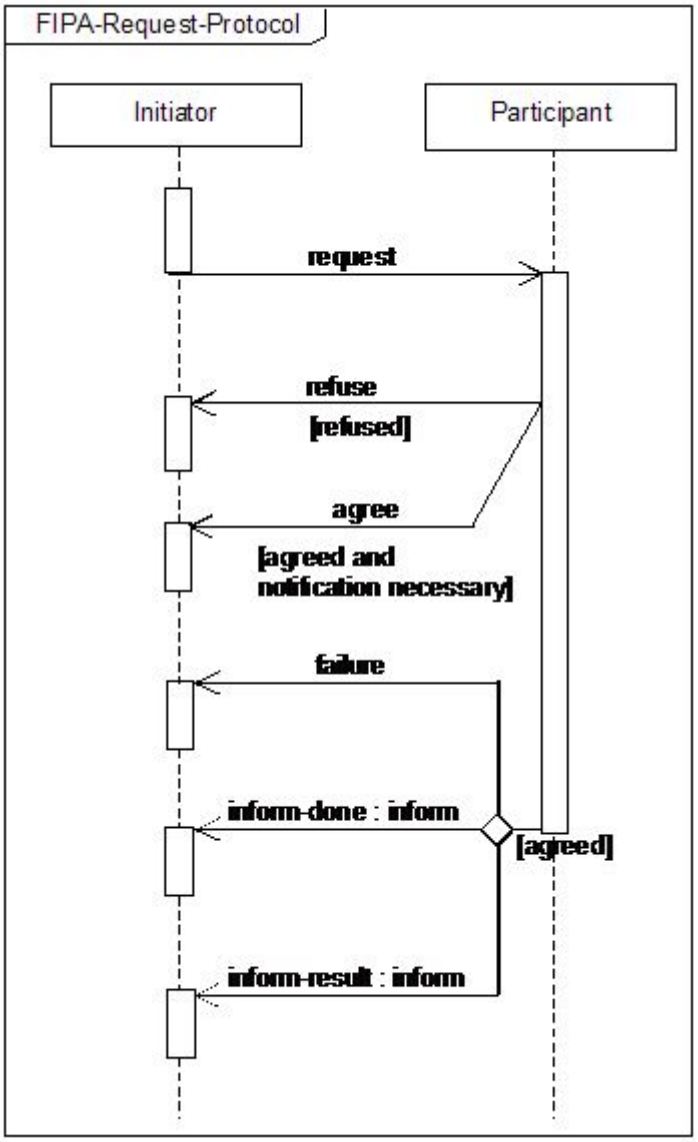

Figura 5.8: Protocolo de interação FIPA-Request [65]

- **inform-result** - Semelhante ao **inform-done**, mas com a diferença de que o **Initiator** é notificado dos resultados.

Na Figura 5.8 podemos visualizar um diagrama, que se encontra em [65], o qual resume e simplifica a compreensão de todo o comportamento do protocolo de interação **FIPA-REQUEST** acima descrito.

5.3.2 Serviço Páginas Amarelas

Outro serviço utilizado na arquitetura e que está contido no *framework* do JADE é o de "páginas amarelas". O JADE tem implementado um agente designado de DF, que é normalmente equiparado a uma lista telefónica, de onde surge o nome de "páginas

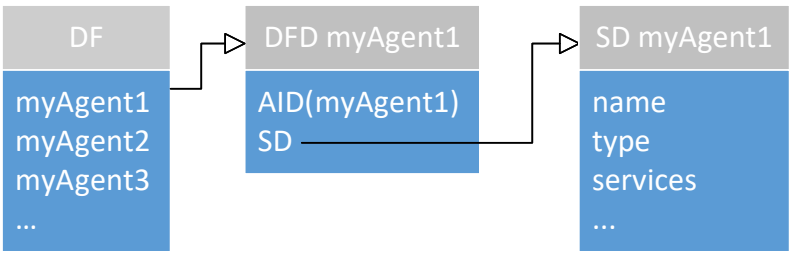

Figura 5.9: Exemplo Estrutura dos Dados no DF

amarelas". Isto porque, é aqui que se registam os agentes que querem publicar os seus serviços e/ou outros dados e onde os outros agentes vão à procura por esses agentes com os serviços e/ou dados desejados.

O DF é um registo das entradas que associam as descrições de agentes ao seu respectivo agente, através de um JADE Agent Identifier (AID). A estrutura de dados que é utilizada para registar um agente no DF é conhecida como DFAgentDescription (DFD) e onde podemos encontrar contidos ambos o AID e sua descrição ServiceDescription (SD) com os serviços e/ou dados (Figura 5.9).

Esta estrutura de dados tanto é usada para o registo, como para a procura no DF. Quando pretendemos registar, temos de fornecer uma descrição completa e um AID; enquanto que na procura, apenas necessitamos de ter uma parte da descrição. A pesquisa no DF retorna uma lista (array) com todas as entradas DFD que coincidam com os dados contidos na descrição usada para procura e de onde poderemos extrair a identificação dos agentes associados para, por exemplo, iniciarmos uma comunicação/interação.

5.3.3 Classes e outras Funções Auxiliares

De modo a diminuir o consumo de recursos, evitar repetição de código e promover uma melhor estruturação da arquitetura, foram implementadas classes com funções que fossem comuns aos serviços utilizados pelo MAS e que retirassem alguma carga computacional deste sistema. Assim surgiu a classe *DFInteraction*,

A classe *DFInteraction* é responsável pela interação com o DF, aqui podemos encontrar funções de adição e remoção de entradas, assim como funções de pesquisa (Figura 5.10).

Para além da classe auxiliar, o JADE também fornece uma interface gráfica que nos permite ver, em tempo real, o estado e informação dos agentes presentes no sistema e no DF. Esta interface aparece quando a arquitetura é executada juntamente com a linha de comandos `jade.Boot` e é uma ferramenta essencial de testes (Figura 5.11).

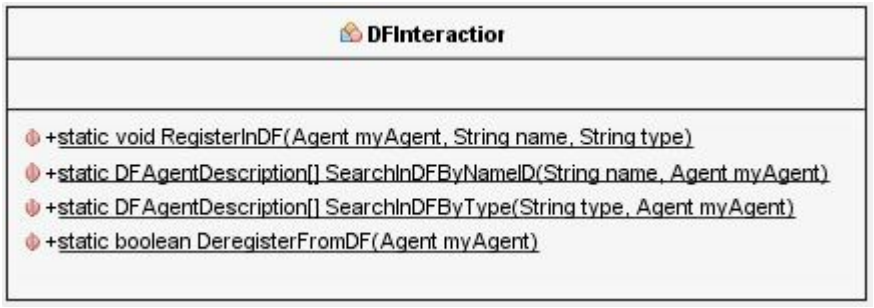

Figura 5.10: Diagrama da Classe *DFInteraction*

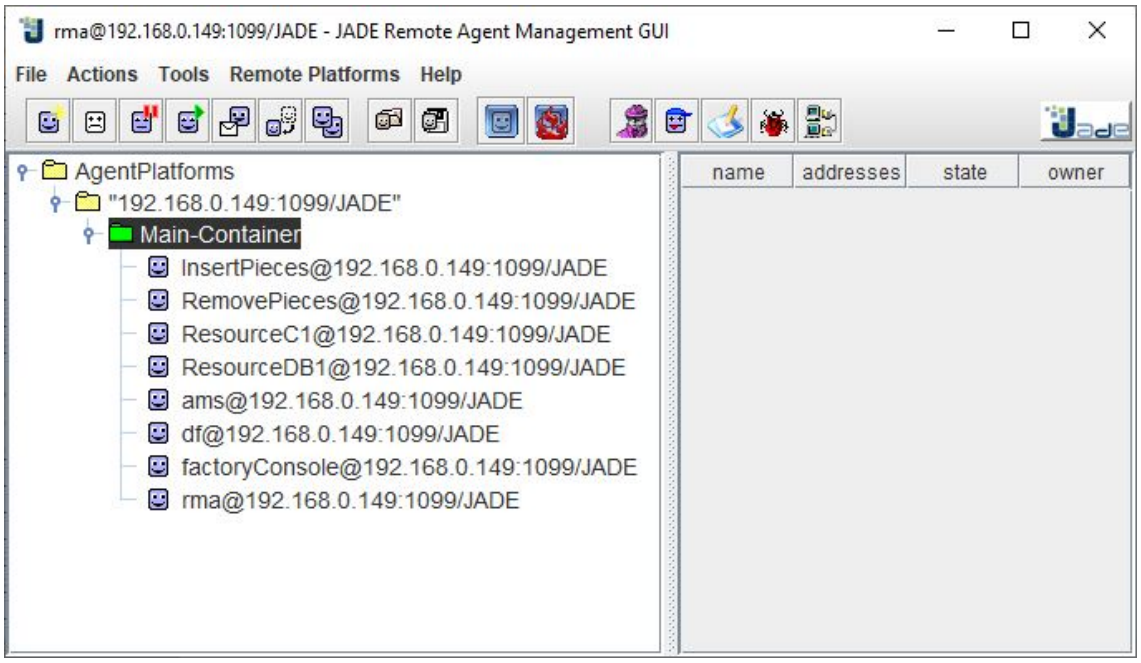

Figura 5.11: Interface Gráfica do JADE

5.4 Agentes

Esta secção descreve como foram implementados os agentes **Resource** e **Product**. Como já fora referido, ambos foram implementados no NetBeans em Java, com recurso ao framework JADE. Estes agentes são o que compõe a camada ciber-física da fábrica, sendo cada agente uma representação virtual de um elemento real.

5.4.1 Agente Recurso

Dos dois tipos de agentes existentes, o RA é o menos complexo. Este é responsável pela a execução das *skills* requisitadas pelos PAs e é estático, ou seja, não se movimenta. Na Figura 5.12 podemos ver as classes que constituem estes agentes.

O RA contém uma função `setup` e o comportamento do tipo `CyclicBehaviour`.

No `setup` o RA regista no DF o seu nome e tipo, ou seja, o nome do recurso, que deverá

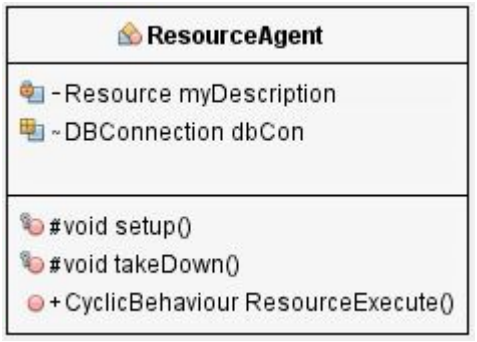

Figura 5.12: Diagrama das Classes que compõem o RA

ser único, e de que tipo é, neste caso *Resource*, de modo a se diferenciar dos produtos. Esta descrição fica guardada no DF com um AID associado e que posteriormente será utilizado pelo PA.

Depois do registo no DF, o agente inicia o seu registo na base de dados **Actual State**. Através da função `register_to_state` da classe *DBConnection*, ele regista-se com:

- `ID` - Nome único do recurso, não confundir com o AID;
- `type` - *Resource*;
- `position` - coordenadas x,y de onde se encontra;
- `skills_collection` - uma coleção das *skills* de que é capaz de executar;
- `quant_capac` - Visto que se trata de um *Resource*, define a sua capacidade.

Após concluídos os registos, tanto no DF, como na base de dados **Actual State**, o agente inicia o seu comportamento `ResourceExecute()`.

O `ResourceExecute()` é um `CyclicBehaviour`, um dos vários comportamentos que o JADE fornece. Este comportamento tem a particularidade de estar permanentemente ativo e a ser chamado recursivamente enquanto o agente existir. O RA está sempre à espera de receber uma mensagem proveniente de um PA, para realizar a execução de uma *skill*. Quando a recebe, este primeiro verifica se a mensagem é válida, se cumpre com as especificações do **FIPA-REQUEST** e se não se encontra com muitos produtos em espera. Caso a mensagem se encontre válida e não seja ultrapassado o limite de produtos em espera, ele dá inicio à execução da *skill*, executa-a e informa o PA do seu término bem sucedido, na forma de um **inform-done**. Na eventualidade de a mensagem não for válida ou não for possível executar por se encontrar ocupado, o RA manda para o PA uma mensagem com **failure** ou **refuse**, respectivamente, tal como se pode encontrar representado na Figura 5.13.

