Tiago Peralta Santos

Licenciado em Ciências da Engenharia Electrotécnica e de Computadores

Infraestrutura para gestão de sistemas de automação híbridos baseados em controladores lógicos programáveis e agentes

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

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

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

Júri:

Presidente: Prof. Doutor Tiago Oliveira Cardoso

Arguente: Prof. Doutor Paulo Jorge Pinto Leitão

Vogal: Prof. Doutor José António Barata de Oliveira

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2015

Infraestrutura para gestão de sistemas de automação híbridos baseados em controladores lógicos programáveis e agentes.

Para a minha ***Família***

Agradecimentos

A fim de expressar os meus agradecimentos àqueles que contribuíram para a realização deste trabalho, dedico a todos eles as seguintes palavras como forma de agradecimento por todo o apoio que manifestaram.

Em primeiro lugar, agradeço a toda a minha família, mais particularmente aos meus Pais e ao meu irmão, por todo o apoio e oportunidades que possibilitaram a realização do meu curso superior. Sem eles nunca teria conseguido concretizar este trabalho, o seu apoio nos momentos mais difíceis e o reconhecimento nos desafios superados deram-me a força necessária para realizar tudo aquilo que sempre desejei. Muito obrigado sou um privilegiado por vos ter sempre ao meu lado.

Ao meu orientador Professor Doutor José Barata, um especial obrigado, pelo voto de confiança e pela oportunidade única de participar no projeto PRIME. Num ano de trabalho, aprendi e conheci muitas pessoas o que me fez crescer pessoalmente e profissionalmente, ficarei para sempre grato por tudo o que me proporcionou. Muito obrigado não só pelo último ano, mas também por estes anos como meu professor.

Ao meu co-orientador Professor Doutor Luís Ribeiro, quero agradecer por tudo aquilo que me ensinou nos meus últimos anos de formação académica e todo o apoio expresso na realização deste trabalho. O seu contributo foi essencial no alcance dos meus objetivos. A sua dedicação e profissionalismo permitiram que superásse todos os desafios impostos, por isso, o meu sincero obrigado por todo o seu apoio incondicional.

Não podia deixar de agradecer ao André Rocha pelo apoio e companheirismo na realização deste projeto. A sua ajuda incondicional e todo o seu apoio foram muito importantes para mim, foi sem dúvida um privilégio trabalhar ao seu lado neste último ano, obrigado por tudo.

Não menos importantes na realização deste trabalho, gostaria de agradecer a todos aqueles com quem tive o prazer de trabalhar neste projeto. Ao Ricardo Peres agradeço o apoio e desejo muita sorte na continuação deste trabalho, ao Elkin Medina, ao Nikolas Antzoulatos, ao Daniele Scrimieri e ao Juergen Effenberger agradeço a forma espantosa como me receberam na Universidade de Nottingham, o esforço e a dedicação na validação deste trabalho, obrigado foi um prazer trabalhar ao vosso lado.

Depois da minha família, professores e colegas, gostaria de deixar um especial agradecimento à Flávia Moura e aos espantosos amigos que estiverem sempre ao meu lado, à Catarina Moura, à Ana Moura, ao Alexander Fernandes, ao Igor Fernandes, ao Hugo Pereira e ao António Gonçalves.

À Flávia Moura, por tudo o que significa para mim e por ter estado ao meu lado ao longo de todos estes anos, sem o seu apoio e compreensão tudo teria sido mais difícil.

À Catarina Moura e à Ana Moura um especial e carinhoso obrigado por todo o apoio incondicional manifestado e acima de tudo por toda a amizade.

Para os meus amigos Alexander Fernandes, Igor Fernandes, Hugo Pereira e António Gonçalves deixo aqui o meu profundo agradecimento, pelo o apoio e camaradagem, são sem dúvida, uns grandes amigos.

Termino, assim, com um profundo agradecimento a todos e com o sentimento de que sou um privilegiado por ter beneficiado do vosso apoio, muito obrigado.

Resumo

Atualmente a manufatura enfrenta diversos desafios devido à customização em massa. Esta customização deve-se à constante necessidade de satisfazer anseios específicos de clientes que exigem produtos diversificados, de baixo custo e prazo de entrega relativamente curto.

Para responder a estes desafios as empresas sentiram a necessidade de produzir muitos produtos diferentes no mesmo sistema. Uma solução seria a reformulação de linhas de produção, mas devido à conjuntura atual esta hipótese tornou-se impraticável para as empresas, sendo necessário arranjar soluções com base em sistemas já existentes.

Para isso têm surgido nos domínios da manufatura novos paradigmas, os Sistemas Flexiveis de Manufatura (FMS), os Sistemas Reconfiguráveis de Manufatura (RMS), os Sistemas Holónicos de Manufatura (HMS), os Sistemas Biónicos de Manufatura (BMS) ou os Sistemas Evolutivos de Produção (EPS), que visam solucionar muitos desses desafios. Através da sua implementação pretende-se obter sistemas mais robustos, dinâmicos, reconfiguráveis e com uma maior tolerância a falhas.

Este trabalho foi integrado no âmbito do projeto *Plug and Produce Intelligent Multi-Agent Environment based on Standard Technology* (PRIME), onde é proposta uma arquitetura auto-organizada capaz de reconfigurar sistemas de manufatura, sem comprometer o seu desempenho ou necessidade de reestruturação.

A presente dissertação contribui no projeto PRIME com a implementação da sua arquitetura com base em conceitos Sistemas Multiagente (MAS), que lhe confere um caráter distribuído de elevada versatilidade e interoperabilidade. Onde foram implementados os agentes, *Prime System Agent* (PSA). *Production Management Agent* (PMA) e *Skill Management Agent* (SMA), responsáveis pelos conceitos de auto-organização e *plug and produce*.

Estes agentes conferem à arquitetura proposta pelo PRIME, escalabilidade, adaptabilidade e reconfigurabilidade, pois a partir da implementação apresentada é possível dividir um sistema em vários subsistemas permitindo, assim, a sua gestão consoante os requisitos computacionais disponíveis e a agregação em diferentes níveis permitem aumentar a granularidade do sistema, facilitando a sua descrição.

A reconfiguração do sistema é feita com base na descrição do produto, a elevada granularidade apresentada com a implementação do trabalho proposto permite que a sua descrição seja mais simples, facilitando a introdução por parte do operador.

Uma vez que, o trabalho proposto consistiu na implementação da arquitetura proposta pelo PRIME, a sua validação esteve a par do segundo demonstrador do projeto. Neste demonstrador o trabalho desenvolvido foi submetido a um ambiente real que permite a montagem de uma dobradiça utilizada na cabine de um camião. Para verificar a *performance* de todo o sistema foi desenvolvido um caso de teste em ambiente virtual que permite obter métricas associadas a tempos que permitem analisar o sistema.

Palavras Chave: Sistemas Reconfiguráveis, Sistemas Multiagente, Tecnologia Padrão, *Plug and Produce*, Auto-Organização.

Abstract

Nowadays manufacturing faces diverse challenges due to mass customisation. This customisation is due to a constante necessity to satisfy specific client necessities, who demand diverse products at low cost and shorter delivery times.

To responde to these challenges companies felt the necessity to produce many different products on the same system. One solution would be to change production lines but due to the actual economic situation this scenario became impractical making it necessary to come up with other solutions using the already existing systems.

Therefore new paradigms have emerged in the manufacturing domain, such as, Flexible Manufacturing Systems (FMS), Reconfigurable Manufacturing Systems (RMS), Bionic Manufacturing Systems (BMS) and Evolvable Manufacturing Systems (EPS) to find solutions to these challenges. With the implementation of these paradigms systems will become more robust, dynamic, reconfigurable and with a higher faults tolerance.

This work was integrated in the Plug and Produce Intelligent Multi- Agent Environment based on Standard Technology (PRIME) project, where a self-organised architecture capable of reconfiguring manufacturing systems is proposed, without compromising its ability or restructuring necessity.

The present thesis contributed to the PRIME project with the implementation of its architecture based on Multiagent Systems (MAS) concepts, which gives it a distributed caracter of high interoperability. Prime System Agent (PSA), Production Management

Agent (PMA) and Skill Management Agent (SMA) were implemented responsible to self-organisation and plug-and-produce.

These agents confere the architecture proposed by the PRIME, scalability, adaptability and reconfigurability. It is possible to divide one system into various subsystems permitting its management according to available computing requirements and the aggregation in different levels permitting the increase of the granular system, facilitating its description.

With the product description the reconfiguration of the system is done, since the architecture shows a high granularity it permits the product description to be simpler making it easier for the operator.

Since this work was implemented in the architecture proposed by the PRIME, the second demonstrator of the project served as its validation, where a hinge of a truck cabine was assembled in a real environment. To verify the performance of the whole system a teste case was conducted in a virtual environment, with the objective to obtain the times which permit to analyse the system.

Keywords: Reconfigurable Systems, Multiagent Systems, Standard Technology, Plug and Produce, Self-Organisation .

Acrónimos

ADACOR *ADAptive holonic COntrol aRchitecture for distributed manufacturing systems*

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

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

CA *Component Agent*

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

CSK Habilidade Complexa,
do inglês *Complex Skill*

DA *Deployment Agent*

DMS Sistemas Dedicados de Manufatura,
do inglês *Dedicated Manufacturing Systems*

DPWS *Devices Profile for Web Services*

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

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

FIPA *Foundation for Intelligent Physical Agents*

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

FMS Sistemas Flexiveis de Manufatura ,
do inglês *Flexible Manufacturing Systems*

HMI *Human Machine Interface*

HMIA *Human Machine Interface Agent*

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

JADE *Java Agent Development Framework*

JINI *Java Intelligent Network Infrasctuture*

IMS *Intelligent Manufacturing Systems*

LMDAOA *Local Monitoring and Data Analyses Optimization Agent*

MAS Sistemas Multiagente,
do inglês *Multiagent System*

OWL *Web Ontology Language*

OPC UA *OPC Unified Architecture*

PA *Product Agent*

PMA *Production Management Agent*

PSA *Prime System Agent*

PRIME *Plug and Produce Intelligent MultiAgent Environment based on Standard Technology*

PROSA *Product-Resource-Order-Staff Architecture*

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

SMA *Skill Management Agent*

SMDAOA *System Monitoring and Data Analyses Optimization Agent*

SSK Habilidade Simples,
do inglês *Simple Skill*

SOA Arquitetura Orientada a Serviços,
do inglês *Service Oriented Architecture*

UPnP *Universal Plug and Play*

WS *Web Services*

WSDL *Web Services Definition Language*

XML *eXtensible Markup Language*

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Descrição do Problema

Devido a diversos aspetos sociais, económicos e tecnológicos muitas empresas de manufatura sentiram a necessidade de reformular conceitos como, por exemplo, a forma como os processos de fabricação são organizados. A agilidade passou a ser um requisito fundamental para empresas que pretendem atingir níveis de sustentabilidade a fim de enfrentarem novos desafios inerentes à globalização.

Assim, numa sociedade cada vez mais exigente e competitiva, em que produtos de alta qualidade, custos de produção reduzidos e ciclos de vida mais curtos começam a ser uma realidade, agilidade e flexibilidade são imprescindíveis ao *know-how* de empresas de sucesso e com capacidade de se adaptar às diferentes adversidades a que são sujeitas.

Ambientes de manufatura modernos são complexos, compostos por componentes de baixo acoplamento e heterogéneos que levaram ao surgimento de alguns conceitos que permitem melhorar estes sistemas a fim de aperfeiçoar a sua agilidade. Esta agilidade pode ser definida com a capacidade que os sistemas têm de sobreviver a um ambiente competitivo em mudança contínua e imprevisível. Impulsionados por produtos e serviços projetados para o cliente, os sistemas têm de reagir de forma rápida e eficaz à evolução dos mercados [RBO+15].

Através de um controlo autónomo e independente, pretende-se controlar cada um dos componentes presentes no *shop-floor*, permitindo reduzir os tempos de *ramp-up* e a interação com o operador na produção de diferentes tipos de produtos [AAM+00].

O conceito de *Plug and Produce* tornou-se um dos requisitos mais importantes dos sistemas de manufatura modernos. Este conceito permite a alteração de estruturas internas e a configuração de sistemas em resposta a uma mudança no ambiente de fabrico [RDOB+14].

Com o surgimento de um controlo autónomo de dispositivos, surgiram arquiteturas baseadas em conceitos de *self-organization* ([FDMSB08]) e *self-awarness* ([HWS11]). Estes conceitos referem-se a um conjunto de habilidades inerentes a um dispositivo capaz de torná-lo autonomamente adaptável a sistemas de manufatura robustos e totalmente distribuídos.

Todos este conceitos emergentes, apesar de serem muito promissores para a flexibilidade e agilidade de sistemas de manufatura, a sua implementação por vezes torna-se difícil [RDOB+14]. O acoplamento de componentes presentes no *shop-floor* pode ser uma tarefa muito complicada e isso deve-se à grande diversidade de tecnologias e arquiteturas pelas quais estes componentes se regem.

Em resposta a estes desafios é imperativo implementar novos sistemas de caráter distribuído e altamente flexíveis capazes de reagir de forma rápida a perturbações provocadas por falhas ou alterações na produção.

A presente dissertação propõe, apresenta e valida uma implementação baseada em MAS, de uma arquitetura que se rege por conceitos de *self-organization* e de *Plug and Produce*, com a expectativa de obter um sistema distribuído, escalável, de elevada granularidade, versatilidade e interoperabilidade capaz de se adaptar e reconfigurar de forma rápida diferentes sistemas de produção, independentemente da tecnologia utilizada e com

base na descrição do produto.

1.2 Aspetos de Investigação e Hipóteses

Tendo em conta os problemas apresentados anteriormente, é imperativo analisar as seguintes questões que se relacionam com o trabalho desenvolvido.

1. Será possível implementar uma arquitetura auto-organizada, distribuída e com elevada granularidade capaz de reconfigurar linhas de montagem independentemente da tecnologia utilizada?

2. Será uma implementação baseada em MAS, capaz de responder a todas as expetativas impostas, sem comprometer o desempenho ou o funcionamento de todo o sistema?

Neste contexto e de acordo com o enquadramento do projeto PRIME, que propõe uma arquitetura auto-organizada, capaz de se adaptar e reconfigurar de forma rápida diferentes sistemas de manufatura, a presente dissertação propõe a sua implementação com base em conceitos MAS. Através de uma pequena descrição do produto e devido à elevada granularidade imposta, é possível reconfigurar um sistema de produção sem comprometer o seu desempenho ou necessidade de reestruturação.

1.3 Visão Geral do Trabalho Desenvolvido

Com a finalidade de responder às questões apresentadas anteriormente é proposta a implementação baseada em conceitos MAS de uma arquitetura auto-organizada capaz de descrever e reconfigurar sistemas de manufatura, sem que para isso seja necessário alterações em estruturas já existentes.

Uma vez que a arquitetura foi proposta no âmbito do projeto PRIME, o trabalho

apresentado contribuiu para a sua implementação. No decorrer desta, foi proposta uma alteração que resultou no surgimento de uma nova entidade que lhe conferiu uma maior robustez e integridade, reduzindo a sua complexidade.

Neste trabalho foram implementadas três entidades distintas com responsabilidades singulares que através da interação entre si, formam um sistema reconfigurável capaz de se adaptar a diferentes sistemas de produção.

As responsabilidades associadas a cada uma das entidades permitem que estas tenham uma visão parcial do sistema e, através da cooperação entre si, representam-no como um todo.

É possível descrever diferentes tipos de sistemas de manufatura. As várias entidades representam diferentes níveis de agregação onde as de mais alto nível têm uma visão geral de todas a entidades associadas. Assim, é possível dividir um sistema em diversas áreas em que cada uma delas é representada por uma entidade. Para que seja possível fornecer ao operador toda a informação do sistema, com a finalidade de aumentar o seu desempenho, é possível extrair da entidade de mais alto nível toda essa informação.

Dependendo do tipo de sistema em causa, a informação a ser processada pode ser elevada. Para que o sistema não fique bloqueado e, desta forma, comprometa o seu desempenho, existem entidades responsáveis pelo seu processamento. Esta característica associada à capacidade de dividir o sistema em módulos desejáveis, confere-lhe uma elevada escalabilidade e granularidade.

A capacidade de agregação em diferentes níveis de informação carateriza a elevada granularidade do sistema, facilitando a descrição do produto. Assim, é possível com base numa descrição simples e em conceitos de auto-organização configurar linhas de montagem.

Por fim, existem entidades responsáveis por abstrair os componentes físicos do sis-

tema, que podem ser adicionados e removidos conforme seja necessário. É esta entidade que através da implementação de uma interface genérica, permite reconfigurar os componentes independentemente da tecnologias por eles utilizadas.

Para a implementação da arquitetura proposta foi utilizada a linguagem de programação Java e a plataforma JADE, permitindo uma implementação de caráter distribuído baseada em conceitos MAS.

Com o intuito de validar e analisar a *performance* da arquitetura implementada, foram considerados dois casos de teste, um real e outro virtual. No caso do teste real, a arquitetura foi submetida a um ambiente de produção real com a finalidade de configurar um sistema de manufatura para a montagem de uma dobradiça utilizada na cabine de um camião. No caso do teste virtual, foi analisada a *performance* de todo o sistema de modo a analisar métricas associadas a tempos relevantes.

1.4 Principais Contribuições

As principais contribuições do trabalho apresentado consistem na implementação de uma arquitetura distribuída capaz de proporcionar uma rápida reconfiguração de linhas de montagem, reduzindo os tempos de paragem inerentes à reconfiguração ou a alterações de produção, sem que seja comprometido o seu desempenho ou necessidade alterações estruturais.

A capacidade de adaptar a arquitetura a diferentes sistemas de manufatura tornou-se possível devido à possibilidade de adicionar e/ou remover componentes conforme necessário, conferindo-lhe uma elevada robustez, escalabilidade e capacidades de auto-organização/*plug and produce*.

De forma a reagir rapidamente às necessidades da demanda, a arquitetura permite a adaptação de forma rápida a diferentes ambientes de manufatura, permitindo a produção

de diferentes variantes de produtos, independentemente da sua natureza.

Uma vez que o sistema é reconfigurado com base na descrição do produto, a agregação em diferentes níveis permite que o sistema tenha uma elevada granularidade, facilitando a descrição do produto e de todo o sistema.

Capítulo 2

Estado da Arte

Numa sociedade em que as tendências de mercado estão em constante mudança, o termo produção em massa designado por produção em larga escala de produtos padronizados através de linhas de montagem, teve o seu grande impacto no início do século XX, protagonizado por Henry Ford, provando que a produção em linhas de montagem, com tarefas especificas e sistematizadas permite produzir quantidades significativas de produtos não-diversificados, baixando de forma significativa o seu preço [RB11].

Apesar da produção em massa ter revolucionado toda a indústria, os produtos não eram diversificados, prova disso é a celebre frase de Henry Ford *"have a car of any colour they wanted as long as it was black"*, que quer dizer que o cliente poderia ter um carro de qualquer cor desde que este fosse preto.

Assim, tempos de uniformidade e padronização, deram lugar a tempos de variedade e turbulência exigindo das empresas flexibilidade e agilidade para produzirem produtos personalizados e com ciclos de vida cada vez mais curtos, tornando-se sustentáveis e satisfazendo as necessidades de clientes cada vez mais exigentes.

A necessidade de responder às constantes mudanças do mercado sentida pelas empresas, levaram ao surgimento de novos conceitos para que uma produção diversificada de reduzidos volumes seja uma realidade. Promovendo a possibilidade de adicionar e remover

componentes rapidamente e, por conseguinte, uma rápida reconfiguração dos mesmos, de modo a adoptar o *shop-floor* a constantes mudanças na produção.

O conceito de reconfiguração tem sido bastante abordado tanto a nível académico como industrial, e isto deve-se muito à sua importância. Tal como referido em [ElM05], o custo associado à reconfiguração é uma vertente muito importante e a ter em conta, bem como o custo de *ramp-up* exigido cada vez que o sistema e os seus componentes são reconfigurados.

Com o evoluir das tecnologias de informação surgiu a Manufatura Integrada por Computador (CIM) e apesar de todos os seus méritos e valências a integração resultou em arquiteturas rígidas de controlo hierárquico, cuja complexidade cresce de forma exponencial com o evoluir do sistema [MVK06], dificultando a adaptação a novos ambientes de produção, o que faz com que uma rápida reconfiguração dos componentes se torne um grande desafio.

Tornou-se imperativo recorrer a arquiteturas de controlo distribuído de fácil reconfiguração, escaláveis e com uma maior tolerância a falhas, que ao contrário das arquiteturas mais convencionais não obrigue a grandes tempo de paragem de produção quer em caso de falha quer em caso de reconfiguração. No entanto, esta integração não é trivial pois pode acarretar um grande investimento por parte das empresas que devido à conjuntura atual é um aspecto a ter em conta. Para isso, muitos têm sido os trabalhos desenvolvidos a nível académico de modo a ir ao encontro das necessidades da indústria [MM05].

A fim de encontrar soluções para os problemas da manufatura foi surgindo na literatura alguns paradigmas, sendo que estes contemplam diferentes domínios, tais como paradigmas no domínio da produção ou engenharia de manufatura, são eles os FMS e os RMS, e no domínio da inteligência artificial e computacional os HMS, BMS, Sistemas Evolutivos de Manufatura (EAS) e EPS sendo que neste domínio os paradigmas são muitas vezes suportados por MAS.

2.1 Sistemas Flexíveis de Manufatura

O conceito de FMS foi introduzido em resposta à necessidade de customização em massa e de uma maior capacidade de resposta às alterações em produtos, tecnologia de produção e mercados. Estes aspetos têm sido considerados por muitos, uma característica de extrema importância na manufatura em busca de novas soluções na flexibilização de sistemas de produção.

Com o surgir deste paradigma foi possível no mesmo sistema, produzir diversos produtos com alteração de ordem ou de pequenos volume de produção. No entanto, geralmente este tipo de sistemas tem uma capacidade de produção menor do que a de linhas dedicadas e o seu custo inicial é mais elevado [KHJ+99].

Um FMS possibilita a fabricação rentável de diferentes tipos de produtos que podem ser trocados de tempos a tempos. Por exemplo, um sistema em que uma ou mais peças têm características geométricas, dimensões e tolerância semelhantes deve ser capaz de produzir os mesmos produtos. Isto permite a produção do mesmo produto em sistemas de produção semelhantes, reduzindo também o tempo de trocar associado a alterações na produção [KHJ+99].

Com o evoluir dos sistemas de manufatura, este paradigma desenvolvido na década de 1980 mostrou algumas fragilidades, pois não consegue lidar com a incerteza. Apesar de permitir a produção mais diversificada de produtos e acomodar mudanças na demanda, só é praticável se as variações forem previsíveis. E um dos grandes desafios dos Sistemas de Manufatura é a capacidade de fornecer produtos de alta qualidade instantaneamente em resposta à demanda [RB11].

2.2 Sistemas Reconfiguráveis de Manufatura

Com os ambientes de produção em constante mudança, onde alterações rápidas de tecnologias e/ou processos são aspetos muito importantes para o prolongamento da vida útil dos sistemas de produção, surgiu a necessidade de desenvolver novos conceitos que permitissem a reação dos sistemas a este tipo de alterações. Os RMS têm um papel preponderante, facilitando a integração de novas tecnologias ou de novas funções no sistema [ElM05].

O conceito de produção reconfigurável tem surgido, nos últimos anos, como uma tentativa de obter uma funcionalidade mutável e escalável onde, componentes, máquinas, células ou material de unidades de tratamento podem ser adicionados, removidos, modificados ou trocados conforme necessário. Isto permite aos sistemas de manufatura responder de forma rápida à constante mudança de requisitos na produção [ElM05].

A reconfiguração de *hardware* pode requerer grandes mudanças no software usado para controlar máquinas individuais ou até mesmo células ou sistemas por completo, para além de que é necessário planear e controlar os processos individuais de produção. Todos este aspetos têm um forte contributo para o aumento constante da complexidade de produtos, processos, sistemas e empresas [ElM05].

O conceito de RMS permite a um sistema uma rápida mudança na sua estrutura, bem como em componentes de *hardware* e *software*. Com o intuito de responder a mudanças bruscas de mercado ou exigências regulatórias este tipo de sistemas permitem o ajuste rápido às diferentes capacidades de produção e funcionalidades do sistema de manufatura, reduzindo assim o tempo de espera na reconfiguração/ajuste de novos sistemas ou de sistemas já existentes, facilitando a integração de novas tecnologias [KHJ+99].

Para que o sistema seja considerado um RMS deve, à partida, ser reconfigurável e criado por meio de módulos de *hardware* e *software* que podem ser integrados de forma rápida e confiável, caso contrário o processo de reconfiguração será demasiado demorado. Assim

para alcançar este objetivo este tipo de sistemas têm de ter modularidade, integrabilidade, customização, convertibilidade e diagnosticabilidade. A modularidade, a integrabilidade e a diagnosticabilidade reduzem o tempo e esforço de reconfiguração, a customização e convertibilidade reduz os custos [KHJ+99].

Um sistema de produção reconfigurável necessita de desenvolver módulos que podem ser rapidamente trocados entre diferentes sistemas de fabrico. Esta capacidade de troca pode ser realizada por igual, estruturas das máquinas e os sistemas de controle e padronização das interfaces que combinam os módulos, o que permite uma capacidade de adaptação a curto prazo às mudanças do mercado pela reconfiguração do sistema de manufatura. Para garantir a fácil reconfiguração deve ser actualizado tanto o sistema físico como o software de gerenciamento e o controlo de modo, a considerar as novas caraterísticas do sistema [KHJ+99].