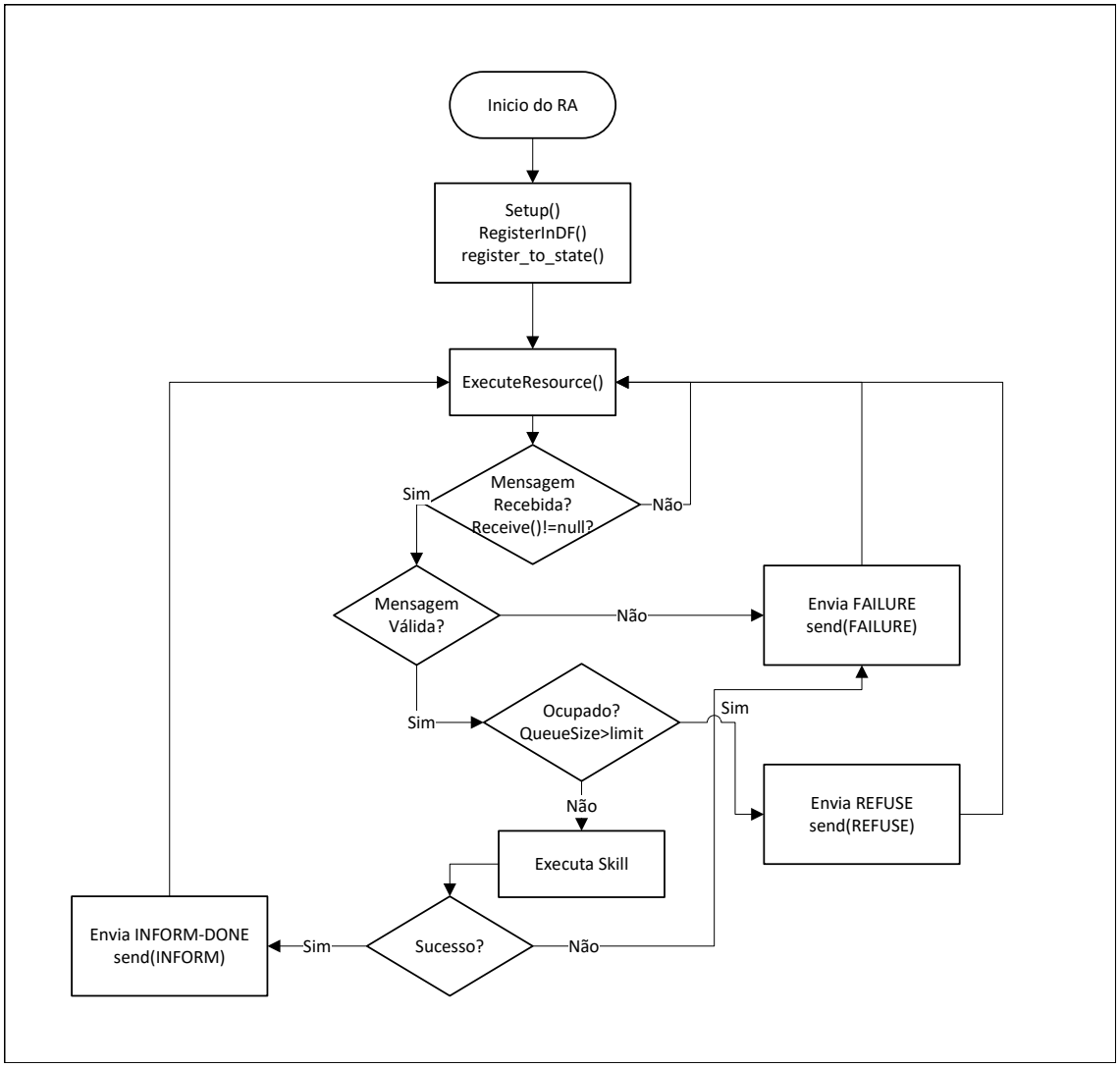

Figura 5.13: Fluxograma representativo do comportamento do RA

5.4.2 Agente Produto

Embora o PA possua a mesma quantidade de classes que o RA (Figura 5.14), este apresenta uma complexidade maior e desloca-se. O PA é responsável por todos os procedimentos necessários para a transformação do produto, desde a sua entrada na fábrica, até á sua saída.

No `setup()` o PA comporta-se de forma muito idêntica ao RA, registando-se no DF e na base de dados **Actual State** com a mesma função `register_to_state` da classe *DBConnection*. As únicas diferenças são o registo do tipo(`type`) como *Product*, quantidade em vez de capacidade e as *skills* que o produto consome. As *skills* são retiradas da descrição do agente e não se encontram ordenadas por prioridade de execução. Portanto, antes de se enviar uma lista com as *skills* para a base de dados, o `setup()` também transforma e organiza as *skills* de forma a estarem em conformidade com a ordem de execução

Figura 5.14: Diagrama das Classes que compõem o PA

pretendida.

Após efetuados os registos tanto no DF, como na base de dados **Actual State**, o agente inicia o seu comportamento `ExecuteListSkills()`. Este é do tipo `SimpleBehaviour`, que não possui particularidade nenhuma e apenas executa o código uma vez. Ao contrário de RA, onde o comportamento é chamado uma vez e fica a correr até o agente deixar de existir, o PA adiciona um comportamento para cada *skill* existente na lista de execução, ou seja, cada *skill* é individualmente executada por um comportamento e estes comportamentos são executados de forma ordenada, dependendo da conclusão do anterior. Estes comportamentos estão igualmente vinculados há existência do agente.

Para a execução de uma *skill*, o `ExecuteListSkills()` do PA começa por a comparar e verificar se é uma *skill* válida, ou seja, uma qual se sabe que pode ser executada. Se não for válida é apresentado um aviso de que a sua execução é impossível e de seguida procede-se à remoção do agente. Caso a *skill* seja válida, o PA vai buscar o próximo passo (*next_step*) à base de dados **Next Steps** através da função `get_next_step()` da classe *DBConnection* com os parâmetros relativos à *skill* a executar, posição atual do produto e quantidade. Esta função devolve uma lista com a identificação de todos os recursos capazes de executar a devida *skill* a partir da localização onde se encontra atualmente o agente. Na eventualidade de não existir nenhum recurso capaz da execução, procede-se à remoção do agente, se encontrar, mas este se encontrar ocupado e for o único recurso capaz de executar a *skill*, o PA espera durante cinco segundos. Se durante esses cinco segundos nenhum recurso ficar disponível, também é feita a remoção do agente. Por fim, se encontrar um recurso capaz da execução e que se encontre livre, são registados o seu nome e posição.

Com o nome do recurso, procede-se à obtenção do AID do RA associado, através de uma pesquisa no DF, para comunicação. O PA desloca-se então para o recurso e dá inicio à comunicação através da criação de uma mensagem no formato do protocolo *REQUEST*. É definido como destinatário da mensagem o RA cujo AID se obteve, estabelece-se que

o conteúdo é a *skill* a realizar e criado um parâmetro globalmente único para identificar todas as mensagens da conversa entre agentes. Depois de definidos todos os parâmetros, efetua-se o envio da mensagem para o RA. Posteriormente o PA fica a aguardar por uma resposta do RA. Se a resposta demorar demasiado tempo a chegar, procede-se de novo à procura do RA no DF e tentamos enviar uma nova mensagem.

O PA ao identificar a recepção de uma mensagem, verifica se se trata de uma mensagem do tipo **inform-done**. Se for, é finalizado o comportamento atual e dá-se inicio ao seguinte com a *skill* associada. Caso contrário, ou seja, no caso de receber uma mensagem com **failure** ou **refuse**, o PA recomeça de novo o comportamento de procura de um recurso, mas na posição em que se encontra agora.

No final, depois de realizadas todas as *skills* existentes na lista de execução, o agente sai do sistema. Nas figuras 5.15 e 5.16, encontramos respectivamente, o fluxograma do comportamento do PA acima descrito e o diagrama sequencial com as interações entre agentes durante um normal processo execução do PA.

5.5 Algoritmos

Os algoritmos constituem a camada mais alta da arquitetura, estes são responsáveis pelas decisões que os PA's tomam e pelo funcionamento eficiente do chão da fábrica. Estes algoritmos retiram informações do estado atual, na base de dados **Actual State**, e utilizam-nas para calcular os melhores trajetos, que são depois inseridos na base de dados **Next Steps**, acedida pelos agentes PA. Sem a execução de pelo menos uma vez, de qualquer um dos algoritmos, o sistema não funciona, pois a base de dados **Next Steps** encontra-se vazia inicialmente.

A particularidade desta arquitetura é a possibilidade de utilizar diferentes algoritmos de otimização e até alterá-los enquanto o sistema corre. Nas subsecções seguintes são descritos alguns algoritmos utilizados para a otimização do processo fabril.

5.5.1 Algoritmo Básico

O algoritmo básico tem como base as distâncias euclidianas, proveniente da aplicação repetida do teorema de Pitágoras, com uma equação do tipo 5.1.

$$dist(R_A, R_B) = \sqrt{(x_{R_A} - x_{R_B})^2 + (y_{R_A} - y_{R_B})^2} \tag{5.1}$$

O algoritmo começa por colocar numa lista, todos os recursos presentes, naquele momento, da base de dados **Actual State**. Depois, para cada recurso existente na lista, regista as posições e calcula a distância de todos os percursos diretos para outros recursos cujas *skills* sejam diferentes. Estes percursos são adicionados a uma nova lista e volta a fazer o mesmo para o próximo recurso da lista inicial, repetindo este procedimento até não haver mais recursos na lista (Figura 5.17).

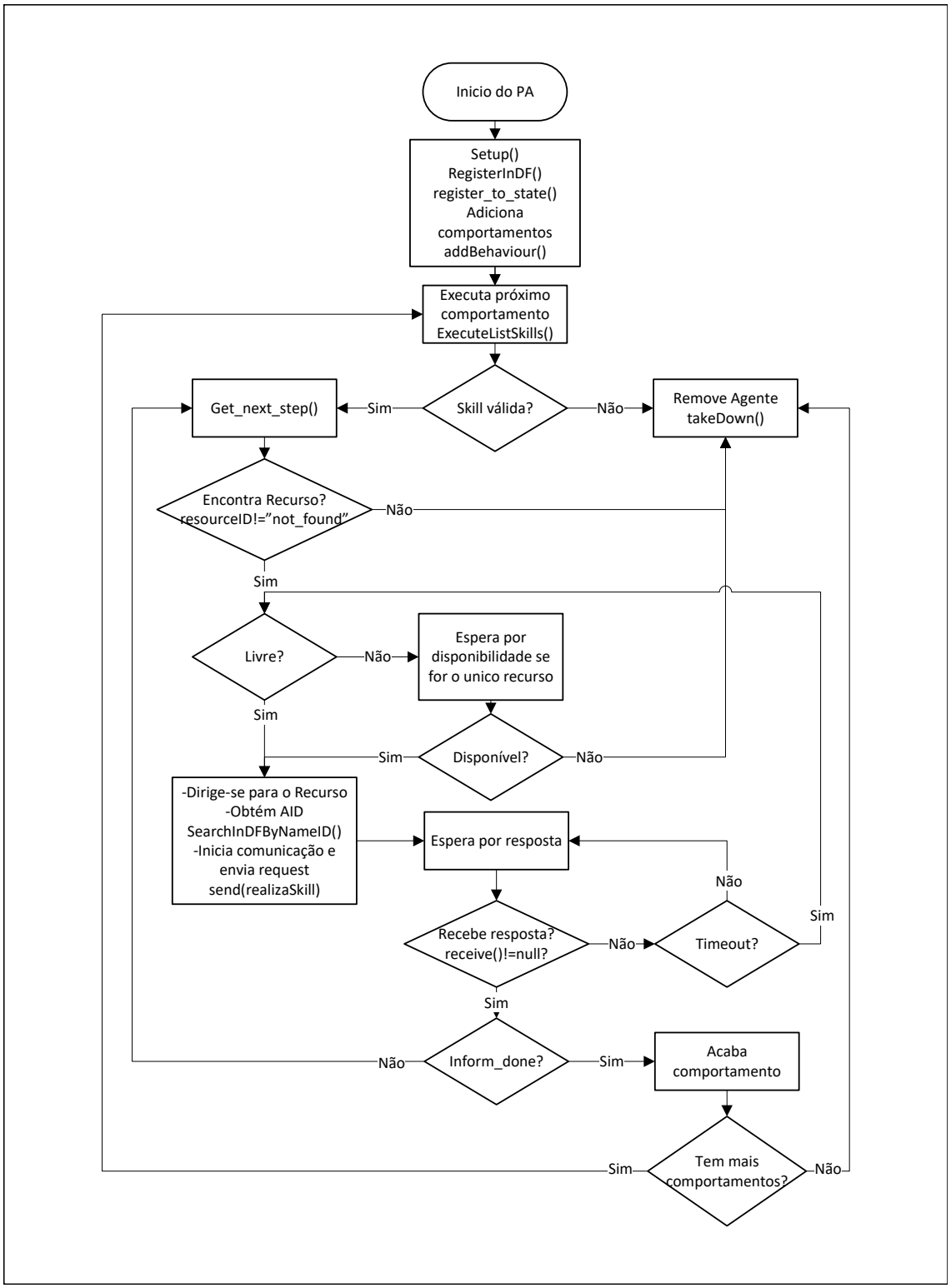

Figura 5.15: Fluxograma representativo do comportamento do PA

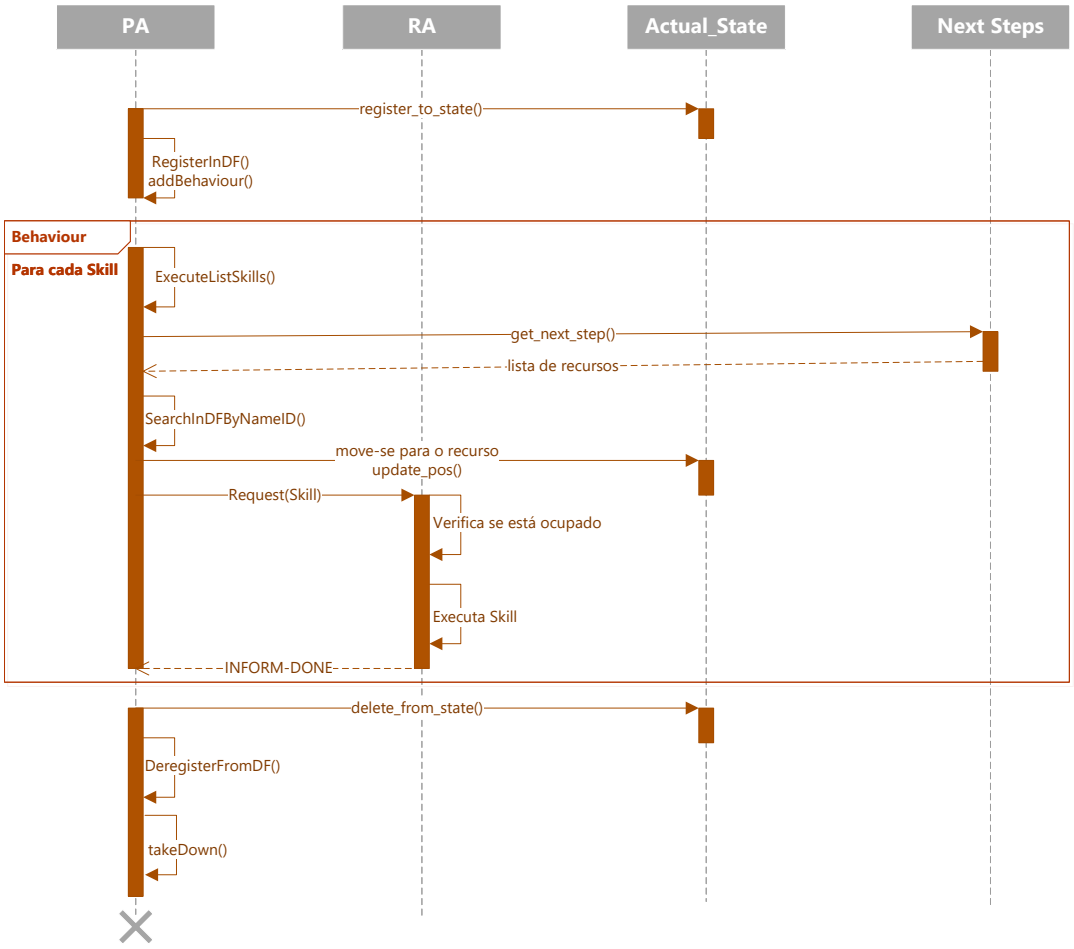

Figura 5.16: Diagrama sequencial representativo das interações normais de um PA

A lista onde estão guardados os percursos é então ordenada por ordem crescente de distâncias. Depois de ordenada a lista, o algoritmo procede à escrita desta lista na base de dados **Next Steps**, colocando os trajetos mais curtos primeiro para leitura e pesquisa dos agentes PA na base de dados. Na Figura 5.18 encontramos um fluxograma que ajuda a compreender o comportamento do algoritmo.

Este algoritmo é completo, mas não é ótimo e cresce de forma exponencial com a adição de novos recursos.

5.5.2 Algoritmo A*

O **A*** [66] é um algoritmo de procura informada, que funciona com base em grafos pesados. A partir de um nó inicial, este procura o caminho até um nó objectivo, que possua o menor custo possível. Ele consegue fazer isto, mantendo em memória e expandindo as ramificações dos nós provenientes do nó inicial, que se expandem "ramo-a-ramo"até chegarem ao nó objectivo e forem satisfeitos os critérios sobre o custo do caminho.

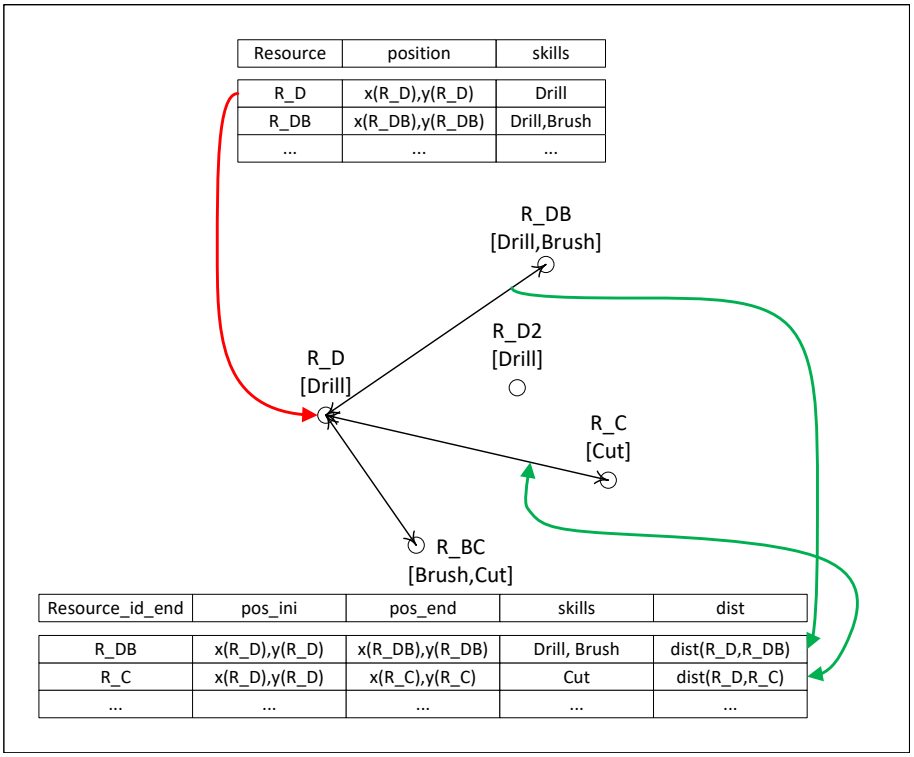

Figura 5.17: Exemplificação do método utilizado para obtenção dos percursos no algoritmo básico

Na parte inicial deste algoritmo podemos encontrar um comportamento semelhante ao algoritmo 'básico', onde retiramos todos os recursos da base de dados, colocamos numa lista e calculamos as respectivas distâncias entre os pontos de partida e chegada dos percursos.

Ao contrário do algoritmo anterior, este conhece todos os tipos de peças que a fábrica pode produzir e as *skills* que têm a executar. Antes da execução do algoritmo em si, o programa acede à ontologia da fábrica e armazena numa lista, outras listas com todas as *skills* que cada tipo de produto existente executa, ou seja, cada lista de *skills* na lista, está associada a um tipo de produto.

Recorrendo à lista anterior, dá-se então inicio à execução do algoritmo **A***[66]. O algoritmo implementado itera sobre a lista de produtos e adquire a lista das *skills* devidamente ordenadas para execução. Esta lista de *skills* é iterada também e está relacionada com a profundidade dos nós que executem uma dessas *skills*, ou seja, os nós que se encontrem a uma profundidade de 1 são os recursos responsáveis pela a execução da primeira *skill*, os nós com profundidade 2 são recursos capazes de executar a segunda *skill*, etc. O algoritmo começa por inserir o nó de partida na lista de nós do algoritmo. Visto que, neste caso, só existe um recurso capaz de executar a *skill* "`Entry`"e este é comum a todos os produtos, o nosso nó de partida selecionado e que é inserido na lista de nós do algoritmo será sempre o recurso `InsertPieces`. Após inserido o nó de partida com a profundidade 0 e incrementado o iterador das *skills*/profundidade, o algoritmo entra num comportamento cíclico

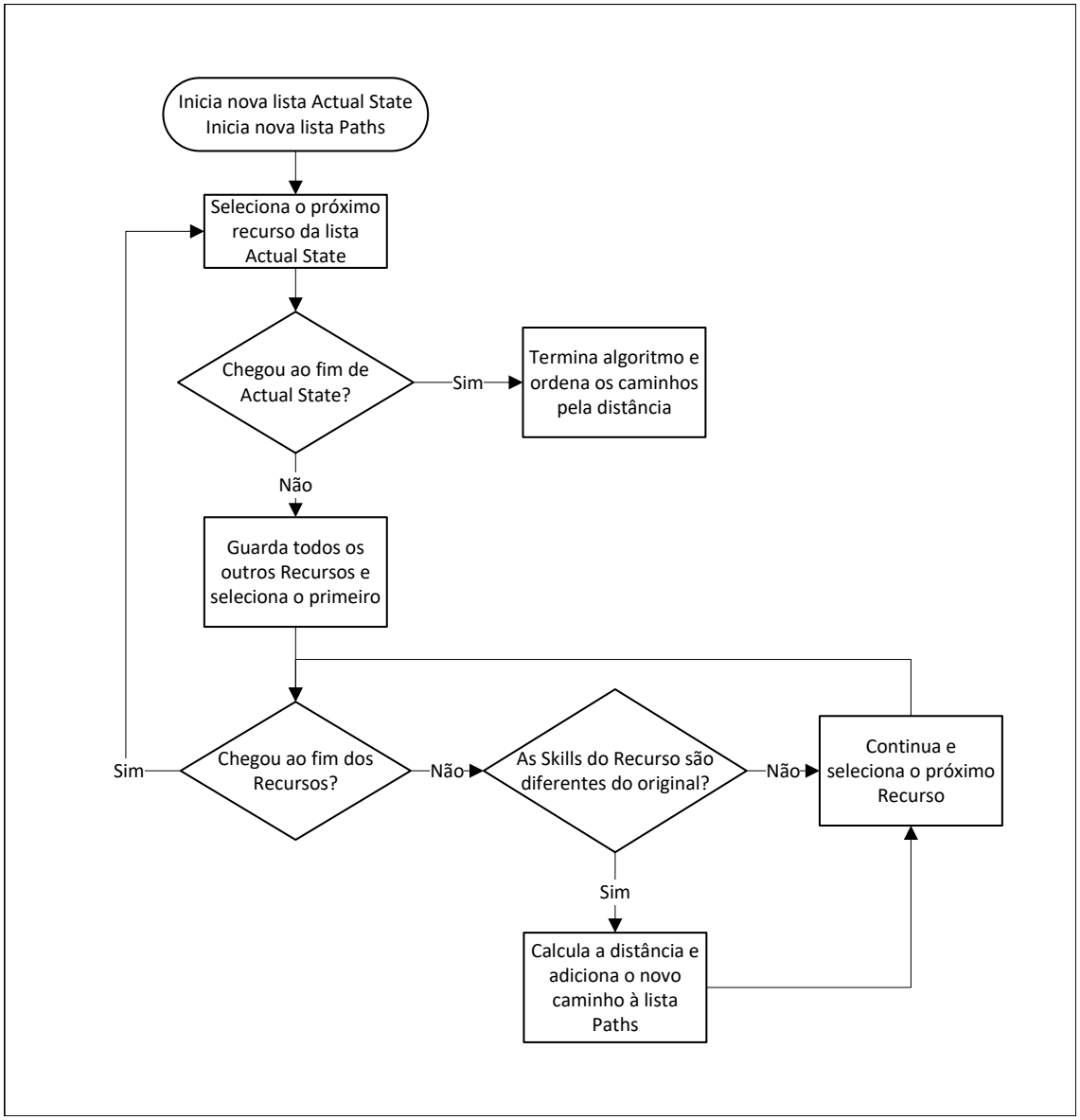

Figura 5.18: Fluxograma representativo do comportamento do algoritmo Básico

que só acaba quando o nó selecionado for o objectivo, que por outras palavras significa que encontrou a solução.