De modo a responder às mudanças de demanda, sistemas de escala e mercados, este paradigma tem como objetivo, reduzir o tempo de *ramp-up*, que se tornou um grande desafio em sistemas de manufatura. O período de *ramp-up* é definido como o período entre o desenvolvimento e planeamento do produto até que são atingidos níveis de produção sustentáveis [KHJ+99].

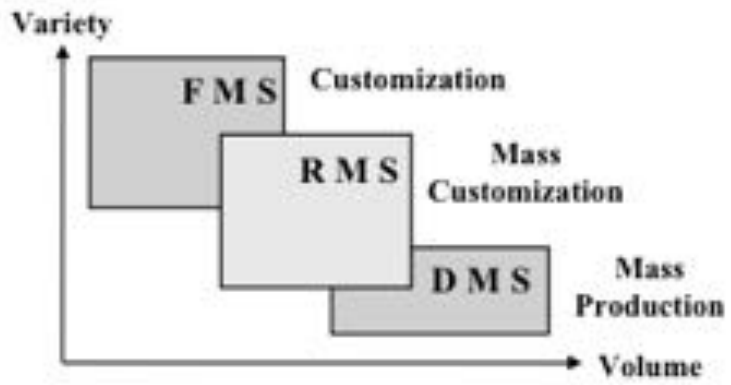

Figura 2.1: Comparação entre os paradigmas de manufatura, retirado de [ElM05].

Tal como se pode verificar na Figura 2.1, os RMSs surgiram da necessidade de posi-

cionar um novo paradigma entre os FMS e os Sistemas Dedicados de Manufatura (DMS), pois os RMSs permitem uma produção com flexibilidade personalizada [ElM05].

Em suma, um RMS deve ser capaz de lançar de forma rápida novos produtos, adaptar-se à capacidade do sistema de produção e às exigências do mercado. Neste tipo de sistemas deve também ser possível uma rápida integração de novas tecnologias, processo e uma fácil adaptação a quantidades variáveis de produtos para nichos de mercado, em sistemas já exigentes. [KHJ+99].

2.3 Sistemas Multi Agente

Segundo a literatura a tecnologia de agentes tem como base um controlo distribuído com um enorme potencial para resolver processos de decisão complexos e dinâmicos, permitindo que redes de entidades inteligentes possam interagir entre si de forma autónoma e racional [MM05].

A tecnologia de agentes surgiu paralelamente com o evoluir das tecnologias de informação e comunicação, no início dos anos 90. A partir desta data o conceito de comportamento racional em busca de um objetivo passou a ser uma realidade, pois a cooperação entre diferentes entidades e forma como se auto-organizam formando uma comunidade tornou-se muito vantajosa.

Apesar de existirem ínumeras definições, existe um consenso geral sobre as suas duas principais abordagens [MVK06]:

1. Um agente é um sistema computacional que está situado em um ambiente dinâmico dotado de um comportamento e inteligente;

2. Um agente pode estar num ambiente com outros agentes. Formando uma comunidade de agentes interagindo, como um todo, que dá origem a um MAS.

Segundo [MVK06], o conceito de agente deve ser associado a autonomia, inteligência,

sociabilidade e proatividade. Um agente é uma entidade autónoma, pois através de uma visão parcial do sistema deve atuar com a finalidade de controlar tanto o seu estado como o do meio em que está inserido. Um agente deve ser dotado de alguma inteligência, através da implementação de determinadas regras para raciocínio, planeamento e aprendizagem, a sociabilidade é garantida através da interação com o meio ambiente coabitando numa comunidade com outros agentes e tomando decisões com base nestas interações, por fim, a proatividade não menos importante que as anteriores permite que um agente seja capaz de adaptar o seu comportamento às mudanças do ambiente sem a intervenção de qualquer entidade externa.

Todas estas características apresentadas anteriormente, permitem que os agentes sejam entidades com bastante mobilidade e transparência, pois esta tecnologia permite que os agentes possam interagir entre si, sem comprometer o seu desempenho ou até mesmo o do sistema.

Um MAS é formado por uma rede de agentes computacionais que interagem entre si de modo a atingir um objetivo comum. Um agente só pode ter uma visão parcial do ambiente na qual é inserido e numa comunidade tem de coordenar as suas ações com base nas ações de outros agentes, antes de decidir o que fazer, assim o objetivo comum só pode ser alcançado se os agentes realizarem as ações acordadas entre eles (cooperação) [MVK06].

Tal como referido anteriormente os MAS têm tido um papel preponderante no desenvolvimento de novos paradigmas de manufatura, e isto deve-se ao facto de, os princípios de autonomia, proatividade e sociabilidade, resolverem muitos dos problemas identificados na indústria, nomeadamente, problemas associados a versatilidade e interoperabilidade.

Em [MM05] é mencionado como se pode adaptar a tecnologia Multiagente à indústria, e como, por vezes, se pode tornar numa tarefa bastante complexa, pois envolve a atribuição de tarefas de baixo-nível e decisões de execução envolvendo limitações de tempo consideráveis, sendo que, a motivação para a distribuição de soluções neste domínio passa

essencialmente por simplificar e melhorar a reconfiguração do ambientes de produção, a fim de melhorar a sua reação à rápida evolução da procura de produtos, recursos, falhas ou atualizações tecnológicas.

2.3.1 Sistemas Holónicos de Manufatura

O conceito de *holon* data de 1968, criado pelo filósofo Húngaro *Arthur Koestler*. *Holon* vem da combinação da palavra grega para o todo *"holos"* com o sufixo *"on"* que significa uma parte, assim para *Koestler* um *holon* é simultaneamente uma parte e um todo, descrevendo uma unidade básica de organização em sistemas biológicos e sociais [BC06].

Um *holon* é definido como um bloco autónomo e cooperativo de um sistema de produção para a transformação, transporte, armazenamento e validação de informações de objectos físicos, capaz de criar e controlar a execução dos seus planos e/ou estratégias onde um conjunto de entidades desenvolvem e executam planos mutuamente [BC06].

Um *holon* pode ser parte de outro *holon*, pois descreve a natureza híbrida de subconjuntos ou partes de um sistema [BC06].

A ideia de introduzir o conceito de *holon* em sistemas de manufatura só surgiu no início de 1990, para beneficiar das caraterísticas de estabilidade e adaptabilidade oferecidas por estes sistemas. Com o intuito de lidar com o aumento da customização de produtos que estava a afetar o setor da indústria transformadora, surgiu a um novo paradigma, os HMS [BC06].

Os HMS surgiram na comunidade de *Intelligent Manufacturing Systems* (IMS) sendo a primeira aplicação do conceito de *holon*, na manufatura e teve como base uma arquitetura também proposta pelo filósofo Húngaro, designada por holarquia. Uma holarquia rege-se pelo princípio de uma hierarquia *open-ended*, sem limitações ascendentes ou descendentes, que consiste num sistema de *holons* que cooperam para atingir uma meta ou determinado objectivo [BC06].

Existem diversos protótipos na literatura que se regem por este paradigma, sendo que na sua grande maioria são implementados sobre os princípios de MAS, pois tendo uma abordagem descentralizada é possível a distribuição da capacidade de processamento com diversas entidades autónomas e cooperativas, *ADAptive holonic COntrol aRchitecture for distributed manufacturing systems* (ADACOR) [BLAT14] e *Product-Resource-Order-Staff Architecture* (PROSA) [VBWV$^+$98] são duas arquiteturas baseadas em HMS.

A arquitetura ADACOR propõe a decomposição de funções de controlo na manufatura em quatro *holons* autónomos e cooperativos aproveitando a modularidade, a descentralização, a agilidade, a flexibilidade e a robustez características destas entidades [BLAT14].

Esta arquitetura propõe quatro tipos de *holons*, os *Product Holons*, os *Task Holons*, os *Operational Holons* e os *Supervisor Holon.* Os *Product Holons* representam os produtos a serem produzidos e contém a descrição necessária à sua produção, os *Task Holons* são responsáveis pela gestão da execução em tempo real, das ordens de produção, os *Operational Holons* representam os recursos do sistema, como por exemplo robôs, por fim, os *Supervisor Holon* responsáveis pela otimização do sistema.

A arquitetura PROSA é composta, essencialmente, por três tipos básicos de *holons*, os *Resource Holons*, os *Product Holons* e os *Order Holons*, no entanto, também propõe a utilização de *Staff Holons* que apesar de não serem entidades com presença obrigatória, podem desempenhar funções preponderantes através da cooperação com os *holons* básicos, a fim de os auxiliar no alcance dos seus objectivos [VBWV$^+$98].

Tal como referido anteriormente os *holons* que descrevem esta arquitetura cooperam entre si. Os *Product Holons* e os *Resource Holons* comunicam com a finalidade de adquiri conhecimento sobre as caraterísticas de determinado recurso. Os *Product Holons* e os *Order Holons* interagem para que se possa obter as informações e os métodos para produzir um determinado produto usando certos recursos. Por último, os *Order Holons* e os *Resource Holons* interagem permitindo a gestão de todo o processo, ou seja, possibilita

o iniciar, a interrupção e a monitorização de processos nos recursos [VBWV+98].

A arquitetura PROSA rege-se segundo dois conceitos que a caraterizam, são eles a agregação e a especialização [VBWV+98].

A agregação é a forma como se pode aglomerar os diferentes níveis de *holons*, ou seja, a forma com eles se organizam e formam um conjunto de *holons* relacionados, que são agrupados e formam, por sua vez, um *holon* maior com a sua própria identidade [VBWV+98].

A especialização garante a separação e a diferenciação de *holons* segundo as suas caraterísticas ou funcionalidades a desempenhar no sistema [VBWV+98].

Em suma, um HMS tem como finalidade preservar a estabilidade da hierarquia no caso de não existirem mudanças ou situações imprevistas, caso contrário, se existirem perturbações ou situações de falha, o dinamismo e a flexibilidade caraterísticos dos *holons* permite ao sistema adaptar-se de forma autónoma e cooperativa às alterações do meio onde se encontram.

2.3.2 Sistemas Biónicos de Manufatura

Os BMS são inspirados no funcionamento de órgãos naturais. Este paradigma baseia-se na hierarquia caraterística dos orgãos, ou seja, um organismo é constituído por diferentes órgãos que estão em constante troca de informações com outros orgãos, formando uma noção de hierarquia onde a informação é trocada entre diferentes camadas [Ued92].

Este paradigma visa lidar com as mudanças imprevisíveis, em ambientes de produção complexos com base em ideias de inspiração biológica, tais como auto-organização, evolução e adaptação [UHFV00].

Os organismos biológicos têm a capacidade de se adaptar às alterações ambientais e sobreviverem através de valências como o auto-reconhecimento, auto-crescimento, auto-recuperação e evolução, que caraterizados pelo *DNA-type* e pelo *BN-type* são dois tipos de informação biológica distintos [UHFV00].

À medida que os organismos vão evoluindo, com o passar do tempo surgem novas gerações, e de geração para geração a sua informação genética também vai evoluindo *(DNA-type)*, e isto deve-se ao facto dos organismos evoluírem de forma individual durante o seu ciclo de vida, adquirindo informação ao longo do tempo *(BN-type)*, sendo esta informação biológica a base para manter os sistemas vivos, autónomos e adaptáveis [UHFV00].

Tal como foi referido anteriormente este paradigma visa lidar com a alterações inesperadas, dotando-o de um grande potencial para solucionar os problemas com que a manufatura se tem deparado nos últimos anos.

Fazendo um paralelismo entre o paradigma e um sistema de manufatura, os diferentes componentes presentes no *shop-floor* podem ser considerados como organismos autónomos que operam sobre os produtos segundo uma informação *BN-type*, sendo desenvolvidos a partir de matérias-primas que expressam a sua própria informação, *DNA-type* [UHFV00].

Assim, é notório o enorme potencial dos sistemas bio-inspirados na resolução de problemas complexos de engenharia, e tal como os paradigmas apresentados nas seções anteriores quando aliados aos MAS, resultam em sistemas altamente úteis no domínio da manufatura, a fim de enfrentar os desafios atuais.

2.3.3 Sistemas Evolutivos de Produção

Atualmente, a manufatura enfrenta tempos de grandes desafios, e é imperativo encontrar formas de os solucionar, pois a sustentabilidade industrial exige níveis elevados de autonomia e de adaptabilidade dos sistemas, de modo a possibilitar uma resposta rápida à necessidades inerentes à customização de produtos. Com a finalidade de encontrar soluções tecnológicas e mecanismos de apoio tem surgido na literatura um novo paradigma denominado como EPS [OB09].

Numa primeira abordagem como EAS [OBF06], este paradigma permite responder a uma série de desafios técnicos e sócio-económicos, tais como a capacidade de evolução por parte dos sistemas de produção no processo de adaptação às mudanças nas condições de operação [RBP11].

O termo evolutivo implica a capacidade que o sistema tem de se ajustar de forma autónoma às exigências impostas. Com o objetivo de fornecer os dados necessários para a tomada de decisões estratégicas contínuas sobre como o sistema deve de ser gerido de forma holística [RBP11].

Um EPS foi desenvolvido a partir de uma perspetiva industrial de modo a responder às necessidades de um sistema quando existe uma mudança de produção, ou seja, quando existe uma mudança, por exemplo, ao nível do controle ou disposição física de produção. Estas mudanças são caraterísticas essencialmente quando existe uma troca de produto ou picos de demanda, tendo sempre como fator essencial o tempo de *ramp-up*. Sendo que, é nesta vertente que inside um grande contributo deste paradigma, visto que, paradigmas como HMS e RMS abrangem soluções de mais alto nível [OB09].

Numa fase inicial um EPS incide sobre as mudanças previstas e imprevistas que podem ocorrer dentro de uma gama de produtos limitada, e caso seja bem sucedido, pode gradualmente ser aplicado à família de produtos associados. Este tipo de sistemas não são considerados como uma solução genérica, pois incidem essencialmente no que acontece

quando existem mudanças do seu estado físico, de controlo ou de produção [OB09].

Assim, tal como referido em [OAB05] o conceito de EPS rege-se segundo dois princípios, um produto só é considerado inovador quando não existe restrições no seu processo de montagem e num sistema onde o dinamismo é imposto têm de ser necessariamente evolutivos, ou seja, é necessário terem a capacidade de evolução para serem tolerantes a mudanças de requisitos.

Apesar de ser um paradigma relativamente recente, têm surgido projetos Europeus onde o contributo de uma arquitetura baseada em EPS foi essencial para o seu sucesso, é o exemplo do EUPASS, do A3 e do IDEAS [OLBH12].

O IDEAS teve como objetivo principal implementar a tecnologia de agentes em controladores disponíveis no mercado. Este projeto foi resultado de vários desenvolvimentos feitos durante o projeto EUPASS [OLBH12].

Uma das principais inovações foi o facto de sistemas de produção altamente adaptáveis poderem ser criados com base em dispositivos de produção através da tecnologia de agentes. Realizar a agentificação das unidades do sistema afim de atribuir aos componentes físicos características até então particulares do conceito de MAS. No entanto, revelou-se uma das principais limitações deste tipo de sistemas pois a adoção de fabricação baseada em agentes ao nível dos dispositivos é difícil devido à inexistência de componentes de baixo custo capazes de suportar agentes [OLBH12].

Os EPS tiveram um grande contributo nestes projetos pois fornece uma solução com boa granularidade, ou seja, é possível suportar sistemas com diferentes níveis de granularidade. Esta granularidade está associada ao nível de complexidade de cada recurso num sistema de manufatura, ou seja, é possível modular um sistema independentemente do tipo de recursos por ele utilizados, quer sejam sensores, pinças, robôs ou até mesmo estações de trabalho que podem representar todo o sistema [OB09].

2.4 SOA e tecnologias emergentes na manufatura

Com o evoluir das tecnologias de informação e comunicação a abordagem de Arquitetura Orientada a Serviços (SOA) tem surgido como solução padronizada para dispositivos em rede e aplicações. Este tipo de arquiteturas permite disponibilizar funcionalidades de um serviço, definidas exclusivamente a partir de uma interface, sem que para isso tenha de ser apresentado qualquer tipo de detalhe de implementação [BBG06].

SOA estabelece um modelo de arquitetura que tem como objetivo aumentar a eficiência, agilidade e produtividade de uma empresa, fornecendo serviços como o principal meio através do qual uma solução lógica é apresentada. Assim, os sistemas baseados em SOA promovem aplicações em que cada função é implementada e exposta como um serviço possível de ser descoberto e utilizado por outros elementos presentes na rede [CJdOC10].

Este tipo de arquitetura deve ser baseado numa plataforma e linguagem de programação independente, a fim de ser amplamente aplicável. A grande maioria das SOAs são construídas com base em TCP/IP mas, cada vez mais, o *eXtensible Markup Language* (XML) tem sido usado como formato de informação padrão [BBG06].

Assim, com base neste tipo de arquiteturas têm surgido na literatura novos conceitos que visam solucionar problemas de interoperabilidade, tais como, o *Java Intelligent Network Infrasctuture* (JINI), o *Universal Plug and Play* (UPnP), os *Web Services* (WS), o *Devices Profile for Web Services* (DPWS) e o *OPC Unified Architecture* (OPC UA).

O JINI foi desenvolvido pela *Sun Microsystems* para fornecer uma rede de serviços e recursos baseados em tecnologia Java. Esta implementação permite que serviços/dispositivos sejam registados e mantidos num serviço para que possam ser chamados por outros serviços/dispositivos. O JINI é uma arquitetura que tem como principal desvantagem a sua dependência de Java e a necessidade de um registo centralizado [BBG06].

O UPnP é uma SOA simples, fácil de usar em pequenas redes. É uma arquitetura de

rede aberta que aproveita as características TCP/IP permitindo controlar a transferência de dados entre dispositivos ligados a uma rede. Tecnicamente, esta é uma arquitetura construída acima da camada do protocolo TCP/IP, através da combinação de vários protocolos padrão, por exemplo, DHCP, SSDP, SOAP, GENA, etc ... [MNTW01].

Os WS são uma arquitetura utilizada na integração de sistemas e na comunicação entre diferentes aplicações. Esta arquitetura promove um conjunto de especificações, como por exemplo, a descoberta de serviços, a descrição do serviço, entre outros, onde os serviços são descritos usando *Web Services Definition Language* (WSDL). WSDL é um formato XML para descrever serviços *Web*, especificando a localização do serviço e as operações ou métodos por ele expostos. Infelizmente, esta arquitetura não permite o suporte a dispositivos, de modo a solucionar este problema surgiu o DPWS [BBG06].

O DPWS surgiu pela primeira vez como uma proposta de utilização de protocolos baseados em *Web Services* para redes de dispositivos em Maio de 2004. Este tipo de arquitetura permite aos dispositivos executarem dois tipos de serviços: os serviços de hospedagem e os serviços de hospedados. Os serviços de hospedagem estão diretamente associados a um dispositivo e desempenham o processo importante na descoberta de outros dispositivos. Os serviços hospedados são na sua grande maioria dependentes do dispositivo de hospedagem para a sua descoberta [CJdOC10].

Existem diversas propostas que visam a integração de DPWS em dispositivos, todavia, é importante referir o contributo de projetos Europeus como o SIRENA [BBG06] e mais recentemente o SOCRADES [DSSG$^+$08], onde empresas como a *Schneider Electric* ou a *Siemens* implementaram e testaram dispositivos com a integração de DPWS no domínio da automação industrial.

O OPC UA é uma nova versão da arquitetura OPC concebida originalmente pela *OPC Foundation* para conectar dispositivos industriais e controlar/supervisionar aplicações. O objetivo desta arquitetura é conseguir o acesso a grandes quantidades de informação em

tempo real, sem perturbar o normal funcionamento de dispositivos. Esta arquitetura é baseada num protocolo cliente-servidor com recurso a tecnologias *web* (XML, WSDL, SOAP, etc ...) de modo a garantir a interoperabilidade entre dispositivos [LM06].

Na tabela 2.1 é possível verificar a comparação entre as caraterísticas das diferentes tecnologias apresentadas anteriormente.

Tabela 2.1: Comparação entre as diferentes tecnologias, modificado de [BBG06].

	JINI	UPnP	WS	DPWS	OPC UA
Suporta dispositivos	x	x	-	x	x
Programação independente	-	x	-	x	x
Elevada escalabilidade	x	-	x	x	x
Segurança	x	-	x	x	x
Elevada aceitação do mercado	-	x	x	x	-

De entre os aspectos referidos anteriormente destaca-se o facto de, os WS não suportarem dispositivos que levou ao surgimento do DPWS, a baixa segurança e escalabilidade inerente à arquitetura UPnP devido ao facto de, esta ser uma arquitetura de rede aberta desenvolvida para pequenas redes e a não aceitação do OPC UA no mercado que se deve ao facto de ser uma arquitetura relativamente recente, que ainda não foi devidamente implementada e testada em ambiente industrial.

2.5 Conclusões Gerais

Conforme foi possível verificar no presente capítulo, empresas de manufatura sentiram a necessidade de reformular conceitos para responder às mudanças na demanda. Estes conceitos têm como finalidade tornar os sistemas mais robustos, adaptáveis, flexíveis e reconfiguráveis.

Para isso têm surgido a nível académico diversos paradigmas como os FMS, os RMS, os HMS, os BMS e os EPS para solucionar muitas das exigências nos domínios da manufatura. Com estes paradigmas surgiram diversos trabalhos que se focam em sistemas reconfiguráveis que propõem arquiteturas específicas que são bastante promissoras, no en-

tanto, implicam um conjunto muito significativo de mudanças nas estruturas dos sistemas de manufatura, tornam-se inviáveis pois devido à conjuntura atual não se pode pedir às empresas uma grande reformulação das suas linhas de montagem, pois apesar de solucionar muitos dos seus problemas o retorno do investimento é tardio, tornando-se insustentável.

Paralelamente a estes paradigmas surgiram os MAS e os SOA que se têm desenvolvido nos últimos anos, no entanto, não têm um modelo mecatrónico, pois a sua orientação é baseada em conceitos de computacionais e não de *Shop floor*.

Um dos problemas encontrados na busca de uma solução para que as empresas possam responder de forma rápida às necessidades do mercado é a diversidade de tecnologias e metodologias por elas utilizadas. Para além da diversidade muitas dessas metodologias têm recursos computacionais reduzidos e tornaram-se obsoletas, o que dificulta a sua integração com os últimos trabalhos desenvolvidos.

Assim, é interessante e pertinente implementar uma arquitetura para a gestão de sistemas de automação híbridos baseados em controladores lógicos programáveis, que permita a reconfiguração de linhas de montagem independentemente da tecnologia utilizada, permitindo a sua integração em diferentes sistemas de manufatura.

Uma implementação baseada em conceitos MAS conferem ao sistema a versatilidade, a interoperabilidade e a escalabilidade necessária para que seja possível adaptar de forma rápida aos diferentes sistemas e às constantes mudanças exigidas a um sistema de produção moderno, assegurando um compromisso entre tecnologia e modelos de produção.

Capítulo 3

Arquitetura

Tal como referido no capitulo 2, têm surgido alguns trabalhos que visam tornar os sistemas de manufatura mais robustos e reconfiguráveis. No entanto, muitos destes trabalhos requerem uma reformulação dos sistemas já existentes e exigir isso às empresas pode tornar-se insustentável, assim é importante apresentar uma solução capaz de responder de forma rápida às necessidades da demanda, tornando os sistemas reconfiguráveis sem que para isso seja necessário reformular os sistemas já existentes.

A arquitetura proposta consiste num sistema altamente adaptável de modo a proporcionar a ambientes industriais um rápida reconfiguração de linhas de produção, independentemente do tipo de tecnologia utilizada. Desenvolvida segundo conceitos MAS, com a cooperação entre entidades genéricas e autónomas, é possível proporcionar um sistema distribuído com um elevada versatilidade e interoperabilidade.

3.1 Arquitetura do Sistema

A arquitetura em estudo foi desenvolvida no âmbito do projecto FP7 PRIME [pri15], e tem como objetivo principal, proporcionar de uma forma fácil a reconfiguração e adaptação de sistemas a alterações e ao monitoramento em ambiente de *shop-floor*.

A arquitetura apresentada é distribuída com elevada escalabilidade e granularidade constituída por diferentes entidades genéricas. Através da interação entre si, fornecem toda a informação do sistema, independentemente do tipo de tecnologia utilizada, ou seja, quer sejam compostas por tecnologias padrão ou dispositivos inteligentes.

A estrutura da arquitetura é composta por agentes genéricos em que cada um tem uma tarefa diferente, como representado na Figura 3.1.