No inicio do comportamento cíclico, o nó selecionado é marcado como "explorado"e de seguida efetua-se a "exploração"desse nó. Na fase de exploração, o algoritmo utiliza a lista criada no inicio para encontrar os recursos capazes de executar a *skill* associada à profundidade definida. Estes recursos ou nós são guardados para que a seguir se determinem os valores utilizados pelo algoritmo. Para cada nó guardado, verifica-se a sua existência na lista de nós do algoritmo. Caso não exista é criado um novo nó sucessor com os seguintes dados:

- **node** - Identificação do recurso/nó sucessor guardado;
- **pos_node** - Posição do recurso/nó sucessor;
- **predecessor** - Identificação do recurso/nó antecessor, ou seja, o nó selecionado ao qual foi feita a "exploração";
- **pos_predecessor** - Posição do nó antecessor;
- **explored** - Informa se o recurso/nó foi expandido/explorado. Como se trata de um nó novo, está predefinido com o valor "FALSE";
- **current_cost** - Custo acumulado do caminho até este recurso, ou seja, o custo de chegar ao nó anterior, mais o custo do caminho até ao nó atual;
- **heuristic_cost** - Custo estimado para chegar ao objectivo a partir do nó atual;
- **sum_of_costs** - Soma dos custos **current_cost** e **heuristic_cost**;
- **depth** - Define a profundidade do nó e por conseguinte, a *skill* de que o nó está encarregue de executar.

No caso de já existir um nó na lista de nós do algoritmo, este compara os valores **current_cost** de ambos. Se o valor do nó já existente for menor, ele ignora o novo e passa para o recurso/nó seguinte. Caso contrário, são alterados todos os parâmetros para os do novo nó e marcado como não explorado. Depois de inseridos os recursos/nós obtidos da exploração, é selecionado o nó da lista de nós do algoritmo que possua o menor valor de **sum_of_costs**, para futura exploração.

Se este último nó selecionado for o objectivo, o comportamento cíclico é interrompido e é criada a solução através dos nós antecessores ligados. Esta solução é colocada na lista que será inserida na base de dados e repetem-se os comportamentos anteriores para cada tipo de produto existente. Após o algoritmo ter percorrido todos os tipos de produtos, são colocados os restantes caminhos possíveis entre recursos, ordenados consoante a sua distancia. Em 5.1 encontra-se o pseudo-código utilizado para criar este algoritmo e de seguida é também apresentado um fluxograma a representar o seu comportamento na Figura 5.19.

Listagem 5.1: Pseudo-Código utilizado para implementar o algoritmo **A***

```
GET list_basic_algorithm
GET products_list
new AStar_list

for each product from products_list
    new AStar_list
    insert start_node in AStar_list
    node_actual = start_node
    skill_depth++
    while(!solution_found)
        node_actual is explored
        search successor_nodes that execute
            products_list.productX.skill(skill_depth)
        for each successor_node found
            if(successor_node already in AStar_list)
                if(cost of node in AStar_list > cost successor_node)
                    change data
                    predecessor node of successor_node = node_actual
                    mark as unexplored
                else
                    continue to next successor_node
                endif
            else
                create new node
            endif
        endfor

        find and select next best node_actual

        if(node_actual == OBJECTIVE)
            solution_found = TRUE
            create and add solution to database_list
        endif
    endwhile
endfor

add remaining paths
write database_list in Next_Steps
```

Há que salientar que a heurística utilizada no algoritmo é de extrema importância para a optimalidade do mesmo. Neste caso, a heurística utilizada é tanto admissível como consistente, o que garante que o nosso algoritmo devolva uma solução ótima.

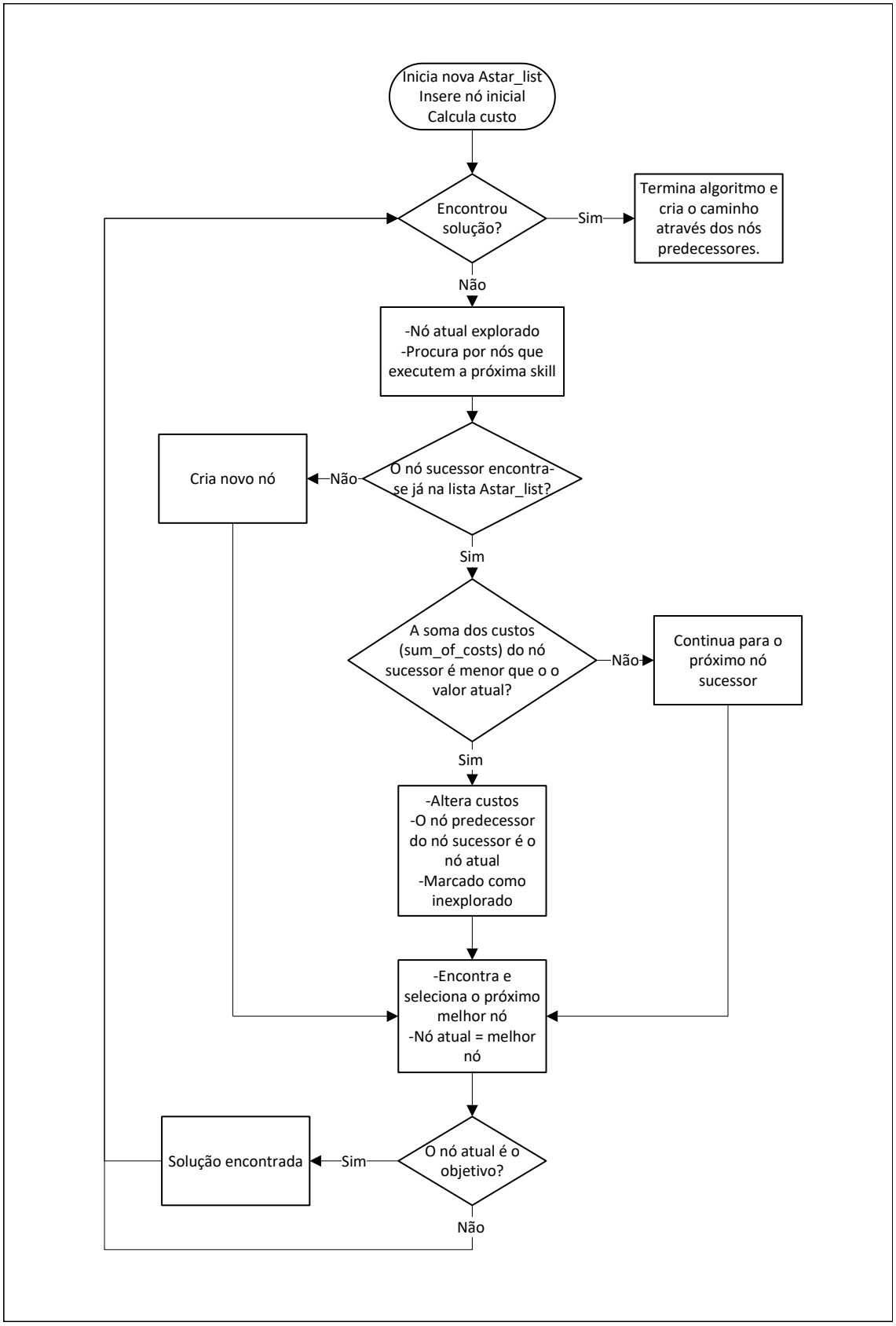

Figura 5.19: Fluxograma representativo do comportamento do algoritmo A*

6 Testes e Validação

Neste capítulo são exibidos os testes realizados e resultados obtidos da avaliação de desempenho e validação da arquitetura MAS, cuja implementação foi apresentada no capítulo anterior. São realizados alguns testes à arquitetura com intuito de avaliar o seu desempenho e robustez. De seguida ambos os algoritmos implementados são expostos a problemas mais complexos, de onde são extrapolados, exibidos, analisados e comentados os resultados obtidos em todos os testes, realçando as diferenças entre algoritmos.

6.1 Cenário de Teste

Para realizar os testes sobre arquitetura implementada, foi criada uma interface gráfica que contém todas as funções necessárias básicas de manipulação da arquitetura. Na Figura 6.1 podemos observar a interface criada, esta interface está dividida em três partes: uma parte de seleção de algoritmos, uma parte para inserção de produtos, e uma parte para manipulação de recursos. Também podemos observar um botão que lança o *script* para executar testes mais complexos que envolvam todas as funções existentes. Todas as funções presentes na interface são executadas em tempo real e podem ser executadas em qualquer altura na execução da arquitetura.

Para simular o chão da fábrica foi definido uma matriz de dimensões de 11x11. Esta matriz não tem quaisquer restrições quanto á sua dimensão e pode tomar as dimensões que se desejar, mas neste contexto de testes foi considerado como o suficiente. No chão da fábrica podemos encontrar recursos posicionados segundo coordenadas X e Y. Antes da execução da interface gráfica, o chão da fábrica não possui qualquer recurso, mas ao executarmos a interface, a fábrica é iniciada e são lançados recursos com ela. São inseridos alguns recursos padrão em posições diversas, que garantem o mínimo para que a arquitetura funcione, e os responsáveis pela entrada e saída de produtos, que se encontram posicionados no centro da matriz (Figura 6.2).

Com excepção dos recursos responsáveis pela entrada e saída dos produtos, podem ser removidos e inseridos quaisquer outros recursos que executem as *skills* conhecidas. Nesta arquitetura foram implementadas cinco *skills*: `Entry`, `Exit`, `Cut`, `Drill` e `Brush`. Das *skills* implementadas, `Entry` e `Exit` são de uso exclusivo pelos recursos responsáveis pela entrada e saída dos produtos(`InsertPieces` e `RemovePieces`, respectivamente), não podendo ser utilizadas por outros recursos. Todos os outros recursos podem ser inseridos

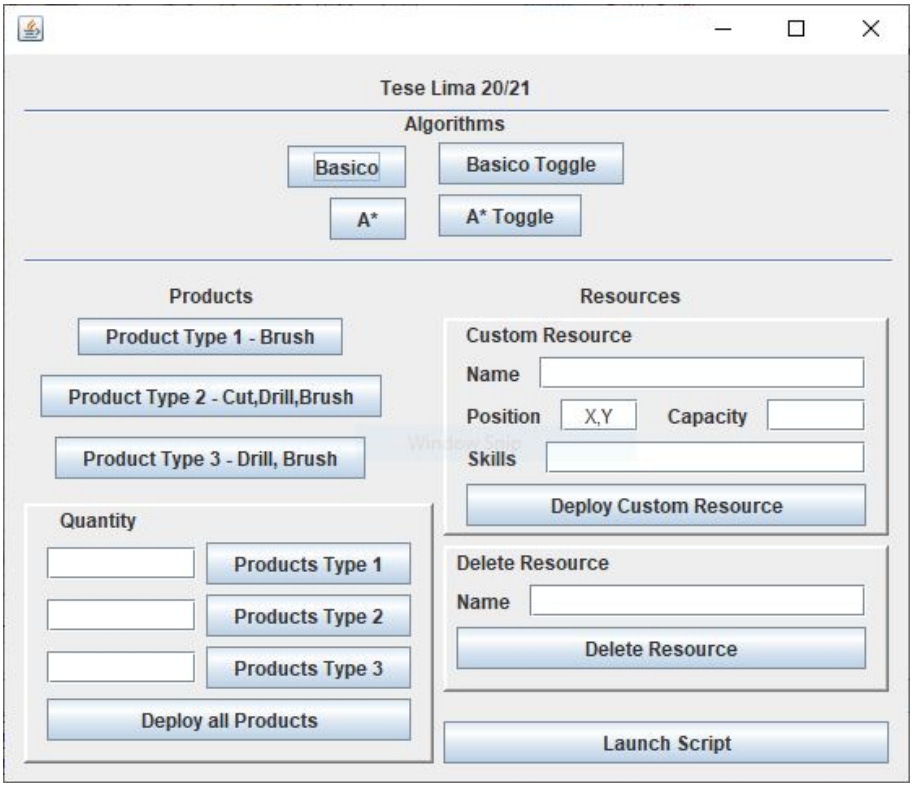

Figura 6.1: Interface gráfica criada para testes na arquitetura

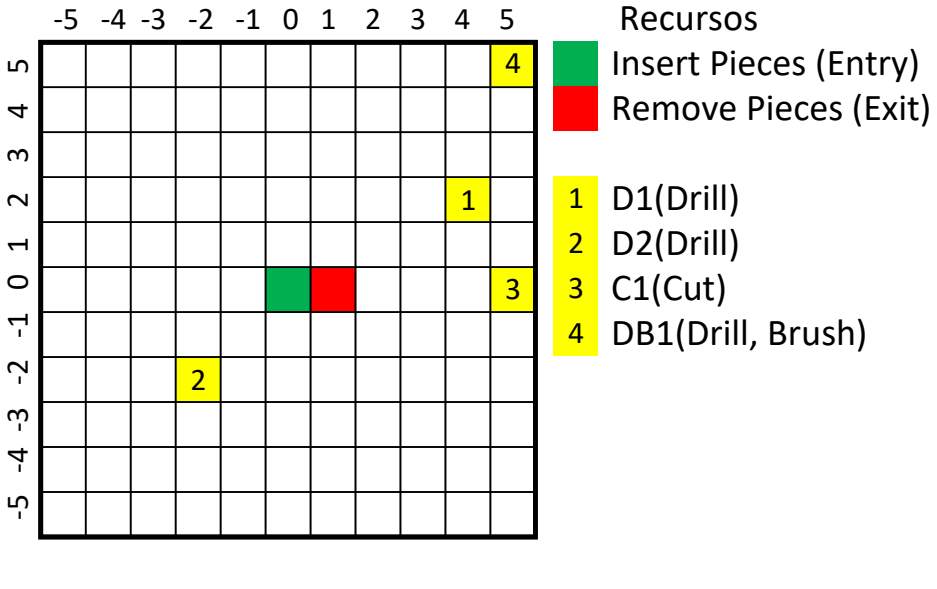

Figura 6.2: Chão da fábrica após sua iniciação

Nome Produto	Skills
Product Type 1	Entry, Brush, Exit
Product Type 2	Entry, Cut, Drill, Brush, Exit
Product Type 3	Entry, Drill, Brush, Exit

Tabela 6.1: Produtos utilizados nos testes

ou removidos com as restantes *skills* disponíveis.É possível adicionar uma maior quantidade de *skills*, desde que estas sejam definidas na ontologia, com a respectiva prioridade associada e sejam alteradas as proteções e filtros nos produtos e recursos.

Relativamente aos produtos e que se pode deduzir da interface na Figura 6.1, existem três tipos: `Type 1`, `Type 2` e `Type 3`.A diferença entre os produtos encontra-se na quantidade e na diversidade das *skills* na lista que executam e, como acontece com os recursos, é possível criarem-se mais tipos de produtos. Todos os produtos possuem ambas as *skills* `Entry` e `Exit`, respectivamente, nas extremidades da sua lista, enquanto as restantes se encontram no meio. Estas *skills* são executadas sequencialmente, respeitando as suas prioridades, previamente definidas na ontologia. Os produtos utilizados para teste são apresentados na tabela seguinte (Tabela 6.1) com as suas respectivas *skills*.

Todos os testes foram executados através do IDE NetBeans, com as bases de dados a serem geridas pelo programa XAMPP com interface no browser Firefox. A máquina onde decorreram estes testes foi um portátil Samsung, a correr o sistema operativo Windows 10, com processador Intel Core i5-3210M (dual-core a 2.5GHz), 16GB de memória (2x8GB DDR3 800MHz) e dois discos: um SSD de 120GB, onde se encontram instalados o NetBeans e o XAMPP, e um HDD de 500GB onde se encontram guardados os ficheiros do projeto. Em todos os testes realizados, tentou-se minimizar o uso de recursos por parte de outros programas que pudessem estar a correr em segundo plano e consequentemente impactar negativamente os resultados obtidos.

6.2 Testes e Resultados

De forma a avaliar o desempenho da arquitetura, começou-se com testes simples para registar o tempo médio que demoram os produtos a executar todas as suas habilidades e o percurso que executam. Foram depois realizados testes mais complexos com inserção e remoção de recursos.

O primeiro teste foi realizado com o sistema ausente de carga, ou seja, cada produto foi lançado individualmente, com intervalos fixos entre eles. Foram executados no total 300 produtos (100 para cada tipo) para cada um dos algoritmos. Na tabela 6.2 e no gráfico 6.3, podemos aferir que, para cada produto, os algoritmos não tiveram qualquer impacto significativo nos tempos e que estes se mantiveram praticamente sempre constantes. As diferenças nos tempos de cada produto é influenciada apenas pela quantidade de *skills* que este tem de executar. Visto que foi definido uma duração de 100 milissegundos para

	Tempo execução (ms)	
Tipo Produto	Algoritmo Básico	Algoritmo A*
Type 1	119.36	117.35
Type 2	318.14	316.91
Type 3	220.25	219.91

Tabela 6.2: Tempo médio de execução dos produtos realizados sem carga

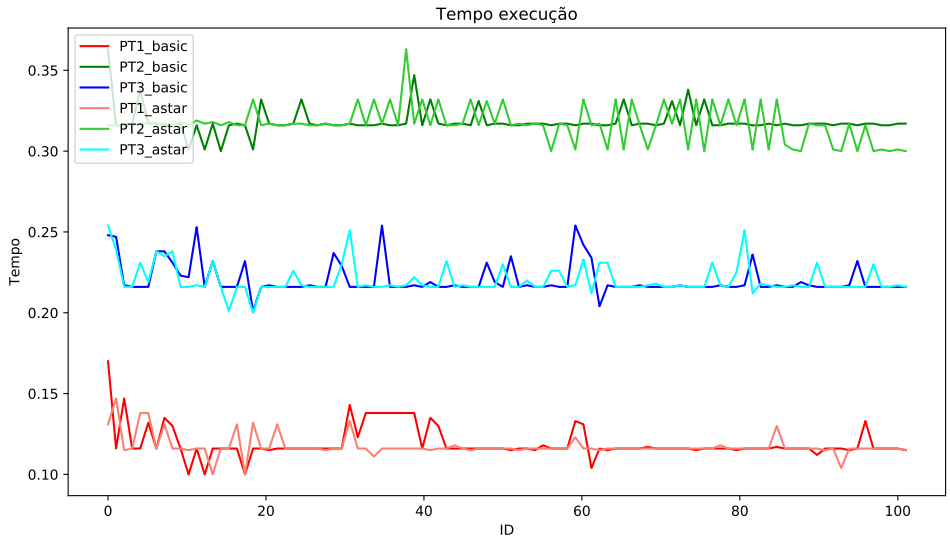

Figura 6.3: Evolução do tempo de execução dos produtos realizados sem carga

um recurso executar uma *skill*, podemos observar que o tempo varia de forma espectável e linear em reflexo do número de *skills*.

Embora o tempo de execução não sofra impacto, existem diferenças entre os algoritmos. Dependendo do algoritmo utilizado, durante os testes foi possível observar que os caminhos que os produtos optaram por seguir, foram diferentes. Nas figuras 6.4 e 6.5 podemos ver as diferenças. Quando o algoritmo utilizado é o **A***, este vai utilizar menos recursos e percorrer percursos muito menores comparativamente com o algoritmo **Básico**. Embora nesta implementação as distâncias não tenham impacto na duração, são de extrema importância num caso real e decerto que terão consequências na execução do produto.

No teste seguinte o sistema foi colocado sobre carga. Foram lançados 1000 produtos em simultâneo para cada tipo e algoritmo, registando-se o tempo de execução de cada produto, assim como os caminhos que fizeram. Na tabela 6.3 e gráfico 6.6 mostram-se os resultados obtidos e podemos observar que existem diferenças nos algoritmos, em particular no produto Type 3.

Os caminhos feitos pelos produtos também sofreram alterações. Visto que os recursos se encontram ocupados e o máximo que conseguem processar são 10 produtos por

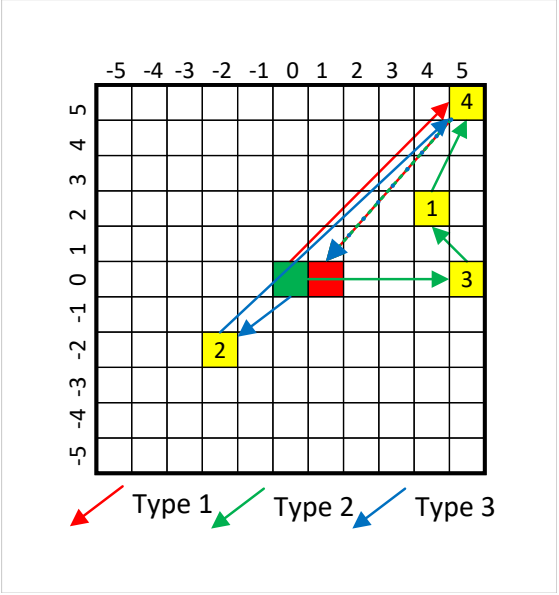

Figura 6.4: Caminho com algoritmo Básico

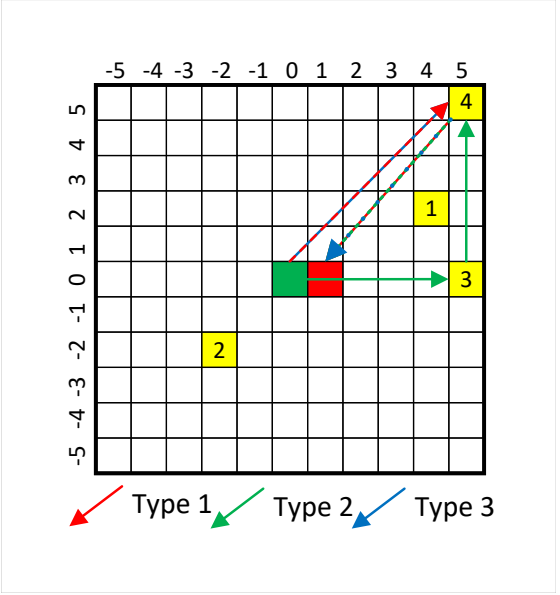

Figura 6.5: Caminho com algoritmo A*

	Tempo execução (s)	
Tipo Produto	Algoritmo Básico	Algoritmo A*
Type 1	67.01	67.74
Type 2	160.99	179.39
Type 3	101.56	144.28

Tabela 6.3: Tempo médio de execução dos produtos realizados com carga

segundo (devido aos 100 milissegundos de demora na realização de cada *skill*), estes começam a avisar os produtos para se dirigirem para outros recursos, se for possível, ou então ficam em lista de espera. Os caminhos "preferidos"das peças continuam a ser, dependendo do algoritmo, os que estão ilustrados nas figuras 6.4 e 6.5, mas, devido à sobrecarga nos recursos, alguns produtos desviam-se desse caminho e tentam realizar as suas *skills* na segunda, ou outra, melhor opção que os algoritmos fornecerem. Estes "desvios"foram registados e são apresentados na tabela 6.4, com as percentagens de utilização em cada recurso por cada *skill* executada.

	Recursos Usados					
	Algoritmo Básico			**Algoritmo A***		
Tipo Produto	**Skill 1**	**Skill 2**	**Skill 3**	**Skill 1**	**Skill 2**	**Skill 3**
Type 1	DB1 - 100%	-	-	DB1 - 100%	-	-
Type 2	C1 - 100%	D1 - 89.8% DB1 - 8.5% D2 - 1.7%	DB1 - 100%	C1 - 100%	DB1 - 89.5% D1 - 10.4 D2 - 0.1%	DB1 - 100%
Type 3	D2 - 77.8% D1 - 22.2%	DB1 - 100%	-	DB1 - 80% D2 - 20%	DB1 - 100%	-

Tabela 6.4: Utilização de Recursos no teste em carga

Em comparação com os testes sem carga, estes apresentam um crescimento no tempo

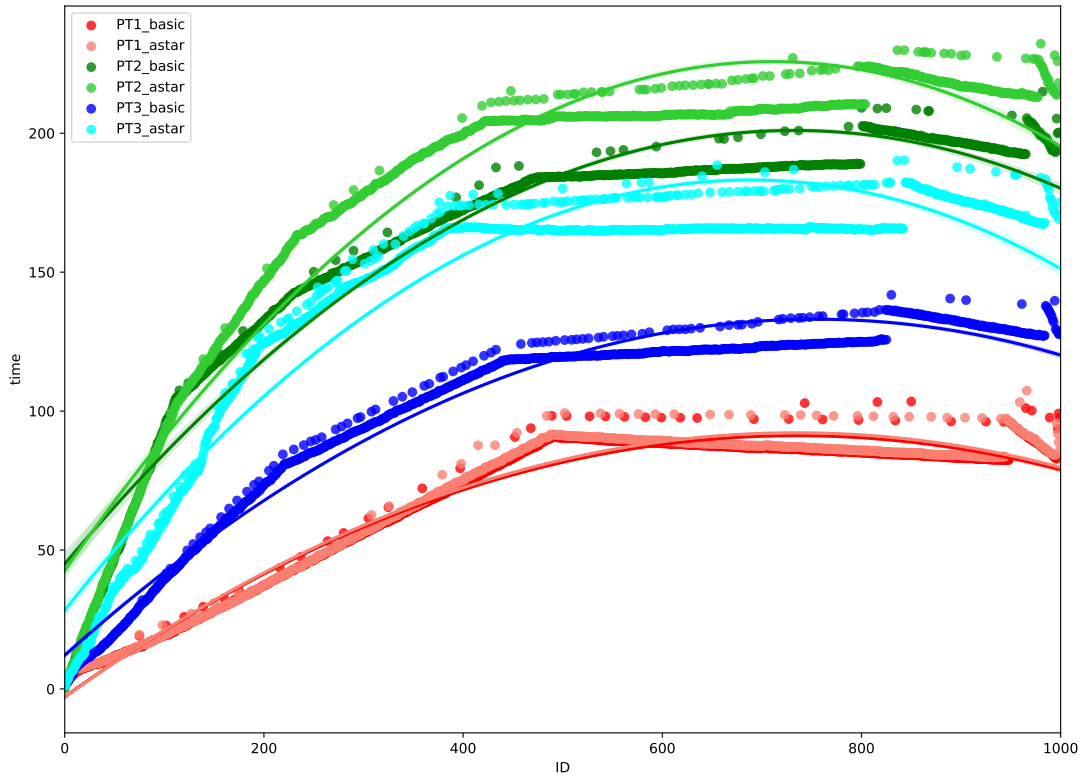

Figura 6.6: Evolução do tempo de execução dos produtos realizados com carga

semelhante a uma função logarítmica que começa a estabilizar à medida que os produtos saem do sistema. Isto acontece devido ao número limitado de recursos para cada *skill*, que obriga os produtos a ficarem à espera dos únicos recursos capazes de a executar, e também pela velocidade a que estes as executam.