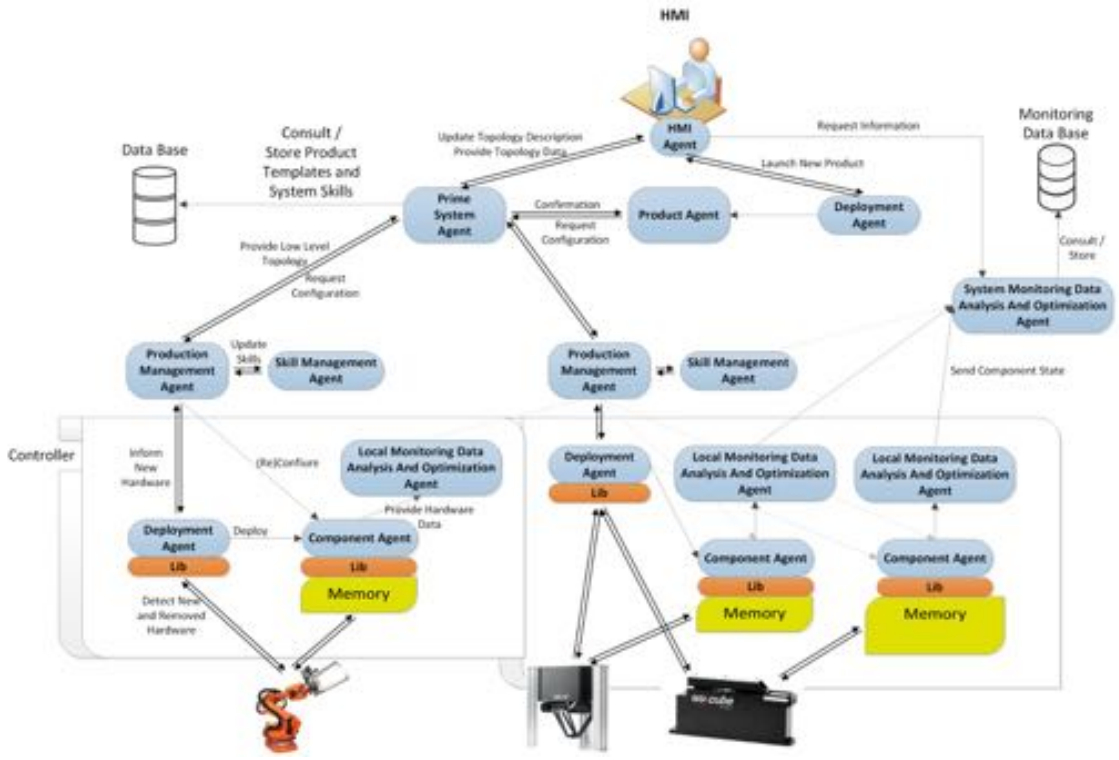

Figura 3.1: Arquitetura do PRIME.

Ao analisar a arquitetura apresentada, esta pode ser dividida em várias partes diferentes. Uma parte de mais alto nível, responsável pela interação com o utilizador, outra de mais baixo nível responsável por abstrair o *hardware* e fornecer às entidades de mais alto nível as suas habilidades e, por fim, um núcleo central responsável pela gestão das habilidades fornecidas e pela configuração de entidades de mais baixo nível. O trabalho realizado incide maioritariamente no núcleo central, na parte de gestão de habilidades e reconfiguração do hardware.

A arquitetura apresentada na Figura 3.1 permite ao utilizador saber através de uma ferramenta desenvolvida pela SIMPLAN [sim15] (*Human Machine Interface* (HMI)), qual o estado do sistema em tempo real, ou seja, qual o *hardware* que tem ao seu dispor e quais as habilidades disponibilizadas, quer pelas entidades de mais baixo nível, quer pela combinação desses mesmas habilidades.

Devido ao desenvolvimento segundo conceitos MAS, tais como versatilidade e interoperabilidade, com esta arquitetura é possível obter um sistema altamente escalável. Independentemente da organização do *shop-floor*, é possível dividir o sistema em vários subsistemas, permitindo assim fazer uma gestão, não só consoante o ambiente de produção sobre a qual se está a trabalhar, como também consoante os recursos computacionais disponíveis.

De seguida, será apresentada uma breve descrição de todos os agentes presentes na arquitetura, sendo que, depois serão apresentados aqueles sob os quais incidiu o trabalho desenvolvido.

***Human Machine Interface Agent* (HMIA)**

O HMIA funciona como uma ligação entre a arquitetura e o utilizador através da comunicação como um HMI. Assim, um HMI para obter informações relevantes como, por exemplo, os requisitos necessários para a execução de determinado produto e deve chamar serviços disponibilizados pelo HMIA, que inicia a comunicação entre

os agentes que fornecem esse tipo de informação.

***Product Agent* (PA)**

Quando o operador quer lançar uma nova variante de produto ou até um novo produto, o PA deve abstrair esse mesmo produto. Este agente é lançado com todas as informações que o descrevem, tais como o plano de processo e as habilidades necessárias para a sua execução. O PA desencadeia as rotinas de configuração, enviando uma lista de habilidades necessárias para a sua execução.

***Prime System Agent* (PSA)**

O PSA é a entidade de mais alto nível em todo o sistema. Este agente é único para cada instalação do PRIME, e é responsável por gerenciar os repositórios que detém toda a informação sistema, por exemplo, topologias e funcionalidades disponíveis.

***Production Management Agent* (PMA)**

O PMA é responsável por operações de *plug and produce* de um determinado subsistema, contendo toda a informação desse subsistema, tanto ao nível de topologia como de habilidades. Este agente funciona como uma árvore, a fim de construir diferentes camadas de agregação aumentando a modularidade e escalabilidade do sistema.

***Skill Management Agent* (SMA)**

Cada SMA trabalha em conjunto com um PMA específico. O SMA é responsável por receber a topologia do PMA e de acordo com as regras definidas, este agente deve verificar a possibilidade de criar habilidades de mais alto nível e caso seja possível informar o PMA de quais as habilidades de mais alto nível que este pode fornecer.

***Deployment Agent* (DA)**

O DA tem a responsabilidade de lançar os agentes da arquitetura PRIME, assim todas as máquinas (computadores e controladores), em que um ou mais agentes serão lançados, é obrigatório que o DA já esteja em execução em cada máquina.

***Component Agent* (*Component Agent* (CA))**

O CA é uma entidade de baixo nível e tem como funcionalidade abstrair cada equi-

pamento que constituem o sistema, contendo toda a informação sobre ele disponibilizada. Este agente tem a capacidade também de reconfigurar o recurso abstraído por ele.

***Local Monitoring and Data Analyses Optimization Agent* (LMDAOA)**

Da mesma forma que o PMA e o SMA, o LMDAOA trabalha a par com um CA. Neste caso específico, cada CA tem uma entidade associada deste tipo. Este agente é responsável por receber os dados relevantes extraídos a partir do CA e analisá-los a fim de definir a primeira camada de monitorização.

***System Monitoring and Data Analyses Optimization Agent* (SMDAOA)**

Este agente é responsável por recolher todos os dados relevantes do sistema, sendo estes dados fornecidos por cada LMDAOA. Com esses dados, ele irá calcular todas as informações e armazenar numa base de dados global com todos os dados históricos do sistema realizando, o monitoramento e análise. Este agente fornece os resultados para o HMIA quando solicitado.

3.2 Definição de habilidade

A definição de habilidade por vezes pode ser algo complexa, pois na grande maioria dos casos depende na natureza do componente, por exemplo, uma pinça oferece capacidades fundamentalmente diferentes de um robô [RBCO10].

Assim é importante existir um mínimo de informação para especificar uma habilidade, de modo a que esta seja especificada independentemente do recurso que a carateriza. Uma habilidade deve conter um Nome e um ID que deve ser único, uma breve descrição para facilitar a sua interpretação por parte do utilizador e uma lista de parâmetros para permitirem o seu controlo e execução. Este tipo de informação é fundamental na caraterização de componentes para assegurar a sua compatibilidade como um todo.

Um produto pode ser descrito por uma ou mais habilidades, sobre a forma de *skill*, sendo que esta *skill* durante o processo de execução podem ser essencialmente de dois tipos:

Habilidade Simples (SSK)

Uma SSK abstrai mecanismos genéricos fornecidos por um determinado modelo em que com o evoluir do sistema podem ou não coincidir com um processo.

Habilidade Complexa (CSK)

Sempre que uma SSK não é capaz de realizar um processo deve ser possível através de um conjunto de regras combinar habilidades disponibilizadas, criando uma nova habilidade sob a forma de CSK, a fim de possibilitar a realização do processo.

3.3 Descrição de Técnica

Tal como foi referido anteriormente o trabalho desenvolvido incide no desenvolvimento do núcleo central da arquitetura do PRIME, mais propriamente no desenvolvimento dos agentes, PSA, PMA e SMA. A presente seção visa não só apresentar o comportamento destas entidades bem como a forma como estes respondem às exigências das restantes entidades do sistema.

3.3.1 *Prime System Agent*

O PSA é a entidade de mais alto nível, numa fase inicial este agente não constava na arquitetura, tal como se pode verificar em [RDOB$^+$14], em que o papel do PSA era até então assumido por um PMA, designado como PMA (root). O PMA (root) na arquitetura apresentada em [RDOB$^+$14] é considerado o PMA de mais alto nível, ou seja ele é o ponto de entrada de todo o sistema e acima dele não existe nenhuma outra entidade.

Com o decorrer do trabalho tronou-se imperativo o desenvolvimento desta entidade, pois os níveis de complexidade do sistema estavam a aumentar de tal forma que as diferenças entre um PMA e um PMA (root) eram visíveis, como por exemplo o PMA (root) nunca podia ser associado a outra entidade pois é a de mais alto nível ao contrário dos outros PMAs presentes no sistema. Este facto exigia uma primeira análise em cada um

dos processos a implementar, de modo a verificar que tipo de entidade era, se um PMA ou um PMA (root). Com a crescente complexidade do sistema esta análise prévia do tipo de entidade consumia muitos recursos, que podem ser reduzidos, visto que, se pretende um sistema que funcione sob dispositivos industriais.

De seguida serão apresentados os principais comportamentos associados ao PSA.

Inicialização do PSA

Sendo esta entidade a de mais alto nível a sua topologia representa a topologia de todo o sistema, dos agentes desenvolvidos este é o primeiro a ser inicializado. Durante este processo de inicialização ocorrem alguns procedimentos essenciais ao bom funcionamento de todo o sistema.

Como se pode verificar na Figura 3.2, aquando da inicialização do PSA, este cria e guarda a sua topologia. Esta topologia contém toda a informação atual do sistema, tendo de ser atualizada sempre que existirem alterações.

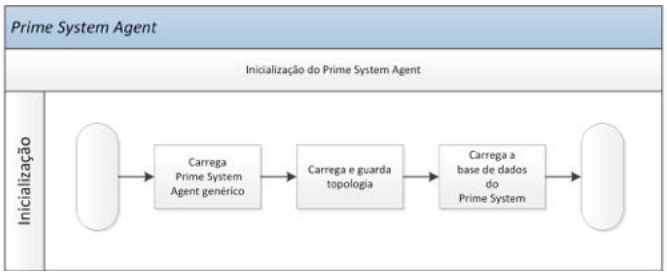

Figura 3.2: Inicialização do PSA.

Na Figura 3.1 estão representadas duas bases de dados, sendo que a *Configuration Data Base* é criada e manipulada pelo PSA. Esta base de dados é carregada para que possa ser manipulada futuramente.

Atualização do Sistema

Sempre que existe alguma alteração na topologia do sistema, subsistemas, componentes ou habilidades, o PSA tem obrigatoriamente de receber essa informação por parte dos PMAs de nível inferior, a fim de manter a integridade de toda a informação.

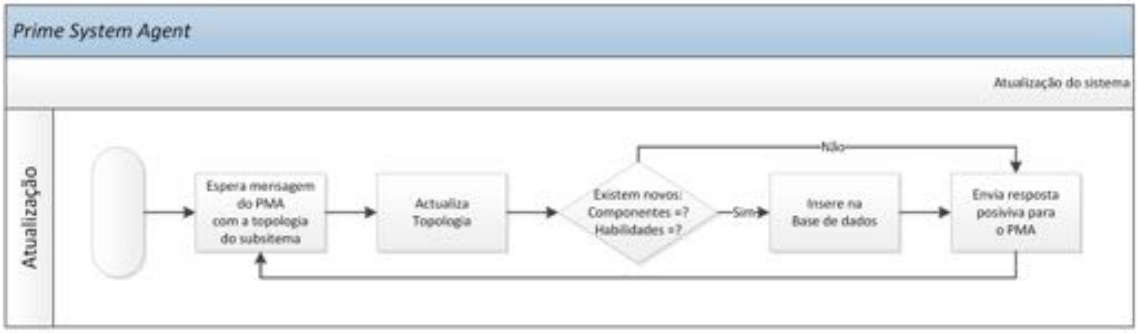

Figura 3.3: Atualização do sistema no PSA.

Como se pode verificar na Figura 3.3, este agente está sempre à espera de uma mensagem de outro agente de mais baixo nível (PMA). No conteúdo dessa mensagem vem a topologia do subsistema do agente que iniciou a mensagem. É com esta informação que o PSA vai actualizar a sua topologia.

A topologia do subsistema recebida contém toda a informação referente ao agente que a enviou, assim o PSA com base na informação vai verificar se existem novos componentes ou novas habilidades no sistema, resultantes da atualização. Caso exista um novo componente, uma nova SSK ou CSK que ainda não estejam guardados na bases de dados este guarda-os. Depois desta verificação feita, responde ao agente que enviou a mensagem a informar se o sistema foi atualizado com sucesso ou não.

Deteção de adição e remoção de componentes

Antes de serem adicionados ou removidos componentes ao sistema é verificada esta alteração. Esta comunicação é iniciada por um DA, e verificada ao longo de todo o sistema.

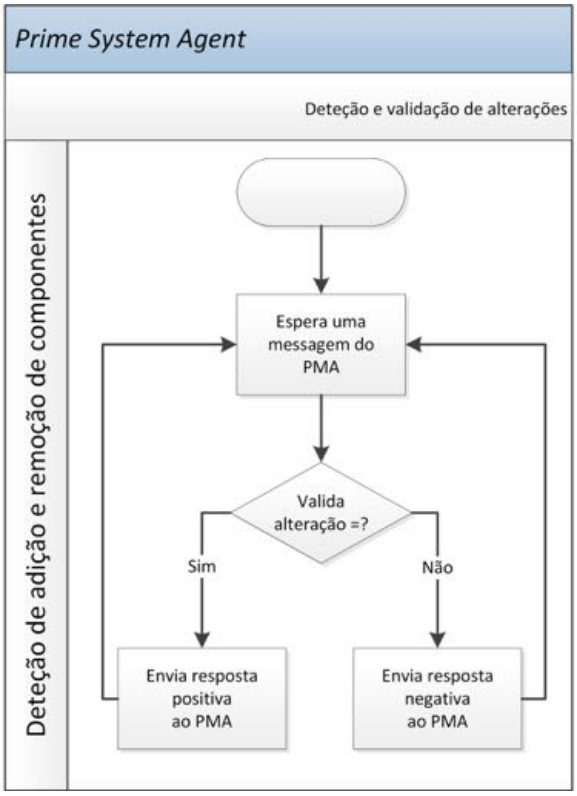

Figura 3.4: Deteção e remoção de componentes no PSA.

Na Figura 3.4, está esquematizado o comportamento do PSA quando recebe uma mensagem do PMA a notificar a deteção de adição ou remoção de um componente.

Ambos os casos, de adição ou remoção de componentes, ocorrem sempre que o DA deteta alterações ao nível dos componentes. Assim é enviada uma mensagem que percorrerá todo o sistema com o intuito de validar a adição ou remoção desse componente. Quando o PSA recebe uma mensagem do PMA valida a alteração e responde constante o resultado da validação.

Gestão de regras para gerar CSK

Tal como descrito anteriormente, os subsistema pode disponibilizar dois tipos de habilidades, no entanto, para gerar habilidades descritas como CSK é necessário existirem regras. É através destas regras que o sistema consegue gerar habilidades de mais alto nível, partindo de habilidades à *priori* disponíveis.

Uma regra para gerar CSK deve conter um nome e um ID que serão atribuídos à nova habilidade. Para disponibilizá-la é necessário que as habilidades do subsistema satisfaçam os parâmetros impostos pela regra.

Para isto, é necessário informar outros agentes de mais baixo nível, mais especificamente os SMAs. A forma como as CSK são geradas será explicado num subcapítulo 3.3.3.

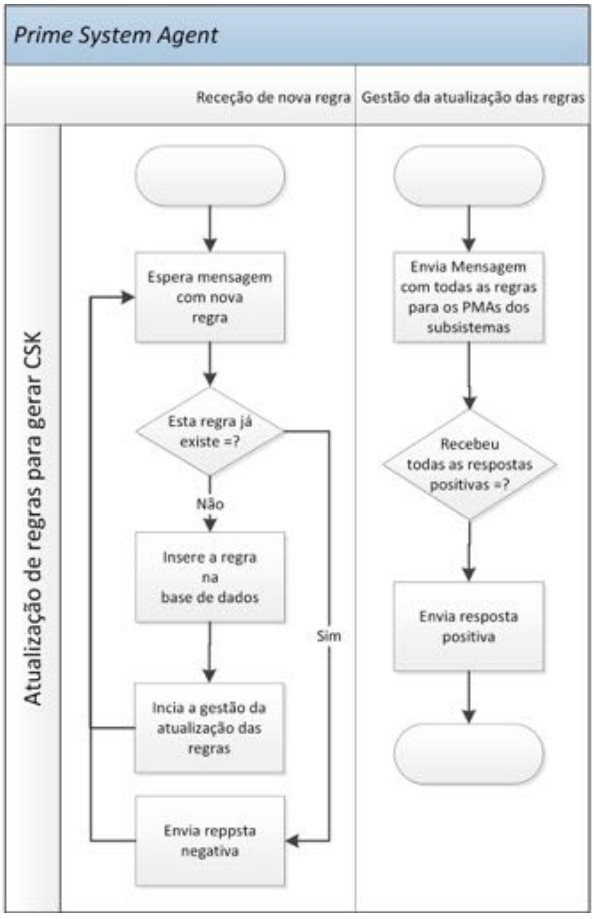

Figura 3.5: Gestão de regras no PSA.

O PSA aguarda que outro agente envie uma regra, quando a recebe verifica se já existe a fim de garantir a integridade da informação. Caso não exista este guarda-a na base de dados, e sendo esta uma nova regra os agentes de mais baixo nível não têm conhecimento da sua existência, então é enviada uma mensagem com todas as regras para os PMAs de nível inferior.

Quando o sistema é atualizado os agentes de mais baixo nível notificam o PSA que a atualização foi feita com sucesso. Garantido que o sistema foi atualizado com todas as regras.

Informação do sistema

Sendo o PSA a entidade que contém a informação de todo o sistema, é expetável que seja a entidade que responde a pedidos desta natureza.

Este tipo de informação disponibilizada pelo PSA é normalmente destinada ao utilizador. Sendo o HMIA responsável por disponibilizar esta informação é ele que desencadeia este tipo de comportamentos no PSA.

Para que o utilizador tenha acesso à informação como, por exemplo, quais os recursos disponíveis, o PSA recebe um pedido do HMIA. Como resposta a este pedido envia a sua topologia, onde está toda a informação do sistema.

Uma das informações cedidas é também os produtos que permitem a (re)configuração do sistema. Estes produtos encontram-se guardados na base de dados. Quando o HMIA pede todos os produtos ao PSA, este acede à base de dados e caso existam produtos são enviados para o HMIA, caso contrário é notificado que não existem produtos guardados.

Antes de um utilizador (re)configurar um sistema pode sentir a necessidade de saber se este disponibiliza todos os requisitos necessários à execução de determinado produto.

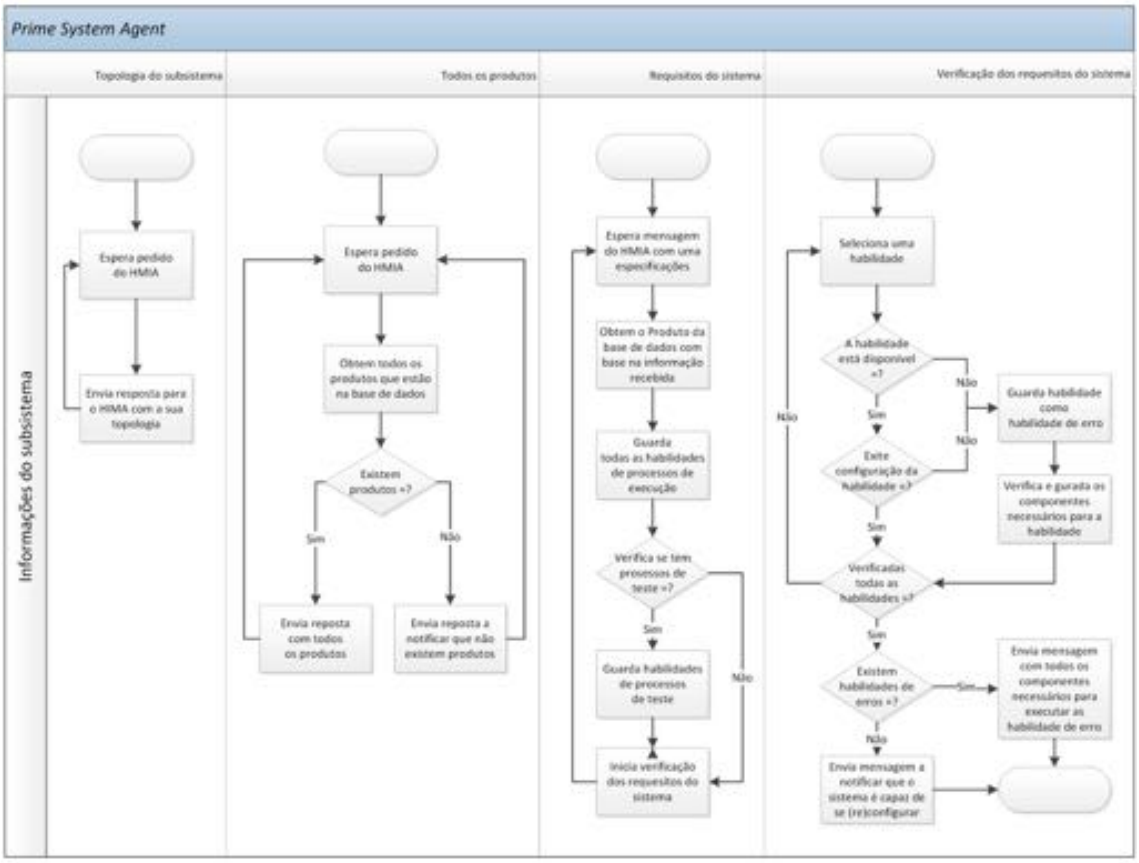

Figura 3.6: Informação do sistema no PSA.

Para isso o HMIA envia uma lista de especificações ao PSA, essa lista contém o *id* do produto, habilidades de teste desejadas e componentes desejados.

Assim quando o PSA recebe a lista vai à base de dados buscar o produto com base no *id* recebido e verifica as habilidades de teste. Por omissão, um produto tem associado a ele todas as habilidades de teste possíveis, bastando verificar dessas quais as desejadas.

Depois de obter o produto é verificado, se o sistema tem as habilidades disponíveis, configurações para cada uma das habilidades e por fim se os recursos presentes nas especificações recebidas permitem a execução de todas as habilidades.

Caso alguma das verificações falhe o HMIA é notificado de quais os componentes em falta para executar as habilidades exigidas pelo produto. Caso contrário, é notificado que

aquele produto pode ser produzido.

Configuração do sistema

Uma das grandes valências da arquitetura apresentada na Figura 3.1, é a capacidade que o sistema tem de configurar ou reconfigurar *hardware* de baixo nível independente da tecnologia utilizada.

É no PSA que se inicia este processo de (re)configuração, a Figura 4.21 descreve os principais comportamentos deste agente quando é exigida a (re)configuração do sistema.

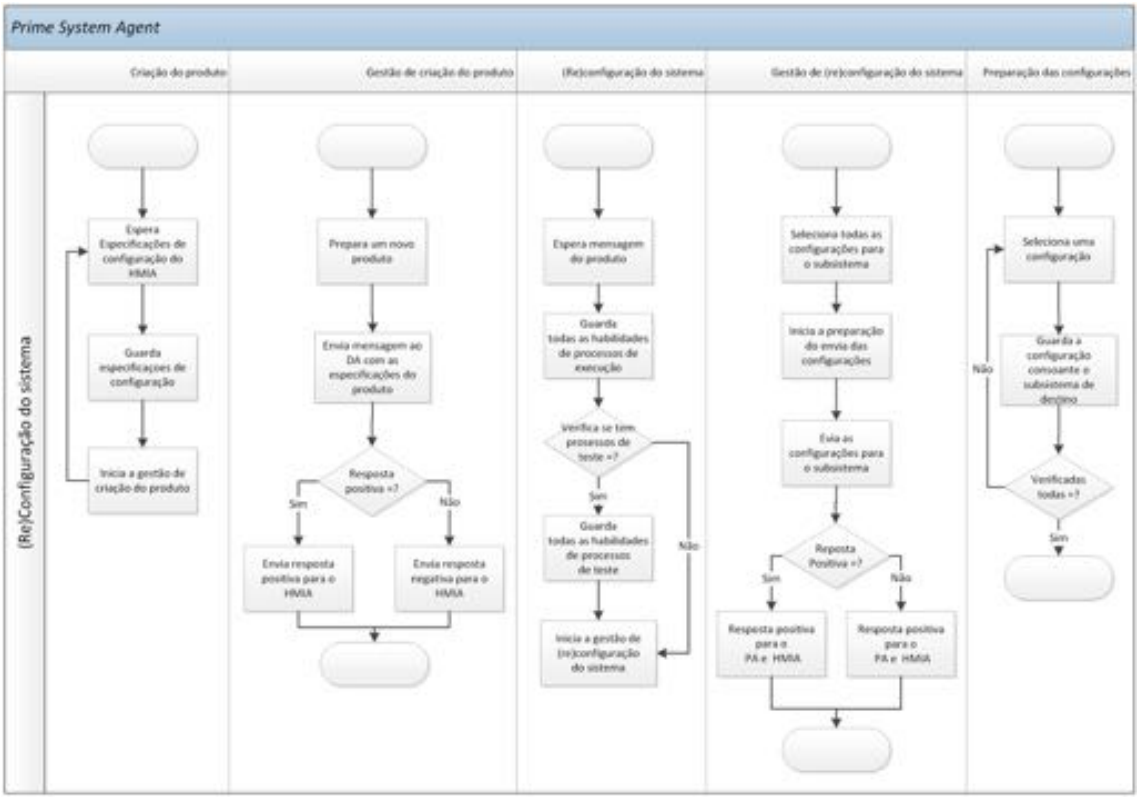

Figura 3.7: Configuração do sistema no PSA.