Repetiu-se o teste em carga para 1000 produtos Type 2 com o algoritmo **A***, mas agora com mais um recurso no sistema capaz de executar as *skills* Cut e Brush. Este recurso permitiu que os produtos pudessem sofrer uma melhor distribuição pelos recursos e teve um impacto significativo no tempo médio de execução dos 1000 produtos. Com este novo recurso, o tempo médio de execução passou dos 179.39s para 150.49s. Uma diferença de 29 segundos.

Portanto, existe uma necessidade de haver um equilíbrio entre as *skills* mais requisitadas e o número de recursos que as executem.

O motivo da discrepância dos tempos de execução para os produtos Type 3 ser tão elevada é, para além do que foi dito aqui antes, consequência do caminho "preferido"que os algoritmos dizem aos produtos para realizar. Embora o algoritmo **A*** indique o menor caminho existente para realizar o produto, este não tem em conta a utilização dos recursos, o que pode levar a uma utilização excessiva de um único recurso. Em produtos Type 3 é visível este problema, visto que, com o **A***, o recurso DB1 seja utilizado duas vezes seguidas, ou seja, o produto após realizar a primeira *skill*, volta para o final da fila à

espera de executar a seguinte que só este recurso é capaz de realizar, entupindo todo o fluxo, enquanto no algoritmo **Básico** este seja utilizado apenas uma vez.

Realizou-se outro teste, o mais complexo, no qual o sistema correu um script para simular um caso experimental perto do real, com inserção constante de peças, adição de um recurso, remoção de um recurso e alteração do algoritmo, tudo enquanto o sistema se encontra em execução. Para este teste o tempo que os recursos demoram a executar uma *skill* foi incrementado para meio segundo.

O script começa com a execução do algoritmo **Básico**, depois lança 15 produtos (5 de cada tipo), lançando de novo 15 produtos a cada 5 segundos até terem sido lançados no total 60 produtos. Após o primeiro lançamento de produtos, é inserido o novo recurso, o algoritmo só volta a correr passado 5 segundos depois da sua entrada. Executa-se o algoritmo e passados 5 segundos alteramos o algoritmo para o **A***. Decorridos outros 5 segundos removemos o recurso `DB1` do sistema. Por último e outra vez passados 5 segundos da última instrução, voltamos a correr o algoritmo **Básico** e deixamos a peças acabarem as suas execuções.

Para o teste, o script foi executado quatro vezes. No total o sistema passou por cinco fases e para cada uma destas fases foram registados os percursos e tempos médios que os produtos demoraram a completar nesses intervalos. A seguir para cada fase são exibidos e comentados os resultados obtidos.

1. **Recursos padrão -> Algoritmo Básico -> 15 produtos -> Novo recurso**
   Nesta fase o programa corre normalmente como se apenas tivesse os recursos padrão conforme na Figura 6.2. Visto que um recurso é inserido após o algoritmo ter corrido, embora este esteja registado na base de dados **Actual State**, não é usado pelos produtos, porque ainda não foram descobertos os caminhos pelo algoritmo. Durante esta fase, todos os produtos que conseguiram acabar as suas *skills* foram dos tipos `Type 1` ou `Type 3`, por possuírem a menor quantidade delas, não houve nenhum produto `Type 2` que conseguisse acabar na primeira fase. Os tempos médios obtidos estão na tabela 6.5. O caminho que os produtos tomaram estão de acordo com o que

	Tempo execução (s)	
Tipo Produto	Algoritmo Básico	Total produtos
Type 1	2.737	19
Type 2	-	0
Type 3	4.436	4

Tabela 6.5: Tempo médio de execução dos produtos durante a primeira fase

   foi apresentado na Figura 6.4.

2. **Algoritmo Básico -> 15 produtos**
   Na segunda fase, antes de serem lançados novos produtos, corremos o algoritmo **Básico**. Agora os novos caminhos dos produtos incluem o quinto recurso colocado

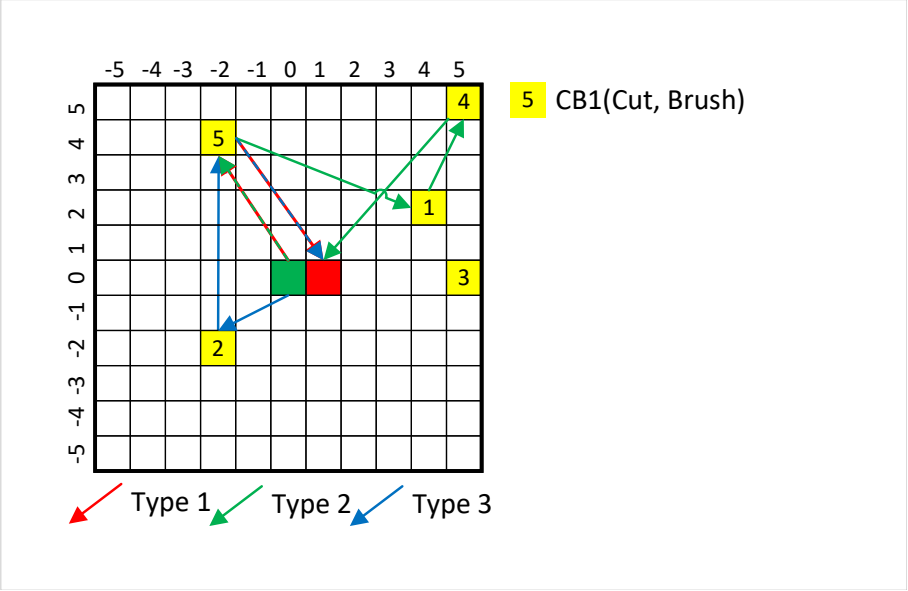

Figura 6.7: Caminhos preferidos após execução do algoritmo com novo recurso

anteriormente, passando agora a executar os caminhos representados na Figura 6.7. A alteração dos caminhos nos produtos só começa a fazer efeito nos produtos que foram lançados recentemente, ou no caso da próxima *skill* a executar passar pelo novo recurso. No entanto, nos produtos mais antigos, devido ao facto de já só estarem a executar as suas últimas *skills*, não apresentam alterações no caminho. Os tempos médios obtidos de cada produto durante esta fase, estão na tabela seguinte 6.6.

	Tempo execução (s)	
Tipo Produto	Algoritmo Básico	Total produtos
Type 1	3.88	4
Type 2	7.7	17
Type 3	6.19	15

Tabela 6.6: Tempo médio de execução dos produtos durante a segunda fase

3. **Algoritmo A* -> 15 produtos**
   Aqui executamos o algoritmo **A*** e lançamos novamente 15 produtos. Pelo mesmo motivo na fase anterior, os produtos em que se observam alterações nos caminhos, apenas ocorre nos produtos mais recentes. Tanto para os produtos Type 1 e Type 3, os caminhos entre os algoritmos permanecem iguais, os únicos afetados são os produtos Type 2. Agora os caminhos preferidos são os que aparecem representados na Figura 6.8 e os tempos médios de execução obtidos encontram-se na tabela 6.7.

4. **Remoção recurso DB1 -> 15 produtos**
   Nesta fase procede-se à remoção do recurso DB1 (representado com o número 4) e

	Tempo execução (s)	
Tipo Produto	Algoritmo A*	Total produtos
Type 1	6.69	24
Type 2	10.63	17
Type 3	8.19	21

Tabela 6.7: Tempo médio de execução dos produtos durante a terceira fase

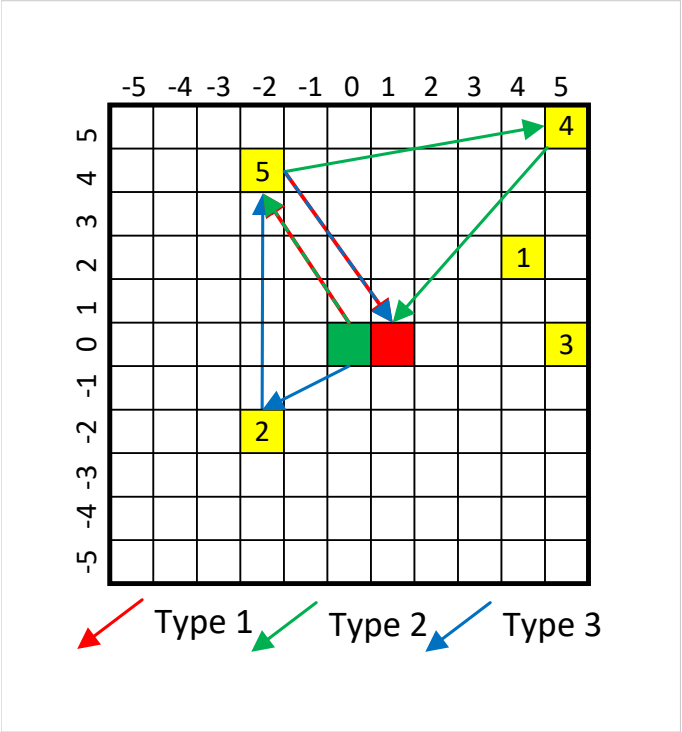

Figura 6.8: Caminhos após execução do algoritmo A*

são depois lançados os últimos produtos para o sistema. Durante esta fase não se executa qualquer algoritmo, ou seja, na tabela dos **Next Steps** os caminhos indicam aos produtos para usar um recurso que não existe. Ao contrário dos algoritmos, o impacto da remoção do recurso é imediato e o sistema reage automaticamente à remoção do recurso, procurando pelo recurso mais próximo do que foi removido, que seja capaz de executar a devida *skill*, descrevendo novos caminhos representados na Figura 6.9. No caso de não existir um recurso capaz de executar as *skills* pedidas, o produto é removido do sistema, inacabado. Os tempos médios registados durante esta fase foram colocados na tabela 6.8.

5. **Algoritmo Básico**
   Na última fase executamos novamente o algoritmo **Básico**, desta vez sem inserir novas peças e deixamos o programa acabar de executar as restantes peças presentes no sistema. Após correr o algoritmo **Básico** os caminhos ficam corretamente definidos e os produtos já não são encaminhados para um recurso inexistente. Por

	Tempo execução (s)	
Tipo Produto	Algoritmo A*	Total produtos
Type 1	7.86	12
Type 2	12.93	7
Type 3	8.69	19

Tabela 6.8: Tempo médio de execução dos produtos durante a quarta fase

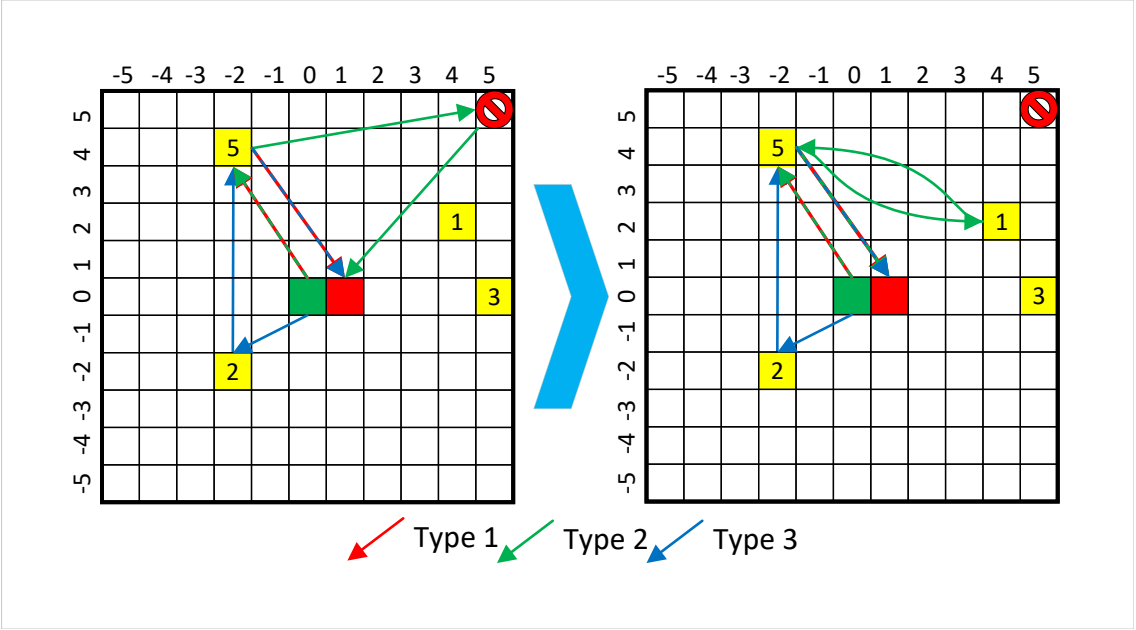

Figura 6.9: Transição dos caminhos após remoção do recurso DB1

coincidência os novos caminhos são iguais aos que os produtos automaticamente tomavam como ilustrados na Figura 6.9. Novamente, os tempos médios da execução dos produtos foram registados e colocados na tabela seguinte 6.9.

	Tempo execução (s)	
Tipo Produto	Algoritmo Básico	Total produtos
Type 1	8.83	21
Type 2	13.43	38
Type 3	10.71	21

Tabela 6.9: Tempo médio de execução dos produtos durante a quinta fase

Com o evoluir das fases e analisando todas as tabelas 6.5, 6.6, 6.7, 6.8 e 6.9, é possível verificar um incremento nos tempos médios. Isto deve-se a um problema semelhante que já foi discutido anteriormente neste capítulo, que é de o sistema não conseguir escoar os produtos à mesma velocidade com que entram.

7 Conclusão e Trabalho Futuro

7.1 Conclusão

Com todo o desenvolvimento e pesquisa contidos nos capítulos anteriores, o trabalho deu boas indicações de que a abordagem pode ser explorada de modo a criar uma arquitetura, que implementa um sistema de controlo distribuído auto-organizado, onde podemos fazer a integração de vários algoritmos diferentes.

A possibilidade de se conseguir integrar qualquer algoritmo, permite que a arquitetura se possa ajustar aos problemas levantados no sistema de produção, garantindo a melhor otimização do sistema. Devido à arquitetura estar dividida em camadas, o algoritmo utilizado poderá ser futuramente reconfigurado, ou até completamente substituído por outro, sem a necessidade de modificar as restantes camadas.

Pelos testes que foram executados sobre o sistema implementado com base na arquitetura desenvolvida, apresentados no capítulo anterior 6, é comprovada a sua robustez face a um número elevado de produtos em simultâneo, embora acarrete outras desvantagens. A capacidade de se poder introduzir um qualquer número de recursos com diferentes habilidades (na literatura conhecidas como *skills*), concede à arquitetura flexibilidade e modularidade. Nestes testes, também é validada a aptidão da arquitetura reagir a perturbações na configuração do sistema, através dos comportamentos independentes dos produtos, demonstrando as suas capacidades de auto-organização.

A nível de desempenho, foi possível constatar que o tempo de execução de cada produto, embora varie linearmente com a quantidade de habilidades que este tem a realizar, está fortemente correlacionado com a velocidade de processamento de cada recurso, quantidade de recursos e produtos introduzidos no sistema. Estas são limitações físicas que se deverão ter em conta, caso a arquitetura seja aplicada num sistema de grande escala e se pretenda o rápido processamento de um avultado número de produtos.

7.2 Trabalho Futuro

Novos desenvolvimentos com base nesta arquitetura devem ter como objetivos principais os comportamentos e comunicações dos agentes, de modo a diminuir a dependência na camada das bases de dados e melhorar o fluxo dos produtos para outros recursos menos sobrecarregados.

Ao nível das bases de dados, a base de dados **Next Steps** poderá estar dividida em várias tabelas, estando cada uma delas associada a um tipo de produto. Isto permitiria uma melhor otimização dos percursos, uma procura mais rápida por parte dos produtos e eventualmente um uso mais optimizado do algoritmo. Poderão também existir mais bases de dados com outras informações ou propósitos.

Na camada dos algoritmos, poder-se-à desenvolver a sua comunicação com as bases de dados, para se usarem algoritmos mais avançados. Os algoritmos aqui presentes também podem ter outras funções a desempenhar para além da otimização dos caminhos dos produtos, como, por exemplo, calcular a melhor posição para colocar um novo recurso.

Na arquitetura implementada não foi tida em conta todo o sistema responsável pelo transporte, pelo que, se for implementada, poderá trazer benefícios ao sistema em robustez, autonomia e uma maior distribuição no controlo.

Bibliografia

[1] M. Hermann, T. Pentek e B. Otto. "Design Principles for Industrie 4.0". Em: *Technische Universität Dortmund Fakultät Maschinenbau Audi Stiftungslehrstuhl Supply Net Order Management* 01 (2015), p. 15. URL: https://www.researchgate.net/publication/307864150_Design_Principles_for_Industrie_40_Scenarios_A_Literature_Review%20http://www.snom.mb.tu-dortmund.de/cms/de/forschung/Arbeitsberichte/Design-Principles-for-Industrie-4_0-Scenarios.pdf (ver p. 1).

[2] H. Lasi et al. "Industry 4.0". Em: *Business & Information Systems Engineering* 6.4 (ago. de 2014), pp. 239–242. ISSN: 1867-0202. DOI: 10.1007/s12599-014-0334-4. URL: http://link.springer.com/10.1007/s12599-014-0334-4 (ver p. 1).

[3] T. Sanislav e L. Miclea. *Cyber-Physical Systems-Concept, Challenges and Research Areas*. Rel. téc. 2. 2012, pp. 28–33 (ver p. 2).

[4] R. Rajkumar et al. "Cyber-physical systems: The next computing revolution". Em: *Proceedings - Design Automation Conference*. New York, New York, USA: ACM Press, 2010, pp. 731–736. ISBN: 9781450300025. DOI: 10.1145/1837274.1837461. URL: http://portal.acm.org/citation.cfm?doid=1837274.1837461 (ver p. 2).

[5] L. Ribeiro. "Cyber-physical production systems' design challenges". Em: *IEEE International Symposium on Industrial Electronics*. Edinburgh, UK: IEEE, jun. de 2017, pp. 1189–1194. ISBN: 9781509014125. DOI: 10.1109/ISIE.2017.8001414. URL: https://ieeexplore.ieee.org/abstract/document/8001414/%20http://ieeexplore.ieee.org/document/8001414/ (ver pp. 2, 15).

[6] K. P. Sycara. "Multi-agent systems". Em: *AI Magazine Volume 19 Number 2* 19 (1998), pp. 79–92. URL: https://www.aaai.org/ojs/index.php/aimagazine/article/view/1370%20http://ieeexplore.ieee.org/document/1372753/ (ver p. 3).

[7] G. Weiss. "Multi Agent Systems: A Modern Approach to Distributed Artificial Intelligence". Em: (1999) (ver p. 3).

[8] S. Karnouskos et al. "Industrial Agents as a Key Enabler for Realizing Industrial Cyber-Physical Systems: Multiagent Systems Entering Industry 4.0". Em: *IEEE Industrial Electronics Magazine* 14.3 (set. de 2020), pp. 18–32. ISSN: 1932-4529.

DOI: 10.1109/MIE.2019.2962225. URL: https://ieeexplore.ieee.org/document/9205563/ (ver p. 3).

[9] B. Vogel-Heuser, J. Lee e P. Leitão. *Agents enabling cyber-physical production systems*. Jan. de 2015. DOI: 10.1515/auto-2014-1153. URL: http://bibliotecadigital.ipb.pt/handle/10198/16202%20https://www.degruyter.com/doi/10.1515/auto-2014-1153 (ver pp. 3, 20).

[10] W. Banzhaf. "Self-organizing Systems". Em: *Encyclopedia of Complexity and Systems Science*. New York, NY: Springer New York, 2009, pp. 8040–8050. DOI: 10.1007/978-0-387-30440-3_475. URL: https://link.springer.com/referenceworkentry/10.1007/978-0-387-30440-3_475%20http://link.springer.com/10.1007/978-0-387-30440-3_475 (ver pp. 3, 20).

[11] B. Dimov. *Industry 4.0 introduced by MOS Consult and only in DMT magazine 05/06*. URL: https://mos-consult.com/en/news/publications/37-industry-4-0-introduced-by-mos-consult-and-only-in-dmt-magazine-5-en (acedido em 22/12/2020) (ver p. 8).

[12] K. Zhou, Taigang Liu e Lifeng Zhou. "Industry 4.0: Towards future industrial opportunities and challenges". Em: *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. IEEE, ago. de 2015, pp. 2147–2152. ISBN: 978-1-4673-7682-2. DOI: 10.1109/FSKD.2015.7382284. URL: http://ieeexplore.ieee.org/document/7382284/ (ver p. 7).

[13] S. Hu et al. "Assembly system design and operations for product variety". Em: *CIRP Annals* 60.2 (2011), pp. 715–733. ISSN: 00078506. DOI: 10.1016/j.cirp.2011.05.004. URL: http://ees.elsevier.com/cirp/default.asp%20https://linkinghub.elsevier.com/retrieve/pii/S000785061100206X (ver p. 8).

[14] A. Azwan Abdul Rahman. "Revolution of Production System for the Industry 4.0". Em: *Mass Production Processes*. IntechOpen, mar. de 2020. DOI: 10.5772/intechopen.90772. URL: www.intechopen.com%20https://www.intechopen.com/books/mass-production-processes/revolution-of-production-system-for-the-industry-4-0 (ver pp. 8–10).

[15] R. Frei, J. Barata e M. Onori. "Evolvable Production Systems Context and Implications". Em: *2007 IEEE International Symposium on Industrial Electronics*. IEEE, jun. de 2007, pp. 3233–3238. ISBN: 978-1-4244-0754-5. DOI: 10.1109/ISIE.2007.4375132. URL: http://ieeexplore.ieee.org/document/4375132/ (ver pp. 9, 15).

[16] H. A. ElMaraghy. "Flexible and reconfigurable manufacturing systems paradigms". Em: *International Journal of Flexible Manufacturing Systems* 17.4 (out. de 2005), pp. 261–276. ISSN: 0920-6299. DOI: 10.1007/s10696-006-9028-7. URL: http://link.springer.com/10.1007/s10696-006-9028-7 (ver pp. 9, 10).

[17] J. P. Shewchuk e C. L. Moodie. *Definition and Classification of Manufacturing Flexibility Types and Measures*. Rel. téc. 1998, pp. 325–349 (ver p. 9).

[18] L. Monostori et al. "Cyber-physical systems in manufacturing". Em: *CIRP Annals* 65.2 (2016), pp. 621–641. ISSN: 00078506. DOI: 10.1016/j.cirp.2016.06.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0007850616301974 (ver p. 10).

[19] J. Barata e L. M. Camarinha-Matos. "Coalitions of manufacturing components for shop floor agility - the CoBASA architecture". Em: *International Journal of Networking and Virtual Organisations* 2.1 (2003), pp. 50–77. ISSN: 14709503. DOI: 10.1504/IJNVO.2003.003518 (ver p. 10).

[20] A. Tharumarajah. "Comparison of the bionic, fractal and holonic manufacturing system concepts". Em: *International Journal of Computer Integrated Manufacturing* 9.3 (jan. de 1996), pp. 217–226. ISSN: 0951-192X. DOI: 10.1080/095119296131670. URL: http://www.tandfonline.com/doi/abs/10.1080/095119296131670 (ver pp. 10–13).

[21] K. Ueda. "A concept for bionic manufacturing systems based on DNA-type information". Em: *Human Aspects in Computer Integrated Manufacturing*. Elsevier, jan. de 1992, pp. 853–863. DOI: 10.1016/B978-0-444-89465-6.50078-8. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780444894656500788 (ver pp. 10, 13).

[22] A. Tharumarajah, A. Wells e L. Nemes. "Comparison of emerging manufacturing concepts". Em: *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*. Vol. 1. IEEE, 1998, pp. 325–331. ISBN: 0-7803-4778-1. DOI: 10.1109/ICSMC.1998.725430. URL: https://ieeexplore.ieee.org/abstract/document/725430/%20http://ieeexplore.ieee.org/document/725430/ (ver pp. 11–13).

[23] C. Christo e C. Cardeira. "Trends in Intelligent Manufacturing Systems". Em: *2007 IEEE International Symposium on Industrial Electronics*. IEEE, jun. de 2007, pp. 3209–3214. ISBN: 978-1-4244-0754-5. DOI: 10.1109/ISIE.2007.4375129. URL: https://ieeexplore.ieee.org/abstract/document/4375129/%20http://ieeexplore.ieee.org/document/4375129/ (ver pp. 11, 12).

[24] R. F. Babiceanu e F. F. Chen. "Development and Applications of Holonic Manufacturing Systems: A Survey". Em: *Journal of Intelligent Manufacturing* 17.1 (fev. de 2006), pp. 111–131. ISSN: 0956-5515. DOI: 10.1007/s10845-005-5516-y. URL: https://link.springer.com/content/pdf/10.1007/s10845-005-5516-y.pdf%20http://link.springer.com/10.1007/s10845-005-5516-y (ver p. 13).