Antes de ser desencadeado o processo de (re)configuração do sistema, é necessário lançar o PA que abstrai o produto.

Quando o utilizador pretende (re)configurar o sistema para determinado produto, é necessário fornecer ao PSA algumas especificações. Estas especificações consistem no *id* do produto, quais as habilidades de teste desejadas e dos recursos disponíveis quais os desejados.

Ao receber do HMIA as especificações de (re)configuração, é obtido da base de dados o produto segundo o *id* recebido, e atribuídas as habilidades de teste desejadas. Depois de preparado o produto e toda a informação é enviada para o DA, para que este possa lançar o PA.

Uma vez lançado o PA, na Figura 4.21 pode-se verificar que o processo de (re)configuração do sistema é desencadeado com a receção de uma mensagem do PA.

A informação enviada pelo produto consiste numa descrição de uma ou mais habilidades. As habilidades podem ser de execução, ou de teste. As de execução são obrigatórias e as de teste facultativas, sendo o utilizador que decide quais pertence executar.

Assim quando este comportamento é desencadeado, o PSA guarda a(s) habilidade(s) responsáveis pela execução, se existirem habilidades de teste, guarda-as também. Depois deste processo dá-se início ao comportamento responsável pela gestão da (re)configuração, este processo consiste numa seleção e organização de configurações.

Para que o PSA possa iniciar o processo de (re)configuração do sistema, é necessário introduzir previamente as configurações na base de dados. Estas configurações têm de ter pelo menos uma habilidade associada, de modo a que este agente possa ter acesso a todas as configurações necessárias à execução da habilidade.

Assim o comportamento de gestão da configuração é iniciado com a validação das habilidades, ou seja, o PSA verifica se o sistema tem disponível todas as habilidades exigidas, caso não tenha guarda-as como habilidades indisponíveis.

No caso de o sistema ter disponíveis todas as habilidades exigidas, este vai buscar todas as configurações possíveis à base de dados, caso exista alguma habilidade que não tenha associada pelo menos uma configuração, esta é também guardada com uma habilidade indisponível.

Por fim, é verificado se com a lista de recursos recebida, é possível executar cada uma das habilidades. Sempre que uma habilidade não cumpre todos os requisitos é guardada com uma habilidade indisponível. Depois de analisadas todas as habilidades se existirem habilidades indisponíveis, são enviadas para o PA e para o HMIA, terminando assim o processo de reconfiguração.

Pode ocorrer o caso em que uma habilidade tem mais do que uma configuração disponível. Assim, é necessário associar às configurações um parâmetro que lhe confere uma prioridade. Esta prioridade é associada à configuração com base em preferências, por exemplo, se o utilizador preferir (re)configurar o robô 1 em vez do robô 2 a configuração associada ao robô 1 tem prioridade superior quando comparada com a do robô 2, caso exista mais do que uma configuração para um habilidade é escolhida a que tem maior prioridade, em caso de prioridades iguais é escolhida uma, pois significa que não existe preferência por qualquer uma delas.

Selecionadas as melhores configurações, dá-se início ao comportamento responsável pelo envio das configurações para os agentes de mais baixo nível.

Ao PSA pode ser associado mais do que um PMA. As configurações podem ser para diferentes subsistemas dependendo do subsistema que disponibiliza a habilidade. Então antes de se dar início ao processo de comunicação, as configurações são agrupadas consoante o subsistema à qual se destinam, reduzindo assim o número de mensagens, visto que, um subsistema pode receber mais do que uma configuração e também o facto de não serem enviadas configurações para subsistemas não desejados.

Já agrupadas, as configurações são enviadas para os PMAs correspondentes. Depois dos componentes terem sido reconfigurados o PSA recebe uma notificação com o resultado das (re)configuração, por sua vez, informa o PA e o HMIA desse resultado.

3.3.2 *Production Management Agent*

Um PMA é uma entidade responsável por abstrair um subsistema. Um subsistema compreende um conjunto de recursos que podem colaborar uns com os outros, por exemplo, uma ferramenta a ser utilizada por um robô está no mesmo subsistema, isto é, os CA que abstraem estes componentes estão ligados ao mesmo PMA.

Um subsistema também pode ser parte de um subsistema de nível superior, ou seja, um PMA pode ter outros PMAs no seu subsistema conferido assim a capacidade de estas entidades se organizarem em forma de árvore.

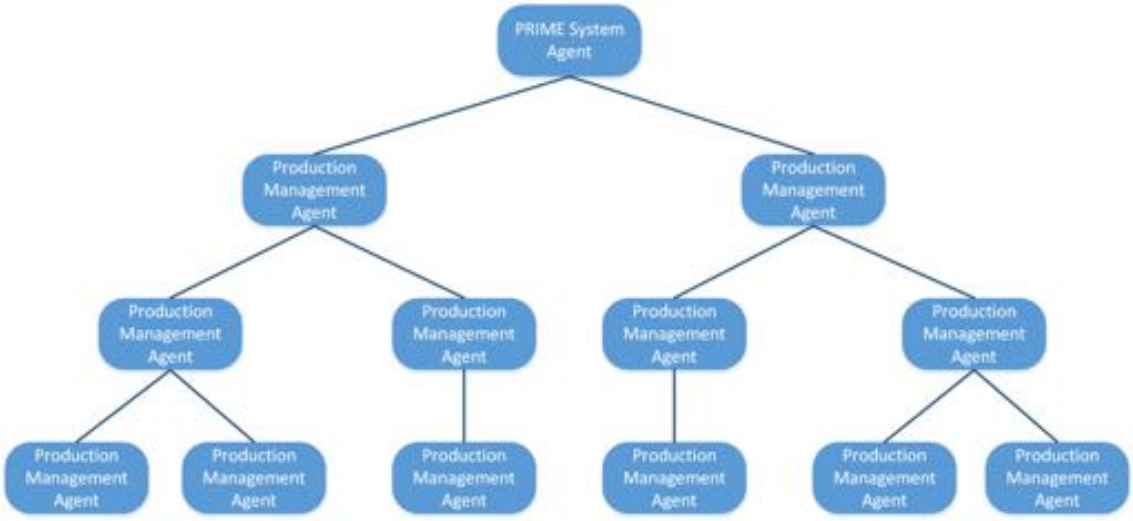

Figura 3.8: Árvore de PMAs.

A Figura 3.8 é um exemplo da forma como os PMAs se podem organizar em forma de árvore. Como se trata de entidades genéricas e com um elevado dinamismo é possível ao utilizador modular o sistema consoante as necessidades do *shop-floor*.

Os níveis mais baixos de PMAs contêm no seu subsistema os CAs que disponibilizam diferentes habilidades (SSK), é com base nestas habilidades que os vários níveis de subsistemas disponibilizam as suas competências sob a forma de habilidades de mais alto nível (CSK). Estas competências disponíveis podem ser agregadas, conferindo ao sistema maior granularidade, facilitando a descrição do produto bem como a do sistema como um todo.

Cada PMA tem associado um SMA, esta entidade é responsável por gerir as competências de cada subsistema, informando o PMA de quais as habilidades disponíveis no seu subsistema consoante a topologia que este disponibiliza. Este comportamento será abordado no subcapítulo 3.3.3, onde será apresentada de forma detalhada os comportamentos caraterísticos do SMA.

Inicialização do PMA

À semelhança do que acontece no PSA , quando o PMA é inicializado, este cria e guarda a topologia da parte do sistema a ele associada, sempre que existirem alterações no seu subsistema ou subsistemas associadas, a sua topologia é atualizada.

Tal como referido anteriormente, associado a um PMA existe sempre um SMA, assim sempre que um PMA é inicializado um SMA. Permitindo que o PMA tenha conhecimento do SMA responsável pela gestão das habilidades a serem disponibilizadas pelo subsistema que representa.

Como um PMA nunca é a entidade de mais alto nível, pois num nível superior pode ter outro PMA ou até mesmo o PSA, este tem obrigatoriamente de enviar a sua topologia ao agente de nível superior, tanto na sua inicialização como em situações de alterações de topologia. Uma vez feita esta comunicação entre o PMA e o agente de mais alto nível, é a este agente que o PMA enviará a sua topologia sempre que esta sofra alterações.

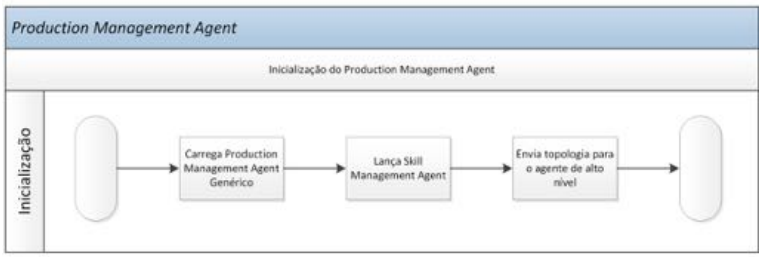

Figura 3.9: Inicialização do PMA.

Atualização do Subsistema

Existem várias causas que podem levar ao desencadear deste comportamento, tais como, a adição ou remoção de componentes ou alterações nas habilidades disponibilizadas.

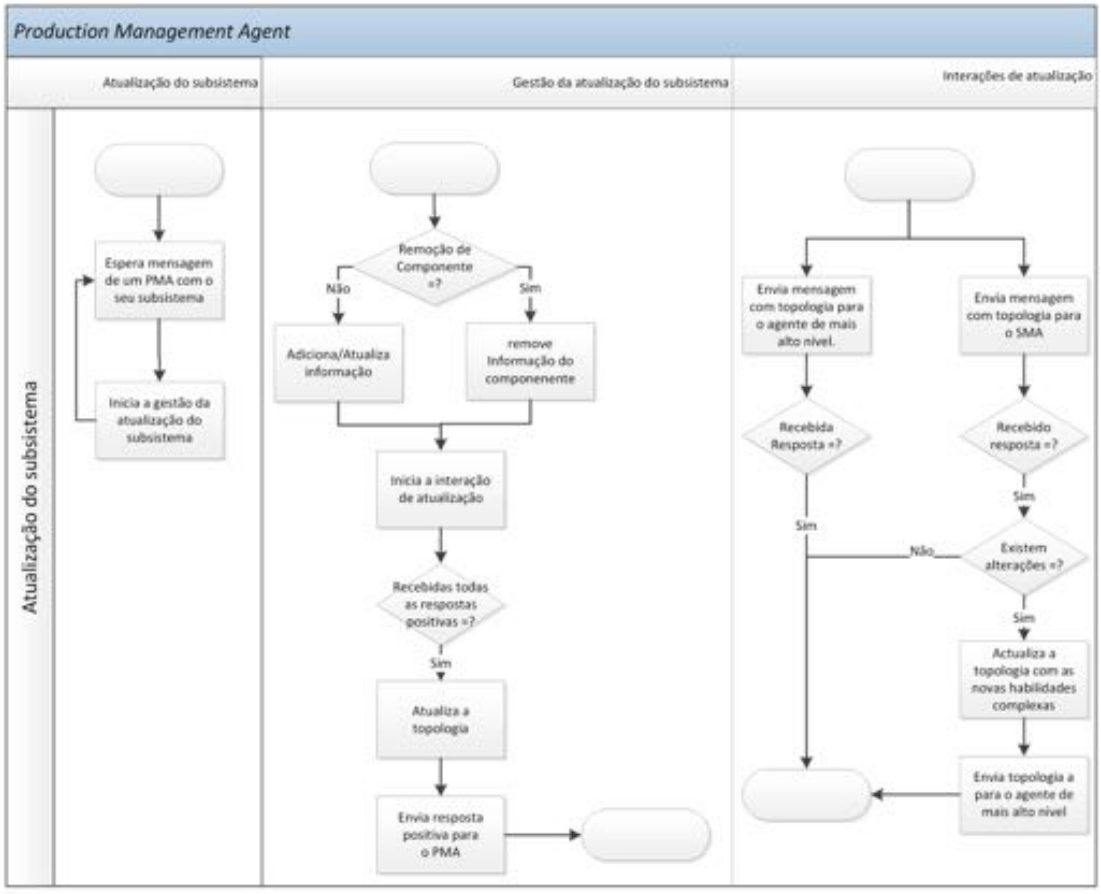

Figura 3.10: Atualização do subsistema no PMA.

Então sempre que existe uma alteração na topologia dos subsistemas pertencentes ao subsistema do PMA este recebe a topologia do subsistema inferior e atualiza a sua topolo-

gia com esta informação. Para garantir a integridade de todo o sistema esta nova topologia é provisória pois é necessário primeiro informar agentes de mais alto nível desta alteração. Só depois de garantir que todos os agentes de níveis superiores foram atualizados é que esta topologia é considerada válida.

Assim é desencadeado o comportamento de interações de atualização que consiste em enviar paralelamente para o agente de nível superior e para o SMA a nova topologia. O facto desta interação ser em paralelo garante também a integridade do sistema pois, independentemente do tempo que o SMA demora a processar toda a informação outras entidades de mais alto nível são notificadas das alterações.

Depois de recebida a notificação do agente de mais alto nível de que toda parte superior do sistema correspondente está atualizada, a topologia até então provisória é dada com definitiva e é notificado o agente que desencadeou este processo de que a atualização ocorreu com sucesso.

Como paralelamente foi desencadeado um comportamento no SMA, o resultado desse comportamento pode ser abordado de duas formas diferentes. No caso de o SMA verificar que com a nova topologia do subsistema existem alterações nas habilidades que disponibiliza, o PMA é notificado dessas alterações, desencadeando todo o processo de atualização da parte superior do sistema ao qual pertence. Este processo repete-se até que o SMA notifique o PMA que não existem alterações na topologia.

Assim é garantida a integridade do sistema, pois sempre que existem alterações num determinado nível da árvore, todos os níveis superiores são notificados, garantindo que o PSA tem a informação atual e real o sistema.

Deteção e adição/remoção de CAs num subsistema

Tal como descrito anteriormente os CAs têm como finalidade abstrair a parte física do sistema, e assim fornecer às entidades de nível superior SSK. Um CA pode ser associado ou removido de qualquer subsistema, no entanto deve de ser estrategicamente colocado de modo a garantir uma maior rentabilidade do sistema.

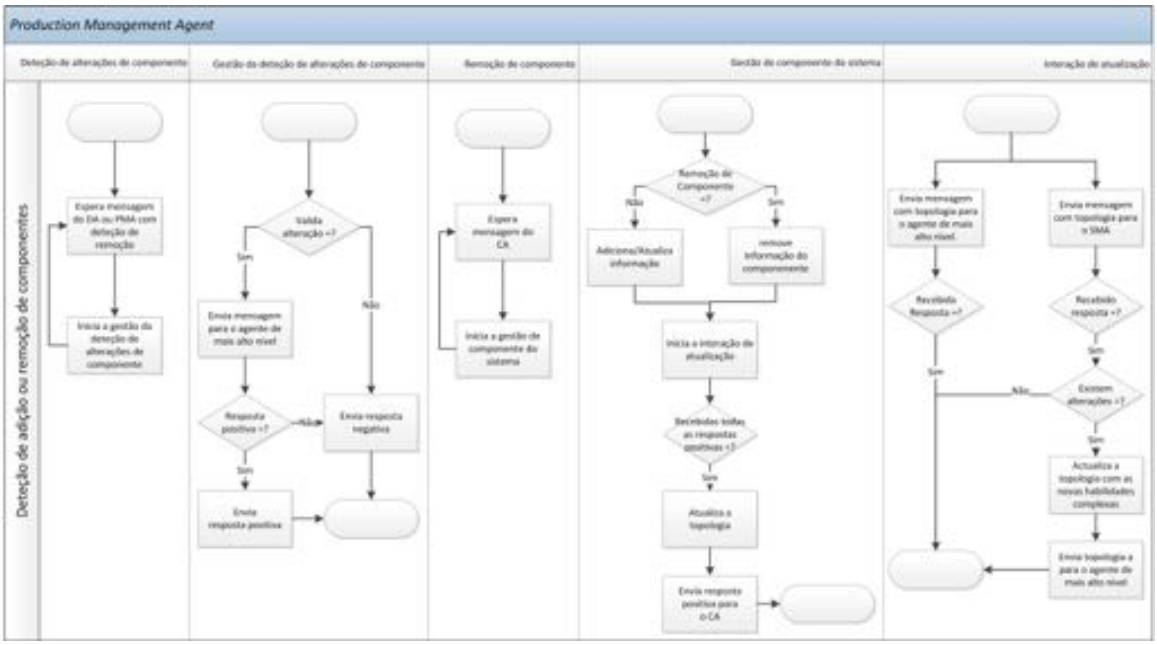

Figura 3.11: Deteção e adição/remoção de CA's no PMA.

Antes de ser associado/removido um CA a um subsistema é necessário validar esta alteração do sistema. Sendo o DA a entidade responsável por lançar agentes no sistema, antes de lançar um CA este desencadeia um comportamento no PMA que valida a alteração provocada no sistema. O PMA informa os agentes de nível superior a fim de validarem também essa alteração, sendo este processo idêntico quando a alteração é referente à remoção de um CA.

Assim quando o PMA é notificado pelo agente de nível superior que a alteração é válida, este notifica o DA para lançar o CA ou para este dar ordem de remoção ao CA. É expectável que em ambos os casos exista uma comunicação entre o PMA e o CA, no caso

de se tratar de uma adição o CA informa o subsistema das suas capacidades, no caso de remoção o CA informa o subsistema da sua remoção. De seguida é desencadeado no PMA o mesmo comportamento descrito em 3.3.2.

No caso do processo não ser validado, ou pelo PMA ou por outro agente de nível superior, o DA é informado e o CA não é lançado no sistema ou não é removido, consoante o processo em causa.

Atualização das regras para gerar CSK

Para que seja possível um subsistema disponibilizar habilidades de mais alto nível (CSK), é necessário informar todos os SMA quais as regras para gerar essas habilidades.

Ao contrário dos comportamentos descritos no subcapítulo 3.3.2, em que a atualização é feita da parte de mais baixo nível para a parte de alto nível, a atualização de regras para gerar CSK é o inverso, iniciando-se no PSA, tal como descrito em 3.3.1, percorrendo toda a árvore até ao último nível de subsistemas.

Como se pode verificar no primeiro comportamento descrito na Figura 3.12, o PMA aguarda que seja envia a lista de regras. Quando este o recebe é necessário informar todos os subsistemas a ele associados e o seu SMA, sendo que este processo também é feito em paralelo tal como os anteriores.

Assim o PMA verifica na sua topologia se tem subsistema(s) a ele associado(s), caso tenha envia para o(s) agente(s) que o(s) representa(m), as regras recebidas. Paralelamente, envia a lista de regras para o SMA, caso este verifique que com estas regras as habilidades disponibilizadas pelo subsistema não sofrem alterações, informa o PMA que não existem alterações, caso contrário envia para o PMA as habilidades a disponibilizar. Posto isto desencadeia-se no PMA o comportamento associado à atualização, descrito no subcapítulo 3.3.2.

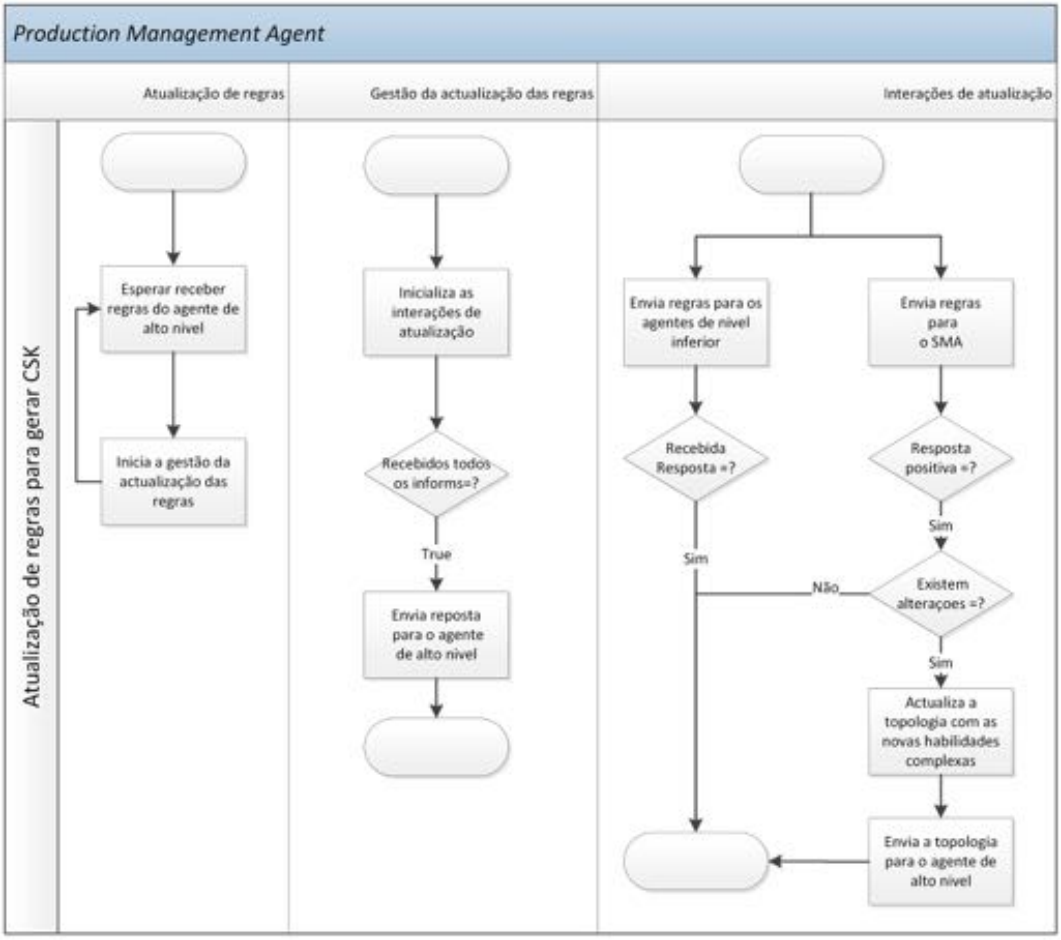

Figura 3.12: Atualização das regras para gerar CSK no PMA.

Gestão de (re)configurações no subsistema

Tal como descrito em 3.3.1 é no PSA que se dá início à (re)configuração do sistema, sendo este agente que desencadeia este comportamento nos PMAs de nível inferior.

Assim, quando um PMA recebe uma configuração é garantido no PSA que esta configuração é para o subsistema que este representa ou para um subsistema associado a ele.

Ao receber a configuração o PMA verifica se a configuração recebida é para uma habilidade por ele disponibilizada, caso não seja, verifica qual é o caminho para o subsistema que disponibiliza essa habilidade enviando a configuração. Um PMA pode receber uma ou mais configurações para diferentes habilidades e isso pode implicar para mais do que

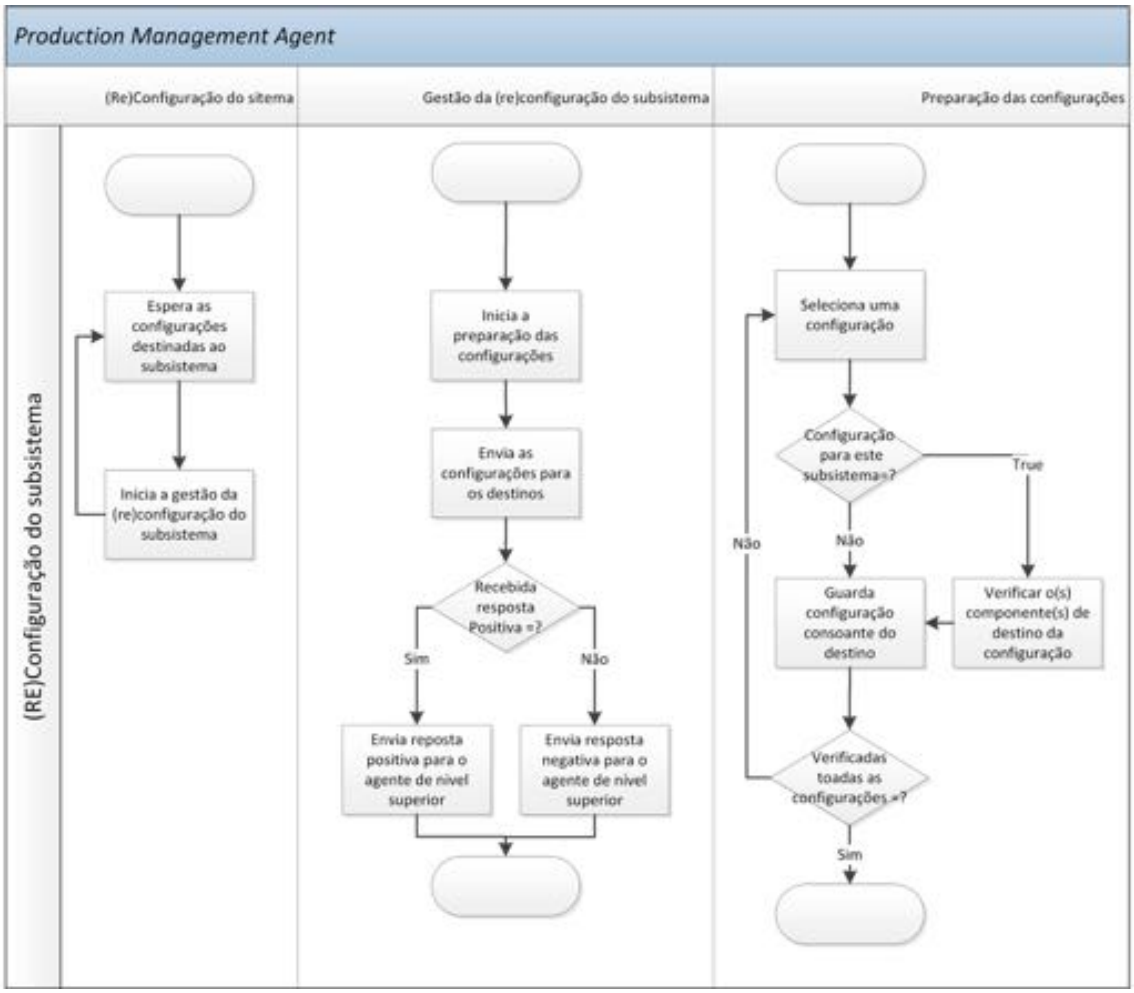

Figura 3.13: Gestão de (re)configurações no PMA.

um subsistema, assim neste caso o PMA agrupa as configurações consoante o próximo subsistema de destino e envia-as.

Para o caso da configuração recebida ser para uma habilidade por ele disponibilizada, o PMA envia uma notificação com a configuração para os CAs responsáveis pela execução de habilidades. Quando os CAs (re)configurarem as entidades físicas que abstraem com essa informação, notificam o PMA com o resultado dessa (re)configuração.

Para que seja possível saber se o sistema foi bem (re)configurado, a resposta é propagada por todo o sistema até ao PSA.

3.3.3 *Skill Management Agent*

Cada PMA tem associado um SMA. Este agente é responsável pela gestão de habilidades do subsistema ao qual está associado, ou seja, com base em habilidades disponibilizadas pelo subsistema e regras pré-definidas, esta entidade tem a capacidade de gerar CSK e informar o PMA a ele associado para que as possa disponibilizar.

Inicialização do SMA

Como este agente é lançado em simultâneo com o PMA à qual está associado, este tem conhecimento da topologia do subsistema e guarda-a.

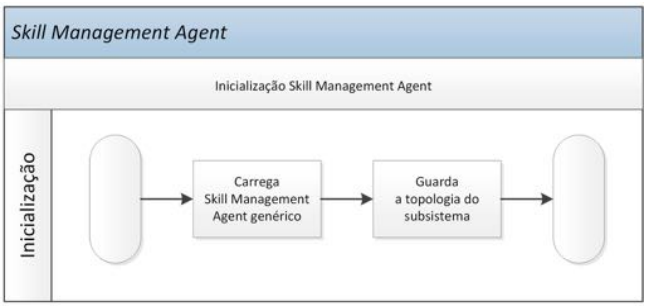

Figura 3.14: Inicialização do SMA.

Gestão de habilidades do subsistema

Tal como se pode verificar no subcapítulo 3.3.2, para que sejam desencadeados comportamentos no SMA, ou o PMA sofreu uma atualização na topologia, ou recebeu uma nova lista de regras, nestes casos o SMA recebe a topologia do subsistema ou a lista de regras, respectivamente.

Ao analisar a Figura 3.15 é visível que nestes dois comportamentos existem bastantes semelhanças. As diferenças são na fase inicial em que o SMA recebe a lista de regras e guarda-as ou quando recebe uma nova topologia, embora neste caso o agente não descarte a topologia recebida anteriormente, pois irá servir como estado de comparação para verificar se existiram alterações.

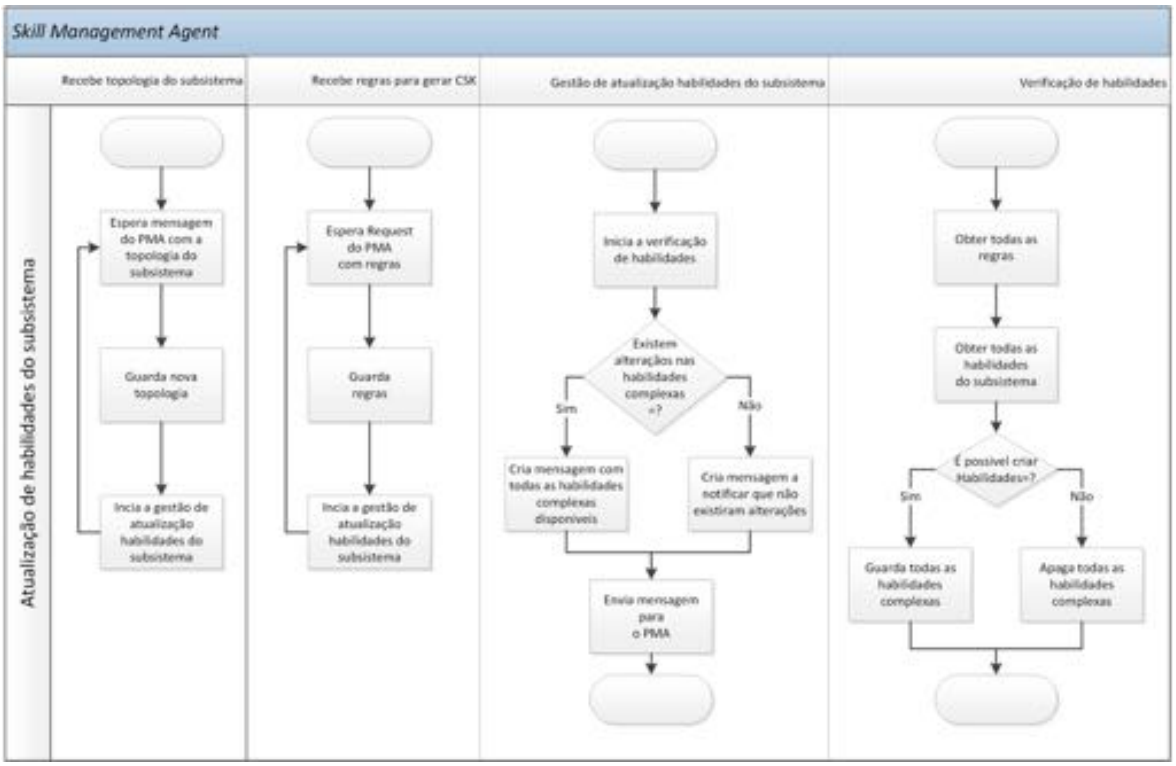

Figura 3.15: Gestão de habilidades no SMA.

Em ambos os casos o comportamento relacionado com a gestão das habilidades é desencadeado.

Com base na topologia do subsistema o SMA verifica quais as habilidades do subsistema. Estas habilidades pode ser habilidades de baixo nível (SSK) disponibilizadas pelos CA associados ao subsistema ou habilidades de alto nível (CSK) disponibilizadas pelos PMAs associados ao seu subsistema.

Baseado nas regras e nas habilidades disponibilizadas, é verificado se é possível gerar CSK, para isso é necessário que as habilidades disponibilizadas satisfaçam os requisitos exigidos pelas regras.

Antes de terminar o processo é verificado se é possível habilidades de nível ainda superior, para isso as CSK geradas anteriormente têm de satisfazer requisitos exigidos nas regras. Isto permite disponibilizar habilidades de nível ainda superior, aumentando ainda

mais a granularidade do sistema.

Depois de verificar quais as habilidades que o subsistema pode disponibilizar o SMA verifica com base no estado anterior do subsistema, se existem alterações ao nível das CSK, e caso existam é enviada a nova lista de habilidades que o subsistema pode disponibilizar, caso não existam o PMA é notificado desse resultado.

Esta verificação torna-se de extrema importância pois é ela que garante a estabilidade do sistema, caso contrário o PMA estaria constantemente a ser notificado de alterações na topologia e, assim, o sistema estaria em constante atualização.

Capítulo 4

Implementação

Este capítulo tem como finalidade explicar os detalhes de implementação, para o desenvolvimento do módulo central da arquitetura apresentado no capítulo 3.

O subcapítulo seguinte consiste numa breve descrição das tecnologias de suporte, utilizadas na implementação do trabalho proposto, sendo que os, restantes visam clarificar todos os detalhes de implementação com base nas tecnologias de suporte apresentadas.

4.1 Tecnologias de Suporte

Para a implementação da arquitetura proposta foi utilizada a linguagem de programação Java, pelo facto de ser uma linguagem de programação orientada a objetos e também pelo facto da plataforma utilizada para a implementação de uma arquitetura baseada em MAS, apresentada no subcapítulo 4.1.1, utilizar esta linguagem de programação.

4.1.1 *Java Agent Development Framework*

Java Agent Development Framework (JADE) é uma plataforma *open source* que suporta um modelo com base no paradigma de agentes, completamente implementado na linguagem Java [JAD14].

Uma implementação baseada em JADE permite o desenvolvimento de aplicações com elevada versatilidade e interoperabilidade, permitindo uma programação distribuída no desenvolvimento de aplicações ponto-a-ponto, em que a comunicação entre os agentes é assegurada independentemente, do tipo de ligação à rede, com ou sem fios [BCG07].

Esta plataforma suporta a coordenação entre vários agentes através de especificações *Foundation for Intelligent Physical Agents* (FIPA) e fornece uma implementação padrão da Linguagem de Comunicação de Agentes FIPA (FIPA-ACL) [FIP15a].

JADE fornece o suporte para a execução de múltiplas, paralelas e simultâneas atividades do agente através do modelo de *behaviour*, onde o comportamento do agente é agendado de forma não preemptiva. A Tabela 4.1 os *Behaviours* utilizados na implementação deste trabalho, bem como uma breve descrição de cada um.

Tabela 4.1: Principais *Behaviours* utilizados e sua descrição.

Java Class	Descrição
One Shot Behaviour	Executa uma única ação.
Simple Behaviour	Executa uma ação contínua. A execução pode ser terminada com recurso a uma variável *boolean*.
Sequential Behaviour	Agenda a execução de *sub-behaviours* sequencialmente. Uma vez executado o primeiro *sub-behaviours* é executado o seguinte e assim sucessivamente até ao último.
Parallel Behaviour	Agenda a execução de *sub-behaviours* paralelamente.
AchieveREResponder	Responde a uma comunicação *FIPA Request Protocol* [FIP15b].
AchieveREInitiator	Inicia uma comunicação *FIPA Request Protocol* [FIP15b].

4.1.2 *H2 Database Engine*

O *H2 Database Engine* [H2d14] é um sistema de gerenciamento de base de dados relacional desenvolvido em Java. Este sistema pode ser incorporado em aplicações Java, ou pode ser executado segundo o modelo cliente-servidor.

Esta tecnologia é *open source* e suporta os standards *SQL, JDBC API*.

4.1.3 *Web Ontology Language*

A *Web Ontology Language* (OWL) é uma linguagem para definir e instanciar ontologias e pode incluir descrições de classes, as suas respetivas propriedades e seus relacionamentos. A OWL é baseada em XML, sendo que isto facilita a troca de informações entre máquinas independentemente do sistema operativo ou da linguagem de programação a ser utilizada [AVH04].

4.2 Implementação do trabalho proposto

4.2.1 Modelo Semântico

Para a implementação deste trabalho foi utilizado um modelo semântico desenvolvido pelo UNINOVA [UNI15], com base em OWL.

O modelo semântico é um modelo que ajuda a definir dados e o relacionamento entre diferentes entidades. O conjunto total dessas entidades constitui a taxinomia de classes usadas no modelo para descrever todo o sistema.

Para a implementação deste trabalho foram utilizadas algumas classes definidas no modelo semântico, na Figura 4.1 está representado o diagrama de classes parcial do modelo semântico, sendo que as classes ilustradas foram as que contribuíram para a implementação deste trabalho.

Na Figura 4.1 onde está representado o diagrama de classes, são destacados alguns objetos, devido à sua importância na implementação, pelo que serão referidos ao longo deste capítulo.

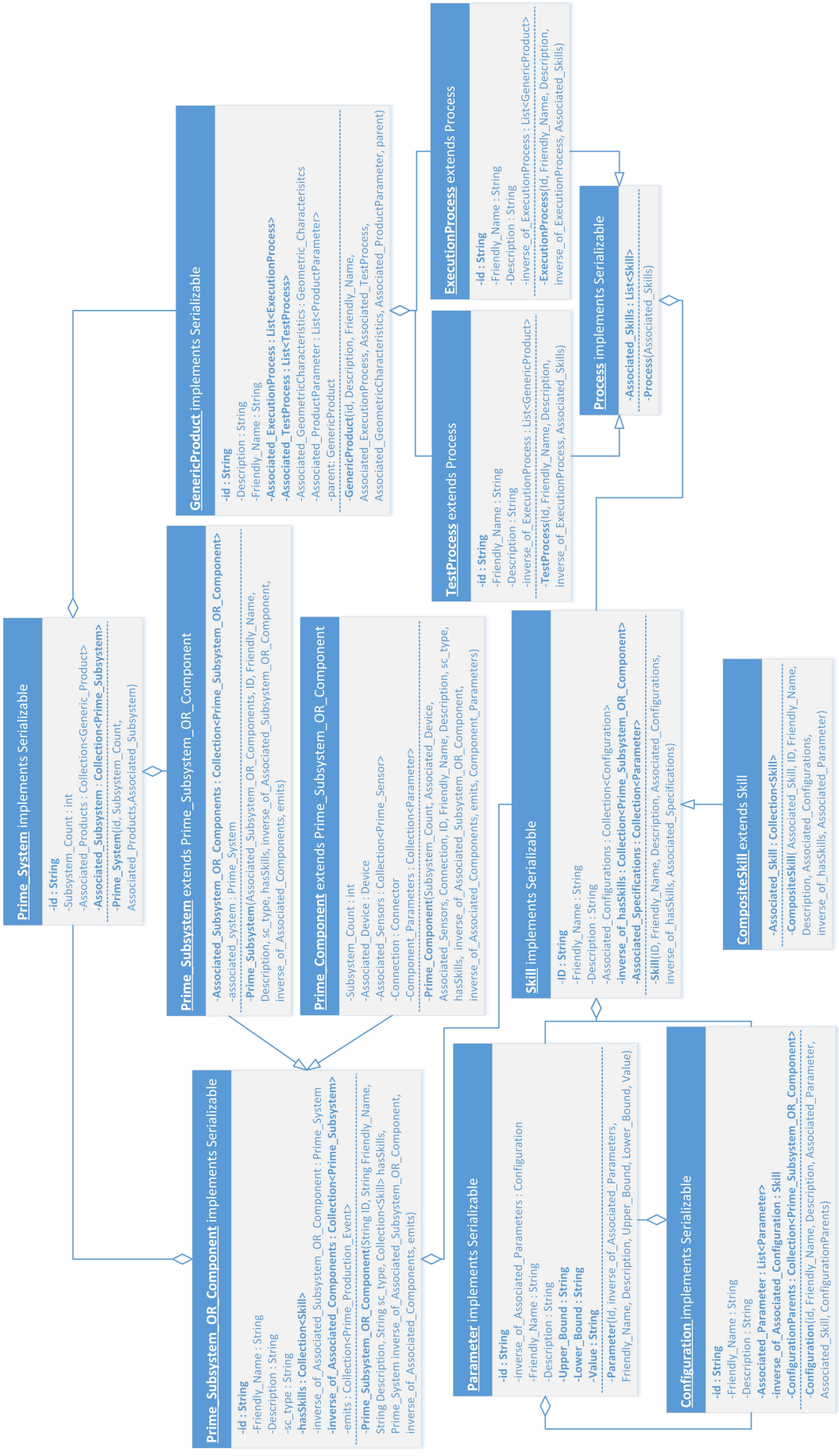

Figura 4.1: Diagrama de classes parcial do Modelo Semântico.

4.2.2 Serialização e deserialização de objetos

Visto que Java é uma linguagem de programação orientada a objetos, é muito comum por vezes recorrer a métodos que facilitem o armazenamento ou a transmissão desses objetos, num formato de dados compatível.

Como JADE foi a plataforma escolhida para a implementação de agentes e, dado que, que esta se rege pelas especificações FIPA, em que os conteúdos das mensagem são do tipo *String*, foi necessário recorrer a uma biblioteca que serializa e deserializa objetos em Java, independentemente do tipo de objeto em causa.

Assim, o processo de serialização consiste em transformar objetos Java numa *String* e a deserialização como o processo inverso, ou seja, transformar uma *String* num objeto Java.

Estes processos vieram a revelar-se de extrema importância pois facilitaram não só a transmissão de informação entre agentes, como também na monitorização da base de dados, em que objetos Java eram serializados e guardados, e quando posteriormente fossem acedidos era feito o processo inverso de deserialização.

4.2.3 Base de Dados

Tal como se pode verificar na arquitetura apresentada no capítulo 3, existe uma base de dados que é manipulada pelo PSA. Esta base de dados foi implementada com base na tecnologia *H2 Database Engine*, apresentada no subcapítulo 4.1.2.

Na Figura 4.2 está representado o modelo da base de dados implementado.

O modelo apresentado na Figura 4.2 é referente à base de dados que é criada e manipulada pelo PSA, e como se pode verificar este é constituído por nove tabelas.

Todas as tabelas implementadas têm como chave primária um *id* que é um iden-

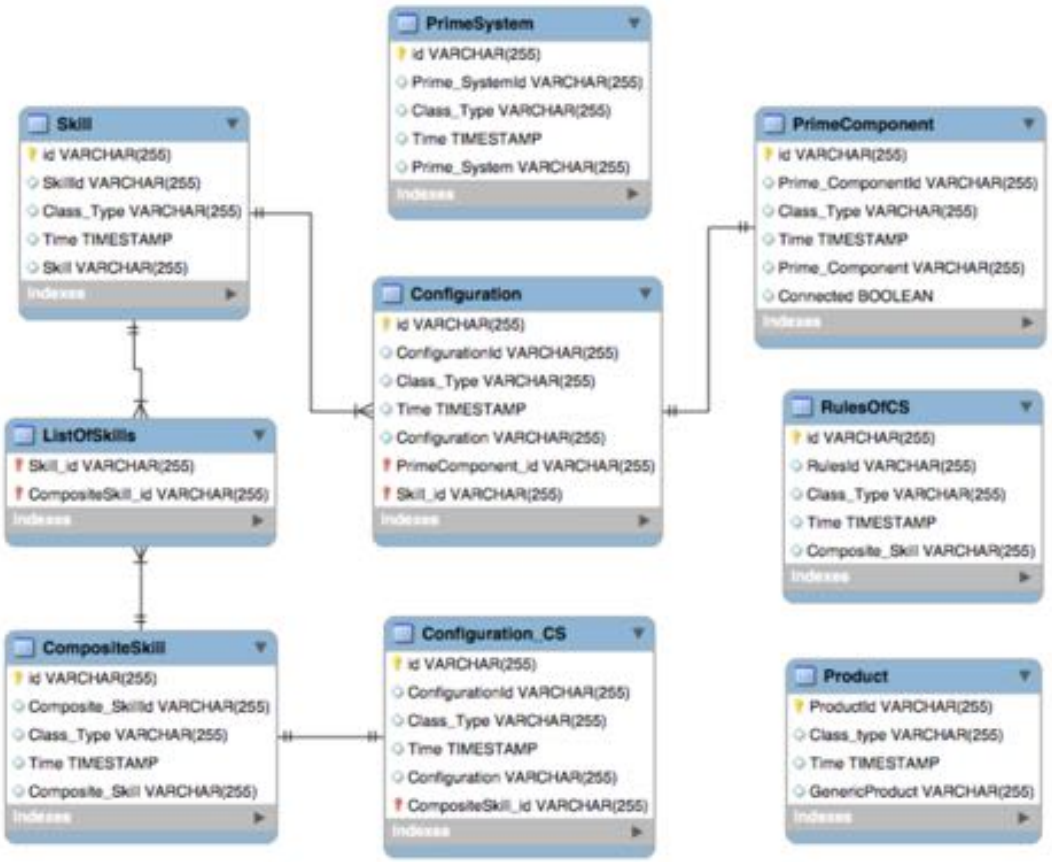

Figura 4.2: Base de dados.

tificador único do tipo *VARCHAR(255)*, um elemento *Time*, que tem a informação de quando é que a tabela foi criada, do tipo *TIMESTAMP* e também um elemento denominado de *ClassType* que representa o tipo de classe que instancia cada um dos objetos guardados, do tipo *VARCHAR(255)*, os restantes elementos são específicos de cada uma das tabelas. Cada tabela tem como finalidade guardar um determinado tipo de objetos, para que tal seja possível, todos os objetos são serializados e guardados sob a forma de *VARCHAR(255)*.

Para um melhor entendimento de cada das tabelas presentes na Figura 4.2, será elaborada uma breve descrição de cada uma delas.

PrimeSystem

Esta tabela tem como objetivo guardar objetos do tipo *Prime_System*, permitindo a qualquer momento ter acesso ao estado do sistema numa data anterior.

PrimeComponent

Esta tabela permite guardar objetos do tipo *Prime_Component*, assim é possível obter o histórico de todos os componentes que já fizeram parte do sistema. Existe uma variável do tipo *BOOLEAN*, o *Connected*, que representa se o componente abstraído por aquele *Prime_Component* está ligado ou desligado, *true* ou *false*, respetivamente.

RulesOfCS

Esta tabela representa as regras introduzidas para gerar CSK, sendo que o objeto guardado é um *Composite_Skill*. Este objeto contém uma lista de SSK necessárias para gerar uma CSK.

Product

Esta tabela permite guardar todos os produtos para o qual se pretende (re)configurar o sistema, estes produtos são guardados segundo objetos do tipo *Generic_Product*.

Skill

Esta tabela permite guardar todas as SSK até então disponibilizadas pelo sistema.

CompositeSkill

Esta tabela permite guardar todas as CSK até então disponibilizadas pelo sistema.

ListOfSkills

As CSK são geradas segundo um conjunto de SSK. Através de duas chaves estrangeiras, permite descrever quais as SSK necessárias para criar determinada CSK.

Configuration

Associada a uma configuração existe uma SSK e um *Prime_Component*, assim esta tabela guarda um objeto do tipo *Configuration*, com duas chaves estrangeiras que a associa a um *Skill* e a um *Prime_Component*.

Configuration_CS

Tal como as SSK também as CSK podem ter associadas uma configuração, esta tabela permite guardar objetos do tipo *Configuration*, e através de uma chave estrangeiras associa-la a uma CSK.

4.2.4 Modelo de dados

Tal como foi referido no subcapítulo 4.1, para a implementação deste trabalho recorreu-se à linguagem de programação Java. Na Figura 4.3 está esquematizado o modelo de dados implantado, onde estão representadas os *Behaviours* e métodos utilizados, bem como o relacionamento entre as diferentes entidades.

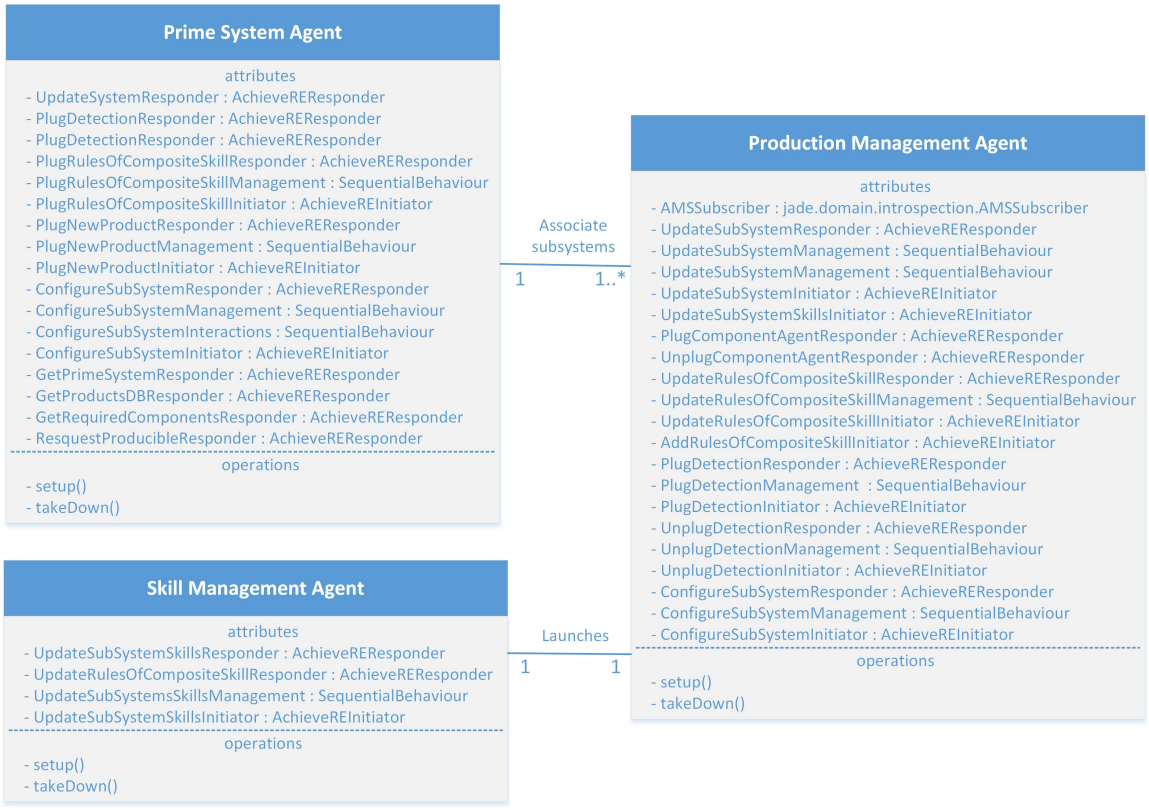

Figura 4.3: Modelo de dados.