[25] H. Van Brussel et al. "Reference architecture for holonic manufacturing systems: PROSA". Em: *Computers in Industry* 37.3 (nov. de 1998), pp. 255–274. ISSN: 01663615. DOI: 10.1016/S0166-3615(98)00102-X. URL: https://linkinghub.elsevier.com/retrieve/pii/S016636159800102X (ver p. 13).

[26] P. Leitão e F. Restivo. "ADACOR: A holonic architecture for agile and adaptive manufacturing control". Em: *Computers in Industry* 57.2 (fev. de 2006), pp. 121–130. ISSN: 01663615. DOI: 10.1016/j.compind.2005.05.005 (ver p. 14).

[27] C. Pach et al. "ORCA-FMS: a dynamic architecture for the optimized and reactive control of flexible manufacturing scheduling". Em: *Elsevier* (). URL: https://www.sciencedirect.com/science/article/pii/S0166361514000396 (ver p. 14).

[28] J. Barbosa et al. "Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution". Em: *Computers in Industry* 66 (jan. de 2015), pp. 99–111. ISSN: 01663615. DOI: 10.1016/j.compind.2014.10.011. URL: https://linkinghub.elsevier.com/retrieve/pii/S0166361514001894 (ver p. 14).

[29] J.-F. Jimenez et al. "Pollux: a dynamic hybrid control architecture for flexible job shop systems". Em: *International Journal of Production Research* 55.15 (ago. de 2017), pp. 4229–4247. ISSN: 0020-7543. DOI: 10.1080/00207543.2016.1218087. URL: https://www.tandfonline.com/doi/full/10.1080/00207543.2016.1218087 (ver p. 14).

[30] B. Lindberg, M. Onori e D. Semere. *Evolvable Production Systems: A Position Paper*. Rel. téc. Swedish Production Symposium, 2006. URL: https://www.researchgate.net/publication/228656209 (ver p. 15).

[31] L. Ribeiro et al. "Self-organization in automation - the IDEAS pre-demonstrator". Em: *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, nov. de 2011, pp. 2752–2757. ISBN: 978-1-61284-972-0. DOI: 10.1109/IECON.2011.6119747. URL: http://ieeexplore.ieee.org/document/6119747/ (ver p. 15).

[32] E. A. Lee. "Cyber Physical Systems: Design Challenges". Em: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, mai. de 2008, pp. 363–369. ISBN: 978-0-7695-3132-8. DOI: 10.1109/ISORC.2008.25. URL: http://ieeexplore.ieee.org/document/4519604/ (ver p. 15).

[33] J. Lee, B. Bagheri e H.-A. Kao. "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems". Em: *Manufacturing Letters* 3 (jan. de 2015), pp. 18–23. ISSN: 22138463. DOI: 10.1016/j.mfglet.2014.12.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S221384631400025X (ver pp. 15, 16).

[34] B. Lydon. *RAMI 4.0 Reference Architectural Model for Industrie 4.0*. 2019. URL: https://www.isa.org/intech-home/2019/march-april/features/rami-4-0-reference-architectural-model-for-industr (acedido em 11/12/2020) (ver p. 17).

[35] Plattform Industrie 4.0. *Plattform Industrie 4.0 RAMI4.0-a reference framework for digitalisation An Overview*. Rel. téc. Berlin: Plattform Industrie 4.0, 2016, p. 32. URL: www.plattform-i40.de/en (ver p. 17).

[36] M. Hankel. *Industrie 4.0: The reference architectural model industrie 4.0 (RAMI 4.0)*. Rel. téc. 2015, p. 2. URL: www.zvei.org/ (ver p. 17).

[37] N. R. Jennings e M. Wooldridge. *Applying Agent Technology: Intelligent Agents and Multi-Agent Systems*. Rel. téc. (ver p. 19).

[38] P. Leitão. "Agent-based distributed manufacturing control: A state-of-the-art survey". Em: *Engineering Applications of Artificial Intelligence* 22.7 (out. de 2009), pp. 979–991. ISSN: 09521976. DOI: 10.1016/j.engappai.2008.09.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197608001437 (ver p. 20).

[39] P. Leitão, A. W. Colombo e S. Karnouskos. "Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges". Em: *Computers in Industry* 81 (set. de 2016), pp. 11–25. ISSN: 01663615. DOI: 10.1016/j.compind.2015.08.004. URL: http://dx.doi.org/10.1016/j.compind.2015.08.004%20https://linkinghub.elsevier.com/retrieve/pii/S0166361515300348 (ver pp. 20, 21).

[40] F. Heylighen e C. Gershenson. "The meaning of Self-organization in Computing". Em: *IEEE Intelligent Systems* 18.4 (jul. de 2003), pp. 72–86 (ver p. 20).

[41] F. Heylighen. *The Science Of Self-Organization And The Science Of Self-organization And Adaptivity*. 2015. URL: https://hr-pioneers.com/2015/06/transformationale-fuehrung-agiles-unternehmen-kickinthepants/%20http://www.academia.edu/download/3243859/science_of_self_organization.pdf (acedido em 16/12/2020) (ver p. 21).

[42] G. DI MARZO SERUGENDO, M.-P. GLEIZES e A. KARAGEORGOS. "Self-organization in multi-agent systems". Em: *The Knowledge Engineering Review* 20.2 (jun. de 2005), pp. 165–189. ISSN: 0269-8889. DOI: 10.1017/S0269888905000494. URL: https://doi.org/10.1017/S0269888905000494%20https://www.cambridge.org/core/product/identifier/S0269888905000494/type/journal_article (ver p. 21).

[43] J. D. Ferreira. *Bio-Inspired Self-Organisation in Evolvable Production Systems*. Stockholm, 2013, p. 238. ISBN: 9789175018775 (ver pp. 21, 23, 25).

[44] A. Colorni, M. Dorigo e V. Maniezzo. *Distributed Optimization by Ant Colonies*. Rel. téc. (ver p. 23).

[45] M. Dorigo e T. Stützle. "Ant colony optimization: Overview and recent advances". Em: *International Series in Operations Research and Management Science*. Vol. 272. Springer New York LLC, 2019, pp. 311–351. DOI: 10.1007/978-3-319-91086-4_10. URL: https://link.springer.com/chapter/10.1007/978-3-319-91086-4_10 (ver p. 23).

[46] M. Dorigo, E. Bonabeau e G. Theraulaz. "Ant algorithms and stigmergy". Em: *Future Generation Computer Systems* 16.8 (jun. de 2000), pp. 851–871. ISSN: 0167739X. DOI: 10.1016/S0167-739X(00)00042-X (ver p. 23).

[47] S. Binitha e S. S. Sathya. "A Survey of Bio inspired Optimization Algorithms". Em: *International Journal of Soft Computing and Engineering (IJSCE)* 2.2 (2012), pp. 137–151 (ver pp. 23–25).

[48] M. Dorigo, V. Maniezzo e A. Colorni. "Ant system: Optimization by a colony of cooperating agents". Em: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 26.1 (1996), pp. 29–41. ISSN: 10834419. DOI: 10.1109/3477.484436 (ver p. 24).

[49] P. Lučić e D. Teodorović. "Computing with Bees: Attacking Complex Transportation Engineering Problems". Em: *International Journal on Artificial Intelligence Tools* 12.03 (set. de 2003), pp. 375–394. ISSN: 0218-2130. DOI: 10.1142/s0218213003001289 (ver p. 24).

[50] P. Lucic e D. T. .-. 1. I. I. C. 2. On. "Transportation modeling: an artificial life approach". Em: *ieeexplore.ieee.org* (). URL: https://ieeexplore.ieee.org/abstract/document/1180807/ (ver p. 24).

[51] T. Sharma, M. Pant e V. Singh. "Improved local search in artificial bee colony using golden section search". Em: *arXiv preprint arXiv:1210.6128* (2012), pp. 1–6. URL: http://arxiv.org/abs/1210.6128 (ver p. 25).

[52] X. S. Yang. "Firefly algorithms for multimodal optimization". Em: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5792 LNCS. Springer, Berlin, Heidelberg, out. de 2009, pp. 169–178. ISBN: 3642049435. DOI: 10.1007/978-3-642-04944-6_14. arXiv: 1003.1466. URL: https://link.springer.com/chapter/10.1007/978-3-642-04944-6_14 (ver p. 25).

[53] A. K. Kar. "Bio inspired computing – A review of algorithms and scope of applications". Em: *Expert Systems with Applications* 59 (out. de 2016), pp. 20–32. ISSN: 09574174. DOI: 10.1016/j.eswa.2016.04.018. URL: https://linkinghub.elsevier.com/retrieve/pii/S095741741630183X (ver p. 25).

[54] J. Dias-Ferreira et al. "BIOSOARM: a bio-inspired self-organising architecture for manufacturing cyber-physical shopfloors". Em: *Journal of Intelligent Manufacturing* 29.7 (out. de 2018), pp. 1659–1682. ISSN: 0956-5515. DOI: 10.1007/s10845-016-1258-2. URL: http://link.springer.com/10.1007/s10845-016-1258-2 (ver pp. 26, 27).

[55] J. Kennedy e R. Eberhart. *Particle Swarm Optimization*. URL: https://ieeexplore.ieee.org/abstract/document/488968/ (ver p. 26).

[56] X. S. Yang e M. Karamanoglu. "Swarm Intelligence and Bio-Inspired Computation: An Overview". Em: *Swarm Intelligence and Bio-Inspired Computation*. Elsevier Inc., jan. de 2013, pp. 3–23. ISBN: 9780124051638. DOI: 10.1016/B978-0-12-405163-8.00001-6 (ver p. 27).

[57] W. Xiang e H. P. Lee. "Ant colony intelligence in multi-agent dynamic manufacturing scheduling". Em: *Engineering Applications of Artificial Intelligence* 21.1 (fev. de 2008), pp. 73–85. ISSN: 0952-1976. DOI: 10.1016/J.ENGAPPAI.2007.03.008 (ver p. 27).

[58] C. S. Chong et al. "A bee colony optimization algorithm to job shop scheduling". Em: *Proceedings - Winter Simulation Conference*. IEEE, dez. de 2006, pp. 1954–1961. ISBN: 1424405017. DOI: 10.1109/WSC.2006.322980. URL: http://ieeexplore.ieee.org/document/4117838/ (ver p. 27).

[59] T. L. Lin et al. "An efficient job-shop scheduling algorithm based on particle swarm optimization". Em: *Expert Systems with Applications* 37.3 (mar. de 2010), pp. 2629–2636. ISSN: 09574174. DOI: 10.1016/j.eswa.2009.08.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417409007696 (ver p. 27).

[60] A. Rocha, L. Domingos e F. Ribeiro. "Increase the adoption of Agent-based Cyber-Physical Production Systems through the Design of Minimally Invasive Solutions". Em: (2018), p. 197. URL: http://hdl.handle.net/10362/58083 (ver p. 30).

[61] F. Bellifemine, G. Caire e D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley, 2007, pp. 1–286. ISBN: 9780470057476. DOI: 10.1002/9780470058411 (ver p. 41).

[62] *About the XAMPP project*. URL: https://www.apachefriends.org/about.html (acedido em 22/09/2021) (ver p. 43).

[63] "Apache Commons DBCP". Em: (). URL: https://commons.apache.org/proper/commons-dbcp/index.html%20http://jakarta.apache.org/commons/dbcp/ (ver p. 44).

[64] IEEE. *Welcome to the Foundation for Intelligent Physical Agents*. 2015. URL: http://www.fipa.org/ (acedido em 23/09/2021) (ver p. 46).

[65] F. for intelligent physical agents. *FIPA Request Interaction Protocol Specification*. 2013. URL: http://www.fipa.org/specs/fipa00026/SC00026H.html#_Toc26669020%20http://www.fipa.org/specs/fipa00026/SC00026H.pdf (acedido em 27/09/2021) (ver pp. 46, 47).

[66] P. E. Hart, N. J. Nilsson e B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". Em: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. ISSN: 21682887. DOI: 10.1109/TSSC.1968.300136 (ver pp. 55, 56).

JDBC Pool

Listagem I.1: Código de configuração JDBC pool

```
private static BasicDataSource datasource_state = new BasicDataSource();
    static{
        datasource_state.setUrl("jdbc:mysql://localhost:3306/state_tese?zeroDateTimeBehavior=conve
        datasource_state.setUsername("root");
        datasource_state.setPassword("");
        datasource_state.setMinIdle(5);
        datasource_state.setMaxIdle(10);
        datasource_state.setMaxOpenPreparedStatements(100);
    }

    private static BasicDataSource datasource_next_step = new BasicDataSource();
    static{
        datasource_next_step.setUrl("jdbc:mysql://localhost:3306/nextstep_tese?zeroDateTimeBehavio
        datasource_next_step.setUsername("root");
        datasource_next_step.setPassword("");
        datasource_next_step.setMinIdle(5);
        datasource_next_step.setMaxIdle(10);
        datasource_next_step.setMaxOpenPreparedStatements(100);
    }
```