Este modelo de dados complementa o Modelo Semântico apresentado no subcapítulo 4.2.1. Nos seguintes subcapítulos os *Behaviours* apresentados serão caracterizados de acordo com os diversos processos que acontecem no sistema.

4.2.5 Detalhe de implementação dos diferentes Comportamentos

Para garantir a integridade de todo o sistema e da sua informação, foram implementados diferentes comportamentos. As seguintes seções visam esclarecer os detalhes de implementação de cada um destes comportamentos.

Serão apresentados alguns diagramas de sequência, é importante salientar que todas as comunicações entre os agentes regem-se pelo protocolo *FIPA-Request* [FIP15b], representado na Figura 4.4.

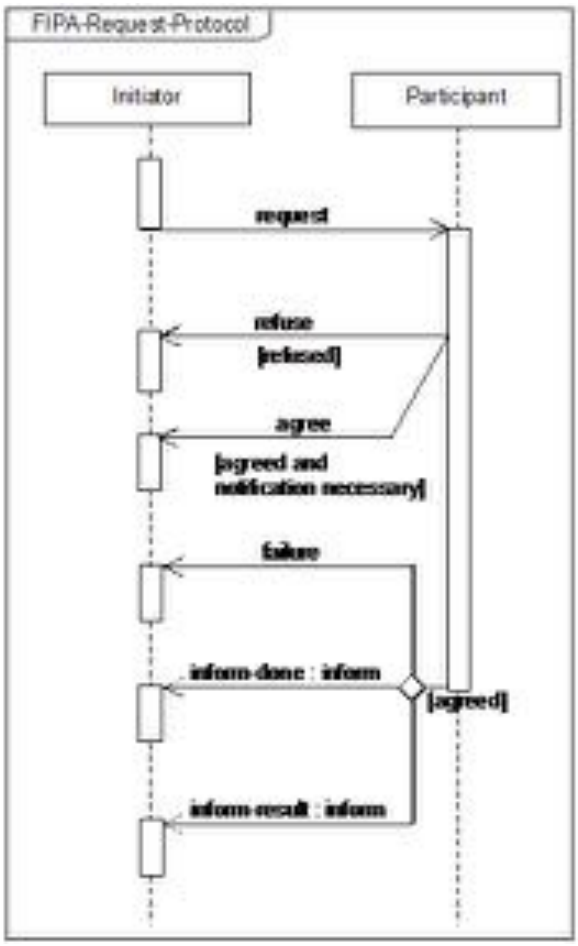

Figura 4.4: Interações do protocolo *FIPA-Request*, retirado de [FIP15b].

O protocolo *FIPA-Resquest* permite que a um agente solicitar outro para executar determinada ação. Tal como se pode verificar na Figura 4.4, o *Participant*, processa um *request* enviado pelo *Initiator* a fim de executar determinada ação. O *Participant* processa

o *request* e decide se o aceita ou recusa, consoante essa decisão, responde com *refuse* em que a comunicação entre os agentes termina ou então pode comunicar um acordo com o *Initiator* [FIP15b].

No caso de o *Participant* comunicar um acordo responde ao *request* recebido com um *agree* e comprometesse a processar a decisão e notificar o seu resultado, no caso de não ser necessário um acordo pode responder com logo com *failure*, *inform-done* ou *inform-result* [FIP15b].

Uma resposta com um *failure* significa que falhou a tentativa de processar o *Request*, com um *inform-done* significa que conclui com êxito o processamento do *Request* ou com um *inform-result* caso seja necessário notificar o resultado do processamento do *Request* [FIP15b].

4.2.6 Criação da árvore de PMAs

Para que seja possível configurar, adicionar ou remover CA é necessário primeiro criar a árvore de PMAs. Uma vez que o agente de mais alto nível é o PSA este é o primeiro a ser lançado, sendo que posteriormente podem ser lançados os PMAs.

Quando o PSA é lançado na plataforma este cria e guarda um objeto do tipo *Prime_System* (Figura 4.1). O *id* deste objeto é sempre igual ao nome do agente que o instanciou.

O *Prime_System* é atualizado sempre que necessário, é nele que está a informação de todo o sistema. Como é este agente o responsável pela gestão do modelo semântico e da base de dados, quando o agente é lançado também carrega ambos os repositórios e atualiza-os sempre que necessário.

Sempre que um PMA é lançado na plataforma, é necessário passar como parâmetro o nome do agente ao qual este se vai ligar, este nome é guardado numa *String* e de futuro

todas as comunicações efetuadas para a camada superior são efetuadas para o agente com esse nome. Quando um PMA comunica com o agente de nível superior na sua grande maioria envia a topologia do subsistema que este abstrai, esta topologia é representada por um objeto do tipo *Prime_Subsystem* em que o seu *id* corresponde ao nome do agente que o instanciou. Este objeto é criado e guardado quando o agente é lançado na plataforma e atualizado quando necessário.

Associado a um PMA existe sempre um SMA, quando o PMA é inicializado é inicializado em simultâneo um SMA. O PMA guarda numa *String* nome do SMA, para de futuro saber qual dos SMAs é que gere as CSK do seu subsistema.

Na Figura 4.5 está representado o diagrama de sequência de mensagens trocadas por PSA, PMAs e SMAs ao longo da criação da árvore de agentes. A troca de mensagens entre PSA, PMAs de alto nível e SMAs de alto nível é representativa também da atualização do sistema quando existem alterações na topologia de PMAs de baixo-nível, como se vai verificar ao longo do presente capítulo.

Para garantir a consistência da árvore e de todo o sistema é necessário garantir que o agente cujo o nome é passado por parâmetro para o PMA, exista. Assim é expetável que ao primeiro PMA a ser inicializado seja passado como parametro o nome do PSA, aos restantes tanto pode ser associado o PSA como um PMA. Um PMA só se pode ligar a um agente, (PSA ou PMA), mas a ele podem ser ligados quantos agentes sejam necessários.

Como se pode verificar na Figura 4.5 quando um PMA é inicializado os primeiros comportamentos desencadeados são: o envio de um Request com o *Prime_Subsystem* para o agente de mais alto nível e a subscrição do Agente de Gestão do Sistema (AMS), este comportamento será explicado no subcapítulo 4.2.11.

Se o agente de mais alto nível for o PSA este adiciona o *Prime_Subsystem* recebido a uma *Collection* de *Prime_Subsystems*. Esta *Collection* contém todos os *Prime_Subsystems*

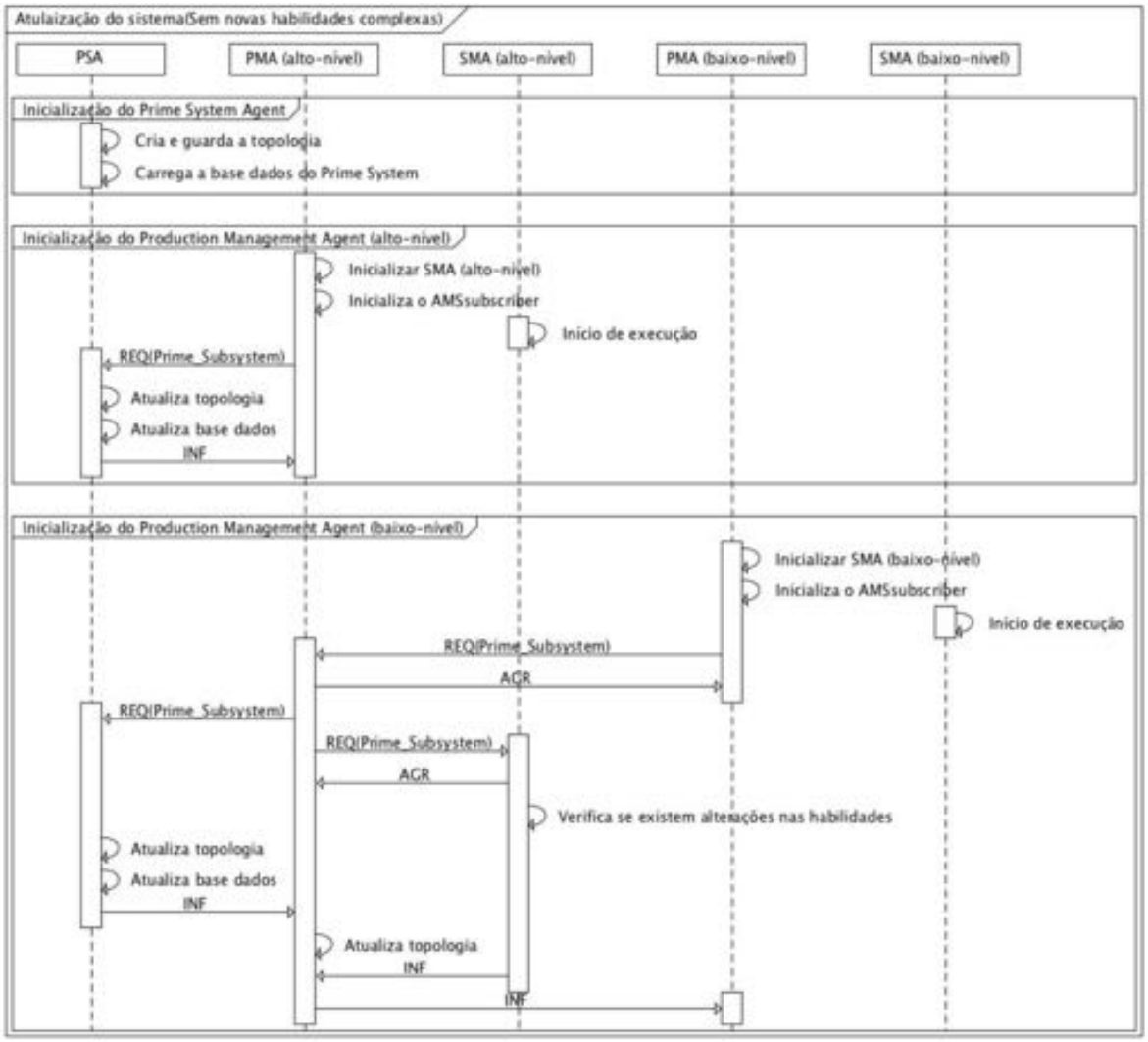

Figura 4.5: Diagrama de sequência da criação da árvore de agentes.

associados ao PSA. Quando este *Prime_Subsystem* for enviado novamente, é substituído pois significa que existiram alterações na sua topologia, garantindo assim que a informação se mantém atualizada.

Depois de atualizado o *Prime_System*, o PSA verifica se existem objetos do tipo *Prime_Component* ou *Skill* e se existirem verifica se nunca foram adicionados à base de dados, e adiciona-os. Posto isto, o PSA envia um *Inform* ao PMA para o notificar do sucesso da actualização do sistema.

Na Figura 4.6 estão descritos os comportamentos desencadeados pelo PSA quando é

recebido um Request com um *Prime_Subsystem*, bem como o *Behaviour* utilizado na sua implementação.

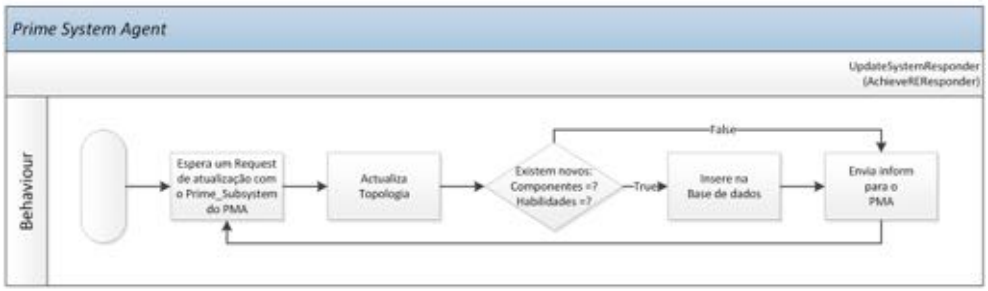

Figura 4.6: Comportamento do PSA aquando da atualização do sistema.

Se o agente de mais alto nível for um PMA este adiciona o *Prime_Subsystem* recebido a uma *Collection* de *Prime_SubSystem_OR_Components* do seu *Prime_Subsystem* (Figura 4.1). O *Prime_Subsystem* e o *Prime_Component* estendem a classe do tipo *Prime_SubSystem_OR_Component*, assim é nesta *Collection* que o PMA guarda os objetos enviados pelos agentes de mais baixo nivel, quer sejam PMAs ou CAs.

O *Prime_Subsystem* é enviado paralelamente para o agente de mais alto nível e para o SMA responsável pela gestão das habilidades disponibilizadas pelo subsistema. Assim, garante-se a coerência do sistema, pois independentemente do tempo de processamento associado gestão das habilidades por parte do SMA o sistema, mantém-se atualizado.

Quando um PMA recebe um Request com um *Prime_Subsystem*, este cria uma cópia do seu *Prime_Subsystem*, e é a este objeto que adiciona o *Prime_Subsystem* recebido enviando-o para o agente de mais alto nível. Só depois de receber um *Inform* desse agente é que o PMA assume que aquele é o seu *Prime_Subsystem*, assim é garantida a consistência da informação, pois um subsistema só é atualizado depois dos agentes de mais alto nível atualizarem sua topologia.

Na seguinte Figura 4.7 é possível verificar estes procedimentos bem como os *Behaviours* necessários para a sua implementação.

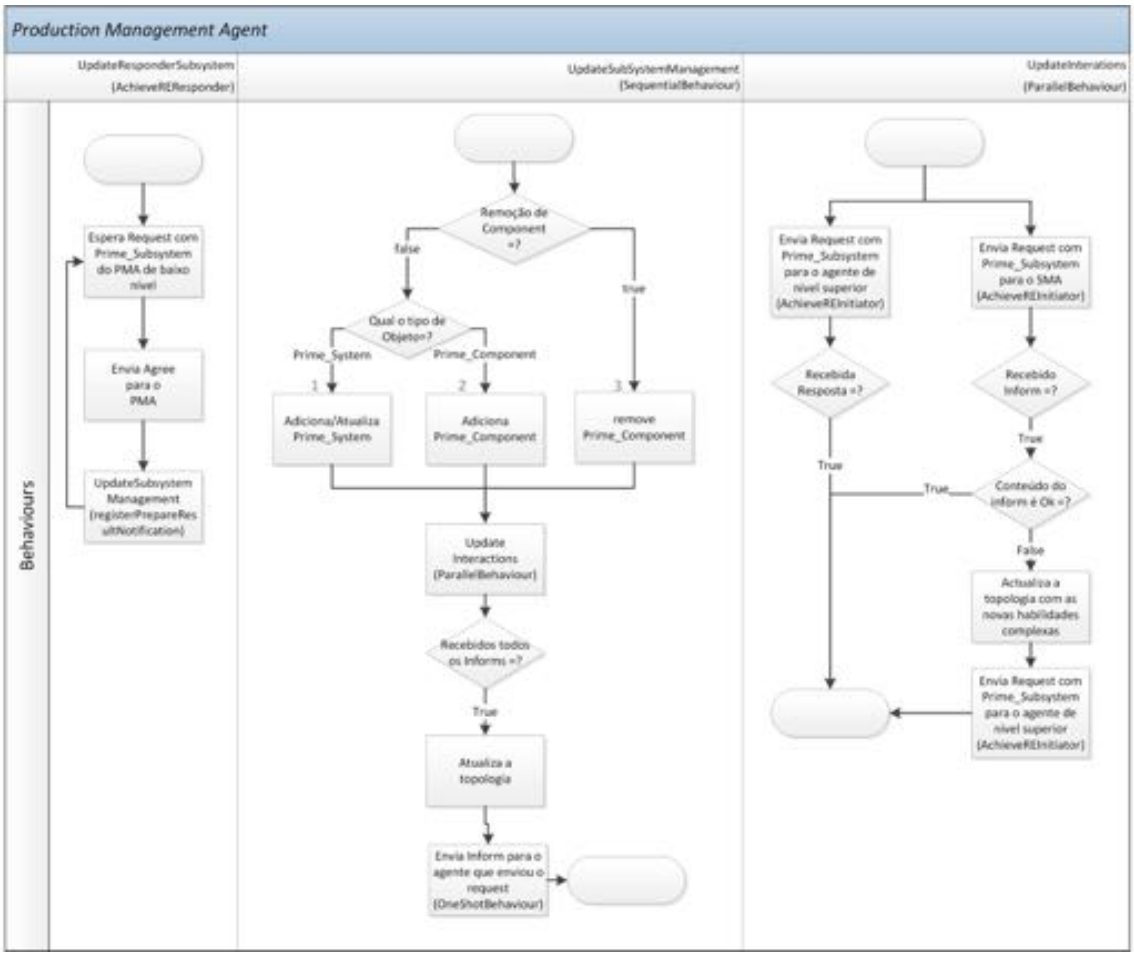

Figura 4.7: Comportamento do PMA aquando da atualização do subsistema.

Como se pode verificar na Figura 4.7 está esquematizado a implementação de um *SequentialBehaviour*. Este *Behaviour* permite que o PMA só atualize a topologia depois de receber um *Inform* do agente de mais alto nível e, posteriormente, enviar um *Inform* à camada inferior a notificar que até ao seu subsistema todo o sistema está atualizado.

Sempre que o PMA recebe um *Request* para atualizar o seu *Prime_Subsystem*, o *SequentialBehaviour* (*UpdateSubSystemManagement*) é prescrito, como tal, é necessário identificar os motivos que desencadearam a atualização. Como não se trata da remoção de um *Prime_Component*, através do comando *instanceof* é possível receber um *Prime_Sub-System_OR_Components* e identificar o tipo de objeto recebido. Como neste caso é um *Prime_Sub- System* (ponto 1 da Figura 4.7), é criado um objeto igual ao objeto

Prime_subsystem que contém a topologia do subsistema, sendo este objeto atualizado com a informação recebida.

Da forma como a arquitetura foi definida a comunicação ascendente entre agentes é garantida, pois cada PMA tem associado um agente de nível superior. A comunicação descendente já não o é, sendo esta essencial no processo de (re)configuração de componentes.

Quando o PMA recebe um *Prime_Subsystem* este acede a todos os objetos do tipo *Prime_Component* nele guardados. Na *Collection* de *Prime_Subsystem*, denominada *inverse_of_Associated_Components*, adiciona o seu *Prime_Subsystem*. O mesmo acontece aos objetos do tipo *Skill*, na *Collection* de *Prime_SubSystem_OR_Component* denominada *inverse_of_hasSkills*, (Figura 4.1).

Sendo este processo repetido em todos os níveis de PMAs nestas *Collections* estão guardados os *Prime_Subsystems*, de cada um deles. Fornecendo assim o caminho inverso desde o PSA até ao PMA que tem associado os *Prime_Component* ou a *Skill*.

De seguida, o objeto criado é serializado e enviado num *Request* para o PMA de mais alto nível e para o SMA, como esta comunicação é paralela, recorreu-se a um *ParallelBehavior*, que permite que o PMA não fique bloqueado à espera de um *Inform* do SMA para finalizar o processo de atualização.

Quando o SMA recebe um *Request* do seu PMA, são desencadeados nele comportamentos para o processamento das CSK disponibilizadas pelo subsistema, neste caso considera-se que não existem alterações. Assim o PMA recebe do SMA um *Inform* com o conteúdo *"Ok"*. Verificando o conteúdo dessa mensagem não é desencadeado nenhum tipo de comportamento. Sendo o caso contrario comum a outros processos de atualização, este será abordado no subcapítulo 4.2.12.

4.2.7 Inserção de regras para gerar habilidades complexas

Para que os subsistemas possam disponibilizar CSK é necessário fornecer a todos os SMAs do sistema as regras para gerar as CSK. Tal como uma CSK (Figura 4.1), uma regra também é um objeto do tipo *CompositeSkill*, este objeto tem uma *Collection* de *Skills* onde é possível associar SSKs e uma *Collection* de *Parameters*s onde é possível associar os parâmetros exigidos para gerar a CSK.

Assim como as regras estão na base de dados este processo é iniciado no PSA, tal como demonstra a Figura 4.8.

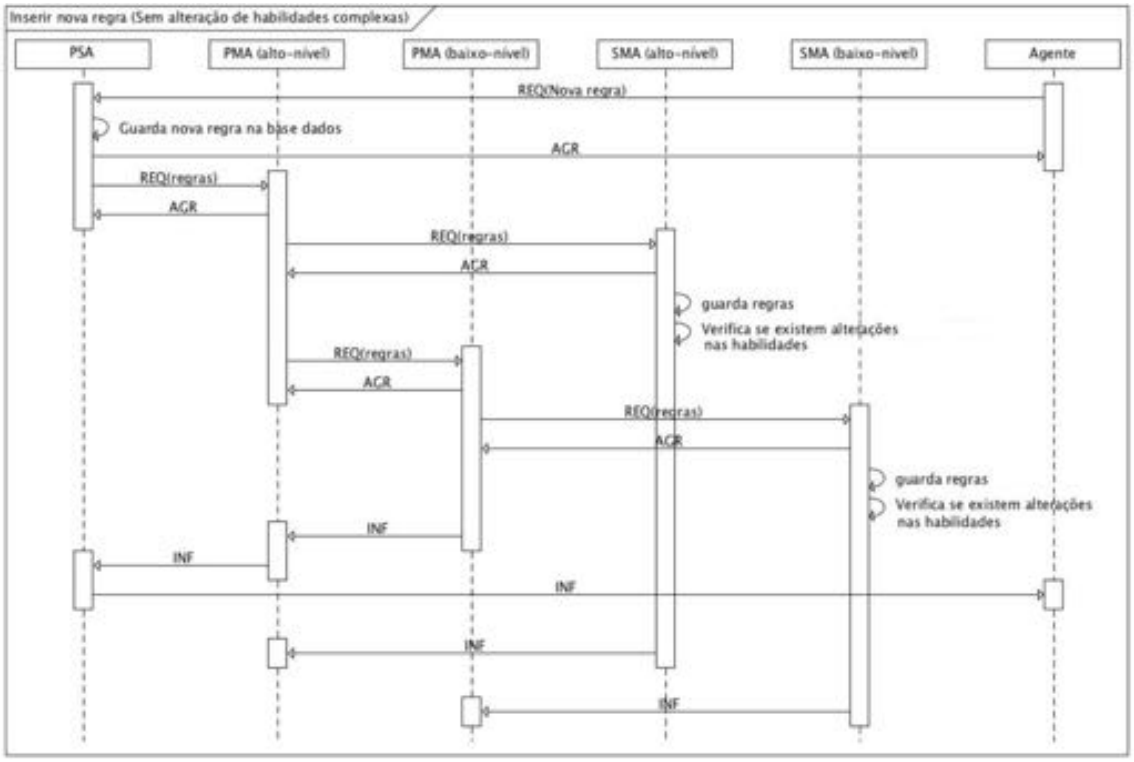

Figura 4.8: Diagrama de sequência da inserção de regras para gerar CSK.

Quando o PSA recebe um *Request* com uma regra, verifica se a regra já existe na base de dados, caso não exista guarda-a e dá início à comunicação com os PMAs de nível inferior, enviando todas as regras. Caso contrário, responde com um *Failure* para garantir que não existem regras repetidas na base de dados.

Esta validação da regra consiste em verificar se não existe uma regra guardada que tenha a mesma *Collection* de *Skills* e *Parameters*, evitando a existência de informação redundante.

Considerando o caso que a regra recebida não consta na base de dados, são desencadeados alguns *Behaviours* no PSA, tal como mostra a Figura 4.9.

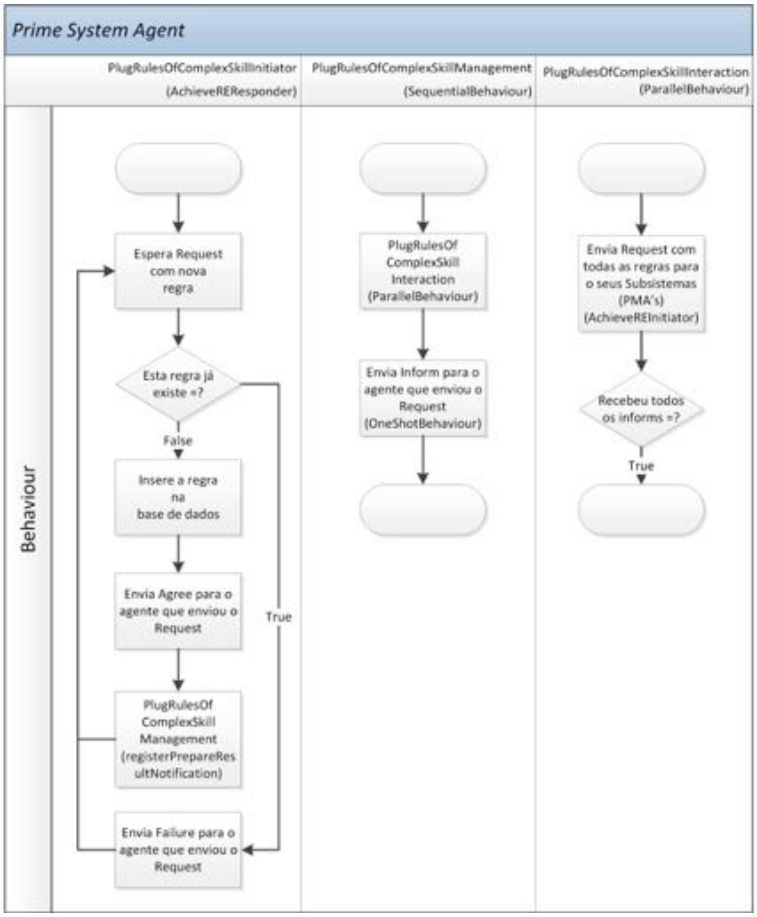

Figura 4.9: Comportamento do PSA ao receber uma regra.

Para garantir que todos os SMAs são atualizados com sucesso, foi implementado um *SequentialBehavior* e um *ParallelBehaviour* (Figura 4.9), o *ParallelBehaviour* permite que no caso de o PSA ter associado mais do que um PMA possa enviar um *Request* em paralelo para os PMAs. Como o *ParallelBehaviour* foi iniciado no *SequentialBehavior*, só depois do PSA receber todos os *Informs*, ou seja, quando o *ParrallelBehavior* terminar, é iniciado um *OneShotBehaviour* que envia um *Inform* para o agente que enviou a regra.

Visto que podem existir várias camadas de PMAs e que cada um tem associado um SMA, foram implementados no PMA os mesmos *Behaviours*, tal como se pode verificar na Figura 4.10.

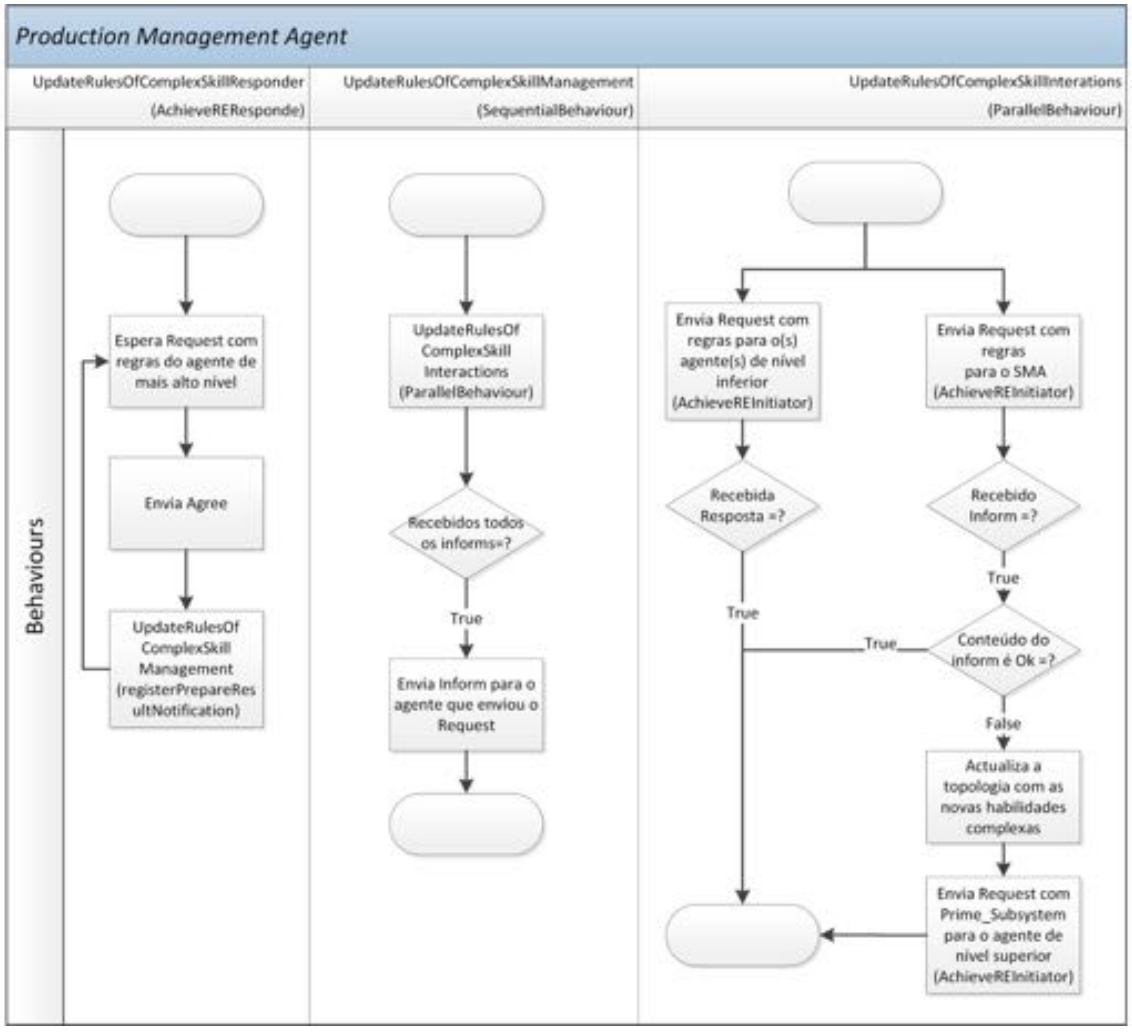

Figura 4.10: Comportamento do PMA ao receber regras.

O objetivo do PMA quando recebe regras do agente de nível superior, é enviar um *Request* para o SMA que gere as CSK por ele disponibilizadas e para os PMAs a ele associados. Para isso é necessário aceder à *Collection* de *Prime_SubSystem_OR_Components* e verificar através com comando *instanceof* quais os objetos do tipo *Prime_Subsystem*, enviando um *Request* para os agentes cujo o nome é o *id* desse *Prime_Subsystem*. Quando

o PMA recebe o *Inform* dos PMAs de nível inferior envia um *Inform* para o agente de nível superior, notificando assim que todos os subsistemas abaixo dele estão atualizados. Todo este processo repete-se até um PMA que não contenha subsistemas associados.

À semelhança do que aconteceu no PSA, no PMA também foi implementado um *SequentialBehavior* e um *ParrallelBehavior*, com a particularidade que o *ParrallelBehavior* permite enviar o *Request* em paralelo para os PMAs de mais baixo nível e para o SMA. Garantindo que o sistema é atualizado independentemente do tempo que o SMA demora a processar as CSK.

Tal como no caso anterior quando o SMA recebe uma regra, podem existir alterações nas CSK disponibilizadas, neste caso assume-se que não existem alterações, pois sendo esse um comportamento comum na atualização do sistema é explicado no subcapítulo 4.2.12.

4.2.8 Deteção de adição e remoção de componentes

Aos subsistemas podem ser adicionados CAs que abstraem componentes, uma vez adicionados também podem ser removidos, mas antes de qualquer um destes processos é feita uma validação do processo.

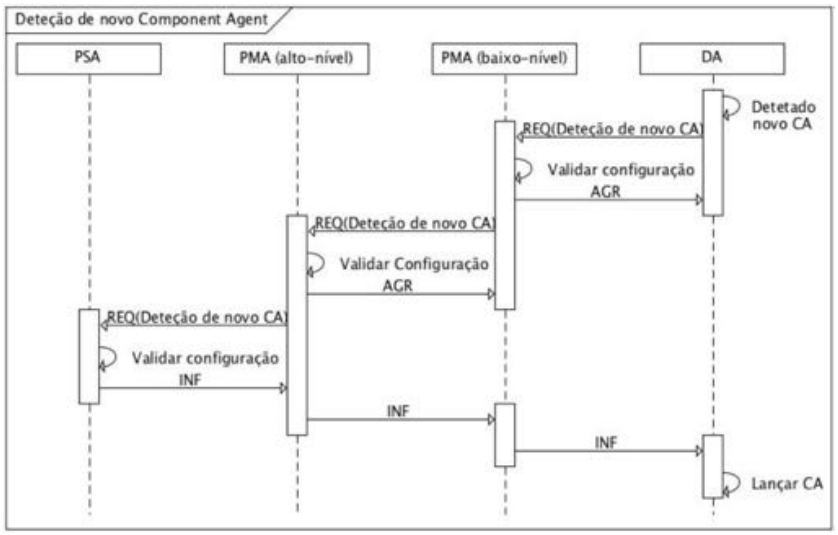

Figura 4.11: Diagrama de sequência da deteção de novo CA.

Na Figura 4.11 está representado o diagrama se sequência referente à deteção de novo componente. Antes de ser adicionado um componentes a um subsistema, o DA que vai lançar o CA envia um *Request* para o subsistema ao qual vai ser associado o CA.

Quando o PMA recebe o *Request* verifica se é possível adicionar o CA ao seu subsistema, caso não seja possível este envia um *Failure* caso contrario envia um *Request* para o agente de nível superior, este processo repete-se até ao PSA.

Quando um PMA recebe um *Inform* do agente de mais alto nível significa que a verificação nas camadas superiores foi bem sucedida, assim envia um *Inform* para o agente que desencadeou o comportamento, este processo repete-se até o DA receber um *Inform*. Para implementar este mecanismo recorreu-se a um *SequentialBehavior*.

Caso o DA receba um *Inform* lança o CA, aquando desta operação é passado como parâmetro ao CA o nome do PMA que validou a sua configuração, caso contrário o DA recebe um *Failure* e não é lançado nenhum CA.

O processo que antecede a remoção de um CA é idêntico.

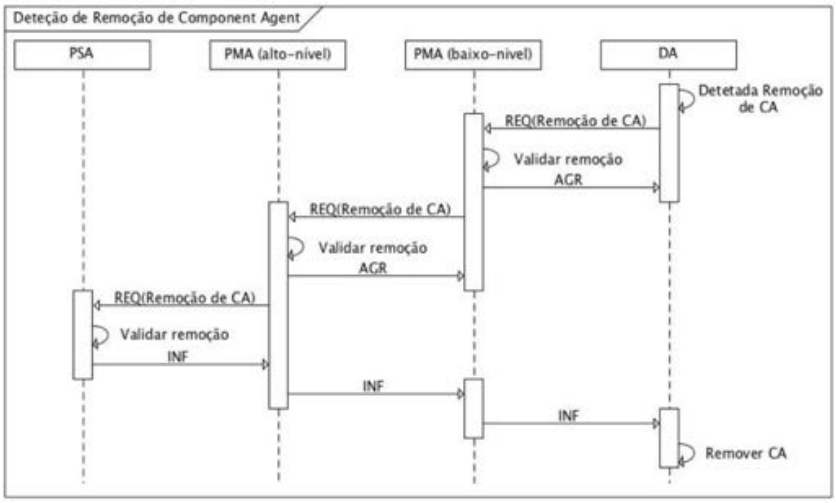

Figura 4.12: Diagrama de sequência da deteção de remoção de um CA.

Na Figura 4.12 está esquematizado o diagrama de sequência que representa a comunicação entre os agentes aquando da deteção da remoção de um CA. A implementação é idêntica à anterior, com a pequena diferença de que quando o DA recebe o *Inform* é para notificar o CA que pode iniciar o processo de remoção.

Depois do PMA enviar um *Inform* para o DA em qualquer um dos casos apresentados o PMA irá receber um *Request* de um CA, a implementação deste comportamento será explicada nos subcapítulos 4.2.9 e 4.2.10.

4.2.9 Adição de componentes num subsistema

Depois de concluída com sucesso a comunicação representada na Figura 4.11, o DA lança na plataforma um novo CA e tal como, foi referido este CA recebe o nome do PMA, a fim de se registrar no subsistema que este representa.

Na Figura 4.13 está representado o diagrama de sequência que representa a troca de mensagens efetuada pelos agentes aquando da ligação de um CA a um subsistema.

Como se pode verificar quando um CA é lançado na plataforma este envia um *Request* com um *Prime_Component* para o PMA que representa o subsistema à qual ele irá pertencer. O *id* do *Prime_Component* corresponde ao nome do CA que o instanciou.

Na Figura 4.14 é possível visualizar os principais comportamentos bem como os *Behaviours* implementados, no registo do CA num subsistema

Quando um PMA recebe um *Request* de um CA o conteúdo da mensagem é deserializado e obtém-se o *Prime_Component*. O objetivo é guardar este *Prime_Component* na sua topologia e informar o agente de mais alto nível e o SMA desta alteração.

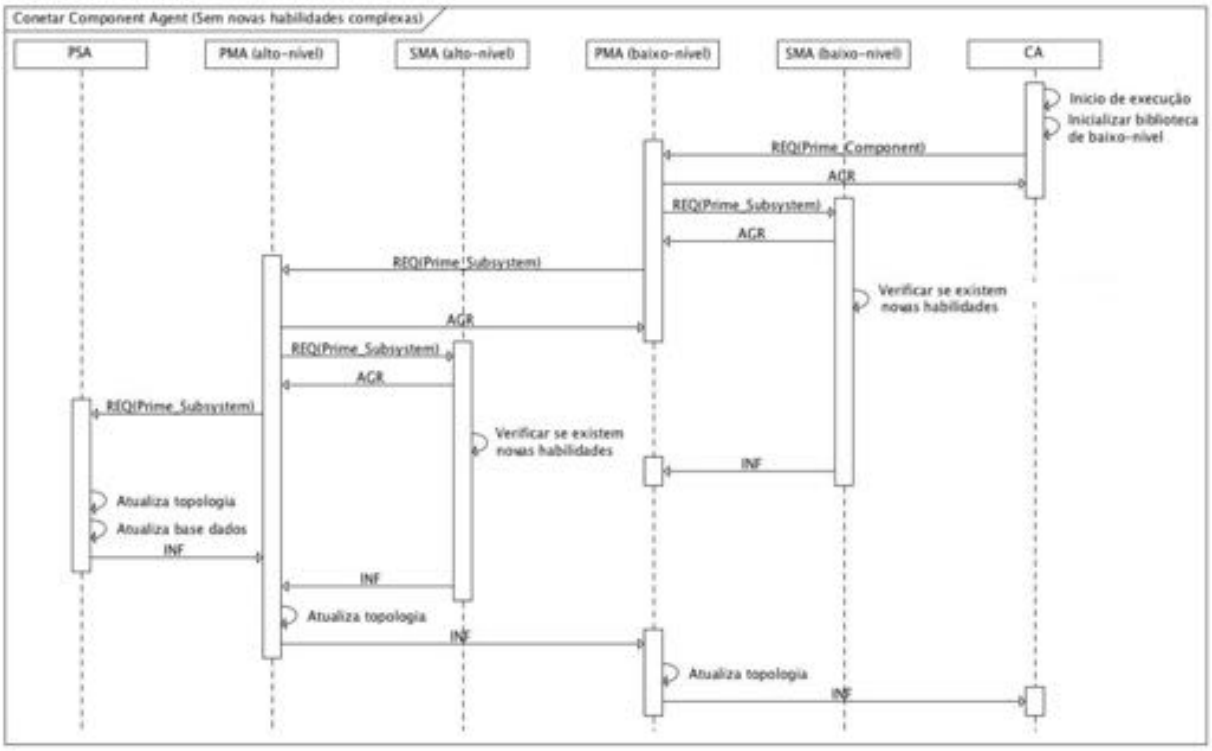

Figura 4.13: Diagrama de sequência da conexão de um CA a um subsistema.

À semelhança do que aconteceu anteriormente é prescrito novamente no PMA o *SequencialBehavior*, *UpdateSubsystemManagement*, é criada um objeto do tipo *Prime_Subsystem* igual ao que representa o subsistema, mas desta vez com a utilização do comando *instanceof* é identificado que o objeto *Prime_SubSystem_OR_Components* passado é do tipo *Prime_Component* então este é adicionado à *Collection* de *Prime_SubSystem_OR_Components* do objeto criado (ponto 2 da Figura 4.14).

À semelhança do que aconteceu no subcapítulo 4.2.6, o PMA adiciona o seu *Prime_Subsystem* na *Collection* de *Prime_Subsystems* do *Prime_Component* (*inverse_of_Associated_Components* Figura 4.1). Sendo este o primeiro *Prime_Subsystem* desta *Collection*, sabe-se em qualquer parte do sistema que este *Prime_Component* está associado e este subsistema.

Posto isto é paralelamente enviado um *Request* para o agente de mais alto nível e para o SMA com o *Prime_Subsystem* serializado. Quando este receber um *Inform* do agente

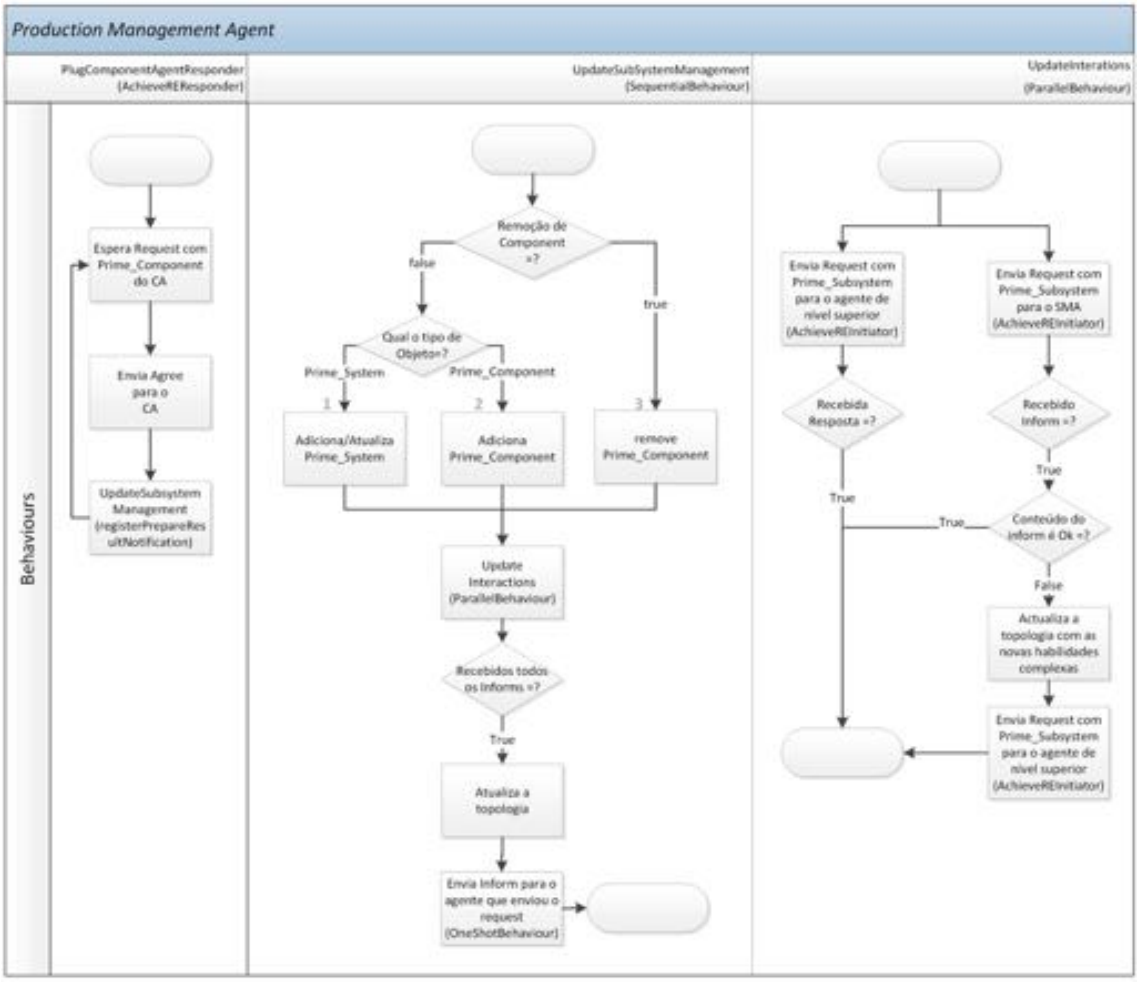

Figura 4.14: Comportamento do PMA quando recebe um *Request* para adicionar um CA.

de mais alto nível, atualiza a sua topologia e envia um *Inform* para o CA a notificar que este foi associado ao sistema com sucesso, este mecanismo deve-se à implementação do *SequencialBehavior* e do *ParallelBehavior* descritos na Figura 4.14.

Visto que o PMA que recebe o *Request* do CA envia o seu *Prime_Subsystem* serializado para o agente de mais alto nível, nele e nos restantes agentes até ao PSA são desencadeados os comportamentos descritos no subcapítulo 4.2.6.

Neste caso também são desencadeados comportamentos nos SMAs, tal como já foi referido, os detalhes de implementação serão explicados no subcapítulo 4.2.12.

4.2.10 Remoção de componentes do subsistema

Tal como é possível adicionar a qualquer momento um CA a um subsistema também é possível removê-lo. Para isso é necessário validar a remoção (subcapítulo 4.2.8). No caso de o DA receber um *Inform* do PMA dá-se início ao processo de atualização de todo o sistema e, posteriormente, a remoção do CA da plataforma.

Na Figura 4.15 está esquematizado o diagrama de sequência da troca de mensagens entre agentes aquando da remoção de um CA da plataforma.

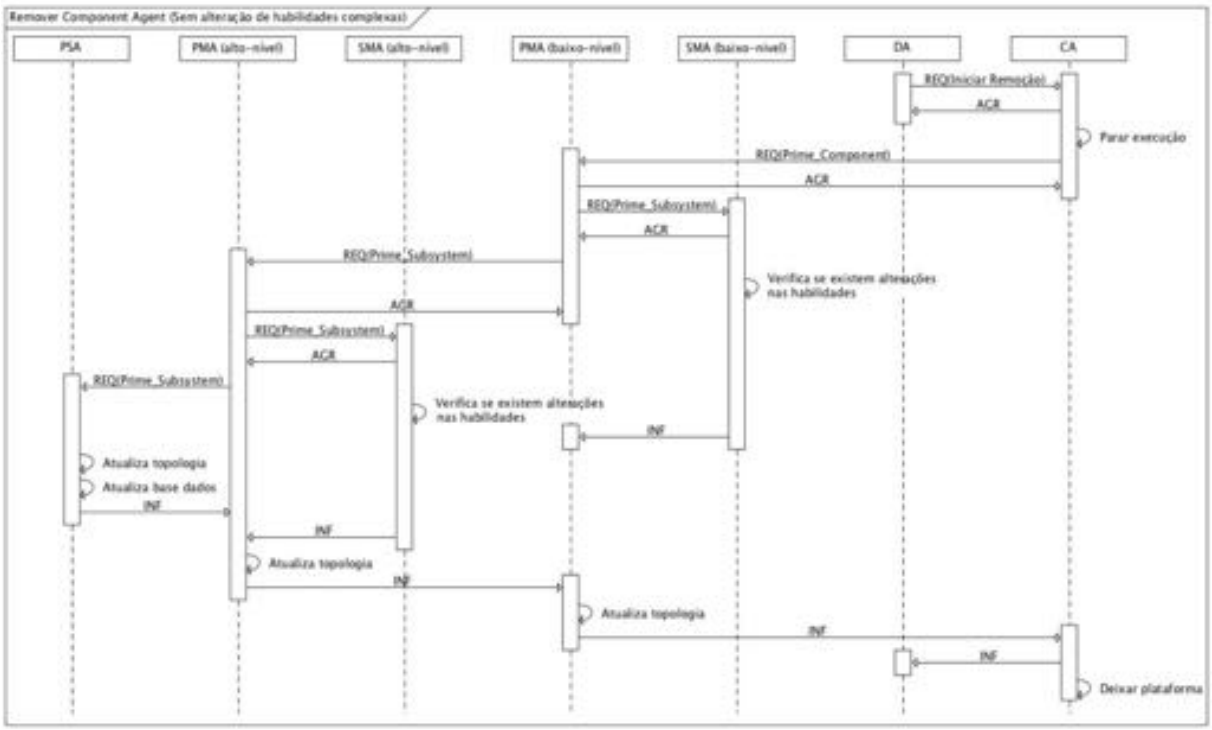

Figura 4.15: Diagrama de sequência da remoção de um CA do subsistema.

Quando o DA recebe o *Inform* do PMA este envia um *Request* ao CA a notificar que pode dar início ao processo de remoção. Assim quando um CA recebe o *Request* envia um *Agree* ao DA e um *Request* com o seu *Prime_Component* serializado ao PMA no qual está associado, pedindo para ser removido.

Quando o PMA recebe este *Request* são desencadeados nele diversos comportamentos conforme mostra a Figura 4.16.

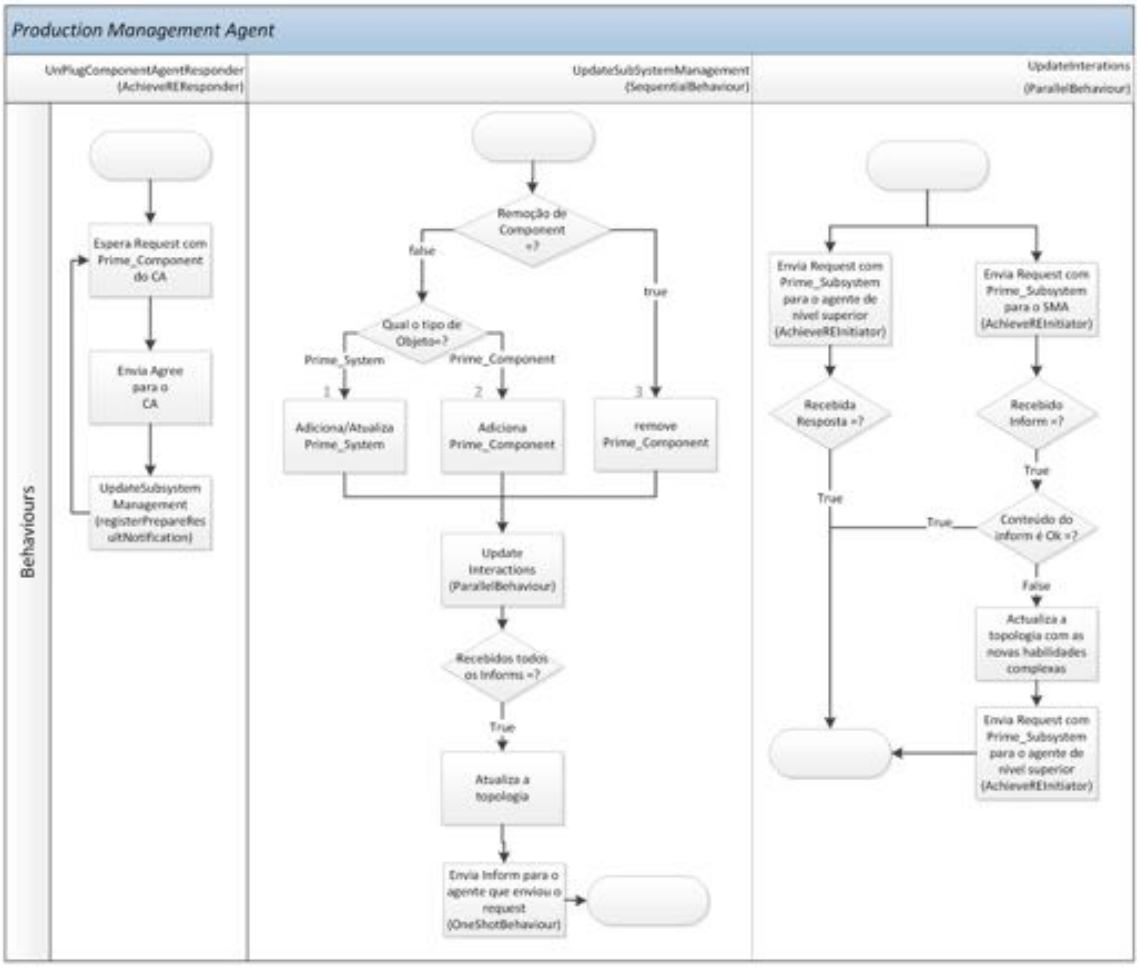

Figura 4.16: Comportamento do PMA quando recebe um *Request* para adicionar um CA.

Como se pode verificar para o CA ser removido é prescrito novamente o *SequencialBehavior UpdateSubsystemManagement*, no entanto, ao contrário dos casos anteriores desta vez, é passado um *Prime_Component*, então significa que se trata de uma remoção de um CA. O PMA cria novamente um objeto igual ao seu *Prime_Subsystem* vai à *Collection* de *Prime_SubSystem_OR_Components* onde estão guardados todos os *Prime_Subsystems* e *Prime_components* a ele associados e remove o *Prime_Component* presente nessa *Collection* com o mesmo *id* do *Prime_Component* recebido (ponto 3 na Figura 4.16).

Depois de removido envia paralelamente o objeto *Prime_Subsystem* criado para o SMA e para o agente de mais alto nível, sendo que este agente e os restantes até ao PSA assumem o comportamento descrito no subcapítulo 4.2.6.

Assim que este recebe o *Inform* do agente de mais alto nível, atualiza a sua topologia com o *Prime_Subsystem* enviado. De seguida envia um *Inform* para o CA, a fim de notificar que o sistema foi atualizado com sucesso. Por sua vez, o CA notifica também o DA com um *Inform*. Uma vez que, o foi garantido que o sistema está atualizado, o CA deixa a plataforma.

Neste caso também são desencadeados comportamentos no SMA, mas tal como referido serão detalhados no subcapítulo 4.2.12.

4.2.11 Remoção de componentes com recurso à subscrição do AMS

Na indústria é recorrente existirem fatores externos que influenciam os sistemas de produção, estes fatores são na sua grande maioria difíceis de prever, tais como falhas de energia.

Para precaver acontecimentos deste tipo foi implementado no PMA a subscrição do AMS.

O AMS é um agente nativo da plataforma JADE e tem como principal objetivo o supervisionamento e gestão de todas as operações na plataforma. Assim foi implementado no PMA um *Behavior* preferido aquando da sua inicialização denominado, *AMSSubscriber*, para isso recorreu-se aos conceitos da ontologia *JADE-Instropection* implementados pela classe *IntrospectionOntology*.

Através da implementação deste *Behaviour* foi possível fornecer ao PMA o método *installHandlers(Map handlersTable)*, este método permite associar *handlers* a eventos, e

assim o agente que subscrever este método recebe notificações sempre que existem alterações na plataforma.

O objetivo deste *Behaviour* é detetar a remoção involuntária de componentes, assim dos eventos disponibilizados apenas foi implementado o *dead-agent*. Este evento permite ao PMA receber uma notificação no *handle* sempre que um agente desaparece da plataforma.

Quando um PMA é notificado do desaparecimento de um agente este acede no seu *Prime_Subsystem* à *Collection* que contém todos os *Prime_SubSystem_OR_Components* associados ao seu subsistema, e verifica se existem algum que tenha o *id* igual ao *LocalName* do *DeadAgent* recebido no *handle*.

No caso de o PMA verificar que existe, significa que um CA anteriormente associado ao seu subsistema desapareceu da plataforma. Então dá-se início a um processo de atualização de todo o sistema, tal como esquematizado na Figura 4.17.

Depois de detetar que o CA que desapareceu fazia parte do seu subsistema, o PMA cria um objeto igual ao seu *Prime_Subsystem* e remove o *Prime_Component* detetado. Serializa o *Prime_Subsystem* e envia-o paralelamente para o SMA e para o agente de mais alto nível. Para isso à semelhança do que foi feito anteriormente recorreu-se à combinação de um *SequentialBehaviour* com um *ParallelBehaviour*.

O comportamento desencadeado nos agentes de mais alto nível até ao PSA é descrito no subcapítulo 4.2.6, no caso do SMA é descrito no subcapítulo 4.2.12.

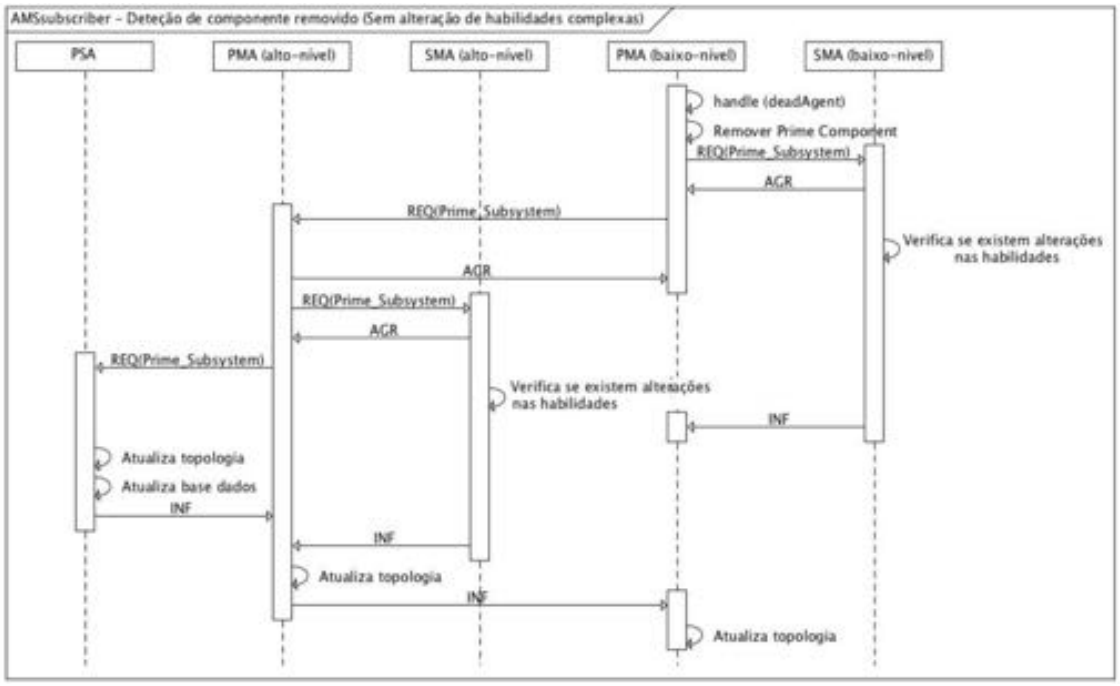

Figura 4.17: Diagrama de sequência após desaparecimento de um CA.

4.2.12 Comportamento do SMA e do sistema quando existem alterações nas CSK de um subsistema

Sempre que existem alterações de topologia no sistema, depois de cada um dos PMAs atualizarem o seu subsistema estes serializam o *Prime_Subsystem* e enviam-no para o SMA responsável por gerir as CSK disponibilizadas pelo seu subsistema. Este comportamento foi descrito nos subcapítulos 4.2.6, 4.2.9, 4.2.10 e 4.2.11.

No caso de existir uma nova regra o PSA coloca todas as regras num *ArrayList* de *CompositeSkills* serializa, e difunde por todo o sistema com o intuito de todos os SMAs terem acesso a essa informação. Assim o SMA recebe um *Request* com todas as regras necessárias para gerar CSK, tal como descrito no subcapítulo 4.2.7.

Nos subcapítulos referidos foi considerado que o SMA não detectava alterações nas CSK disponibilizadas pelo subsistema, neste caso era enviado um *Inform* para o PMA com o conteúdo "Ok".

O presente subcapítulo pretende clarificar a implementação dos comportamentos desencadeados no SMA, quando recebe um *Request* do PMA. Estes comportamentos estão esquematizados na Figura 4.18.

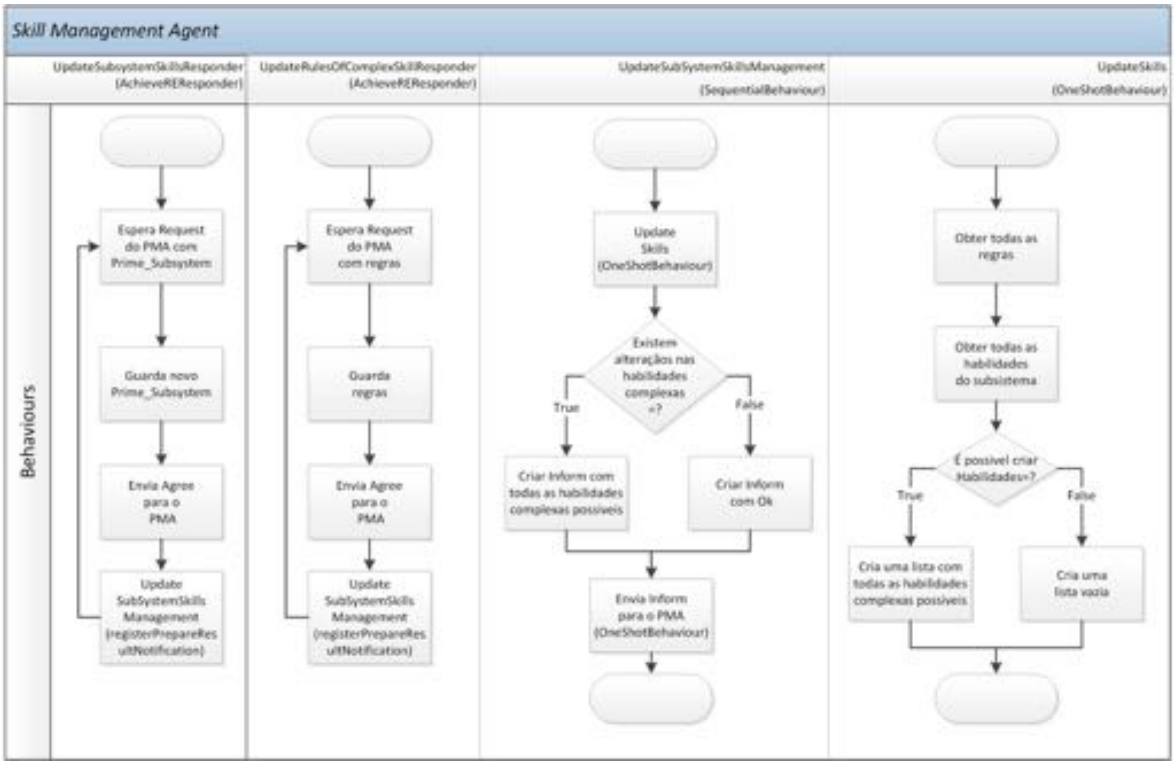

Figura 4.18: Comportamento do SMA após desaparecimento de um CA.

O SMA está constantemente à espera de receber do PMA um *Request* com todas as regras ou com o *Prime_Subsystem*, para isso quando este é inicializado são criados dois *Behaviours*, *AchieveREResponder*, como se pode verificar na Figura 4.18.

Em ambos os *Behaviours* implementados é enviado um *Agree* para o PMA e preferido um *registerPrepareResultNotification* com um *SequentialBehaviour*, *UpdateSubSystemSkillsManagement* (Figura 4.18), que processará toda a informação e notificará o PMA do resultado.

Quando o SMA recebe as regras este deserializa o conteúdo da mensagem e guarda-as para que possa utilizadas de futuro, no caso de receber o *Prime_Subsystem* este guarda-o

numa variável auxiliar para que o estado anterior do subsistema não seja perdido.

O *Behaviour UpdateSubsystemSkillsManagement* é um *SequentialBehavior* que implementa dois *OneShotBehaviour* em que o primeiro processa a informação recebida e o segundo envia um *Inform* para o PMA com os resultados. A implementação do *SequentialBehaviour* garante que o *Inform* só é enviado depois de processada toda a informação.

Tal como já foi referido anteriormente, as regras são representadas por objetos do tipo *CompositeSkill* e como se pode verificar no diagrama de classes da Figura 4.1, uma *CompositeSkill* tem associado uma *Collection* de *Skills* e de *Parameters*, onde estarão as habilidades necessária para gerar a CSK e os parâmetros exigidos, respetivamente.

Assim no *UpdateSkills* (*OneShotBehaviour*), são obtidas todas as habilidades disponibilizadas pelo sistema, são estas as SSK do subsistema e as CSK fornecidas pelos subsistemas associados.

Depois de obtidas todas as habilidades, as regras são analisadas de forma individual. Para melhorar o desempenho é verificado primeiro se as habilidades exigidas na regra são disponibilizadas pelo subsistema. Para isso o *id* da habilidade exigida tem de ser igual ao *id* de uma habilidade do subsistema, caso o subsistema disponibilize todas as habilidades exigidas numa regra, esta é colocada num *ArrayList*.

Esta primeira seleção de regras permite que não sejam consumidos recursos a analisar parâmetros exigidos, caso o sistema não disponibilize habilidades exigidas.

Uma regra pode ou não exigir parâmetros específicos, um parâmetro é representado pelo objeto *Parameter* e como se pode verificar no diagrama de classes da Figura 4.1, um parâmetro tem entre outros *id*, *Upper_Bound*, *Lower_Bound* e *Value*.

Para guardar todas as CSK a disponibilizar pelo subsistema é criado um *ArrayList*

de *CompositeSkill*, no caso de uma regra não ter parâmetros associados, se for cumprido o requisito referente ao *id* esta é adicionada ao *ArrayList* e considerada uma CSK.

No caso da regra exigir parâmetros é necessário verificar se as habilidades do subsistema satisfazem estes requisitos, para isso são comparados os parâmetros da regra com os das habilidades do subsistema.

Para que este requisito seja cumprido, é necessário que os *ids* dos parâmetros sejam iguais e que os valores de *Value*, *Upper_Bound* e/ou *Lower_Bound* cumpram os requisitos apresentados na Tabela 4.2.

Tabela 4.2: Requisitos para a validação de parâmetros

Habilidade / Regra	Value (V_h)	Lower_Bound (L_h)	Upper_Bound (U_h)	Lower_Bound (L_h) e Upper_Bound (U_h)
Value (V_r)	$V_h = V_r$	$L_h < V_r$	$U_h > V_r$	$L_h < V_r < U_h$
Lower_Bound (L_r)	$V_h > L_r$	-	$U_h > L_r$	$U_h > L_r$
Upper_Bound (U_r)	$V_h < U_r$	$L_h < U_r$	-	$L_h < U_r$
Lower_Bound (L_r) e Upper_Bound (U_r)	$L_r < V_h < U_r$	$L_h < U_r$	$U_h > L_r$	$L_h < U_r \wedge U_h > L_r$

Caso os sistema consiga validar todos parâmetros associados a uma regra, esta regra é adicionada ao *ArrayList* das CSK a disponibilizar pelo sistema, pois todos os requisitos foram cumpridos.

Validadas todas as CSK este algoritmo é repetido, mas agora em vez de utilizadas as habilidades do subsistema são utilizadas as CSK geradas, para que seja possível gerar habilidades de mais alto nível, aumentando assim ainda mais a granularidade do sistema.

Depois de geradas todas as CSK possíveis, é necessário verificar se estas implicaram alterações no subsistema. Para isso é utilizado o *Prime_Subsystem* referente ao estado o subsistema depois da ultima atualização e obtidas todas as CSK presentes na *Collection*

de *Skill* (Figura 4.1).

A duas listas de CSK são comparadas, caso não existam diferenças é criado um *Inform* com o conteúdo *Ok*, caso o contrário é colocado no conteúdo o *ArrayList* das CSK serializado, e guardado o *Prime_Subsystem* com as novas CSK sendo este a referência para o estado atual do sistema. Por fim, o *Inform* é enviado para o PMA num *OneShotBehaviour* (Figura 4.18).

Sempre que existem alterações a reportar ao PMA, vai ser desencadeada a atualização das camadas acima do subsistema que sofreu alterações. Este processo repete-se até todos os SMAs enviarem *Informs* com o conteúdo *"Ok"*, perante isto o sistema estabiliza e só voltam a ser desencadeados este comportamento quando o SMA voltar a receber um *Request* do PMA.

4.2.13 Informação do estado do sistema em tempo real

Sendo o PSA a entidade de mais alto nível, é nele que se encontra o estado atual de todo o sistema. Assim para que seja possível fornecer esta informação ao utilizador são trocadas mensagens com o HMIA que, por sua vez, comunicará com uma aplicação externa (HMI), fornecendo toda a informação obtida.

O PSA pode fornecer ao HMIA o objeto *Prime_System* que contém toda a informação do sistema, uma lista de produtos previamente introduzidas na base de dados, ou os requisitos necessário para produzir determinado produto.

Na Figura 4.19 estão esquematizados os diferentes comportamentos associados ao PSA quando solicitado pelo HMIA.

Para que o HMIA possa fornecer ao HMI informações sobre o estado atual do sistema, tais com quais os recursos (CAs) disponíveis, o HMIA envia um *Request* para o PSA a pedir

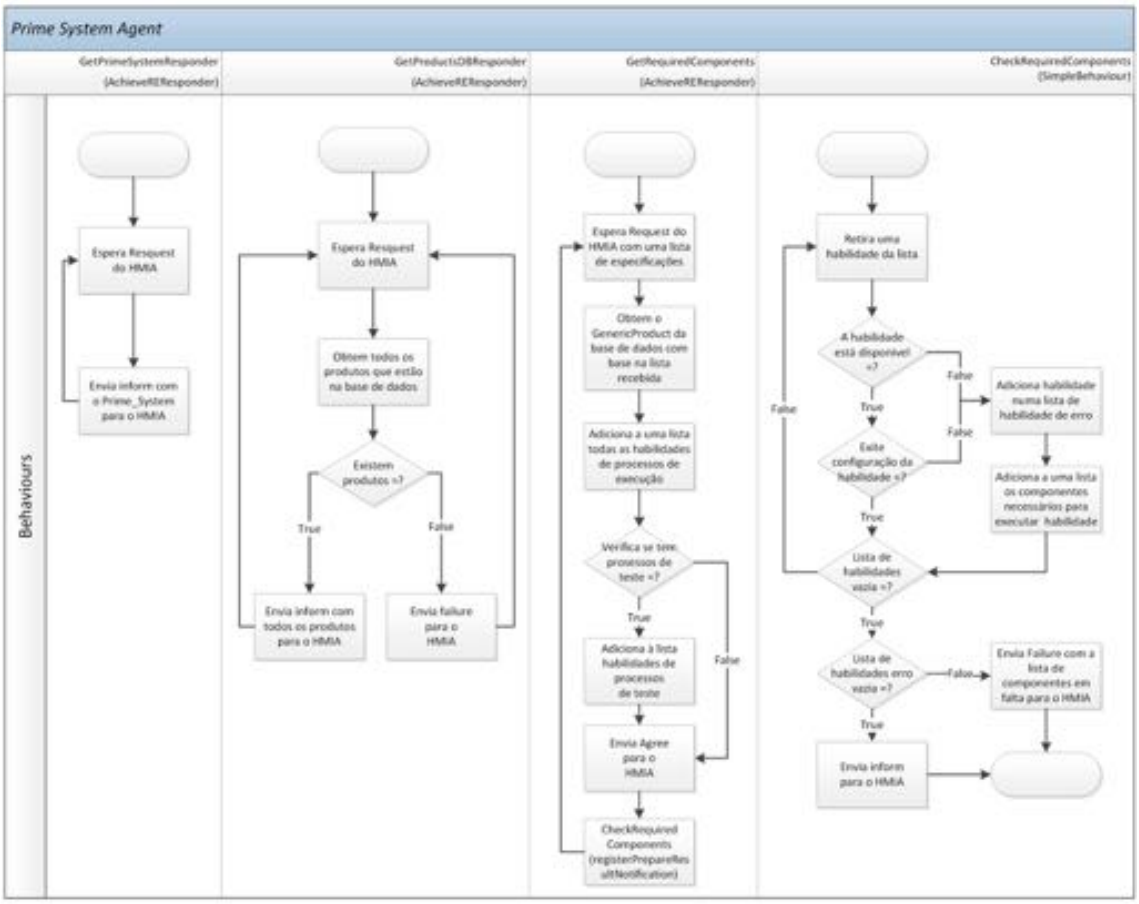

Figura 4.19: Comportamentos do PSA quando solicitada informação pelo HMIA.

a topologia do sistema. Para isso o PSA envia um *Inform* com o *Prime_System* serializado.

Outra informação que o HMI pode requerer, é os produtos que estão guardados na base de dados. Neste caso o HMIA envia um *Request* ao PSA a pedir todos os produtos e o PSA faz uma *query* à base de dados e obtém um *ArrayList* com todos os *GenericProduct* guardados na tabela *Product*, como se pode verificar na base de dados da Figura 4.2.

Depois de obtido um *ArrayList* com todos os *GenericProduct* é serializado e enviado num *Inform* para o HMIA.

Depois de saber todos os produtos disponíveis, antes de reconfigurar o sistema o utilizador pode querer saber se é possível produzir determinado produto com base na topologia

atual do sistema. Para isso, o HMIA envia um *Request* ao PSA com um *ArrayList* serializado. Este *ArrayList* tem as especificações de (re)configurar o sistema para produzir determinado produto.

Este *ArrayList* contém *Parameters* com o *id* do produto, as habilidades de testes a executar nesse produto e a lista de recursos desejada. Assim, o utilizador ao selecionar quais os recursos que pretende utilizar, são adicionados os *ids* de cada um deles à lista de especificações, podendo sempre optar por selecionar todos os componentes associados ao sistema.

Quando o PSA recebe o *Request* deserializa o conteúdo, obtendo o *ArrayList* de *Parameter*. Com o *id* é feita uma *query* à base de dados afim de obter o *GenericProduct* correspondente ao produto pretendido.

O *GenericProduct* contém uma *Collection* de *ExecutionProcess* que por sua vez contém uma *List* de *Skill* (Figura 4.1), a *List* contém todas as habilidades necessárias para a execução do produto, sendo adicionadas a um *ArrayList* de *Skill*.

Por defeito todos os *GenericProduct* guardados na base de dados, têm na *List* de *Skill* presente na *Collection* de *TestProcess*, com todas as habilidades de teste possíveis associadas ao produto, como na lista recebida estão especificadas quais as que o utilizador pretende, estas são adicionas ao *ArrayList* onde constam todas as habilidades de execução.

Posto isto, é enviado um *Agree* para o HMIA a notificar que o pedido esta a ser processado, e é criado no PSA um *SimpleBehaviour* num *registerPrepareResultNotification* (Figura 4.19), para que seja processada a informação recebida e posteriormente enviada uma resposta com o resultado.

O *SimpleBehaviour*, *CheckRequiredComponents* (Figura 4.19), consiste em colocar todas as habilidades obtidas anteriormente numa *Queue* de *Skill*, isto permite remover ha-

bilidades da lista à medida que são analisadas individualmente.

O PSA vai ao *Prime_System* e verifica quais as habilidades disponibilizadas no sistema com base nos *ids* dos componentes recebidos e coloca-as num *ArrayList*, a cada uma das habilidades da *Queue* é verificado se existe alguma igual no *ArrayList*, até que a *Queue* fique vazia.

Caso alguma das habilidades descritas na *Queue* não coincida com as disponibilizadas, é feita uma *query* à base de dados, para obter o histórico de todos os CAs até então associados ao sistema. Tendo todos os *Prime_Components* guardados na tabela *Prime-Component* (Figura 4.2), é verificado quais os componentes necessários para executar a habilidade em causa, e adicionados num *ArrayList* de *Prime_Components*.

Quando a *Queue* estiver vazia, caso tenham sido encontradas soluções para as habilidades indisponíveis é serializado o *ArrayList* de *Prime_Components* e enviado para o HMIA num *Inform*, caso contrário é enviado um *Failure*.

No caso do sistema reunir todas as especificações exigidas para se (re)configurar, é enviado um *Inform* sem conteúdo.

4.2.14 Configuração do Sistema

Para produzir determinado produto é necessário previamente (re)configurar todo o sistema de produção. Este comportamento é desencadeado pelo utilizador que seleciona um determinado produto e algumas especificações e envia para o HMIA através do HMI. Tal como se pode verificar no diagrama de sequência esquematizado na Figura 4.20.

Foram implementados dois tipos de (re)configuração, a manual e a automática. A manual exige que o utilizador selecione dos recursos disponibilizados quais pretende reconfigurar, a automática surge no caso do utilizador não ter qualquer tipo de preferência,

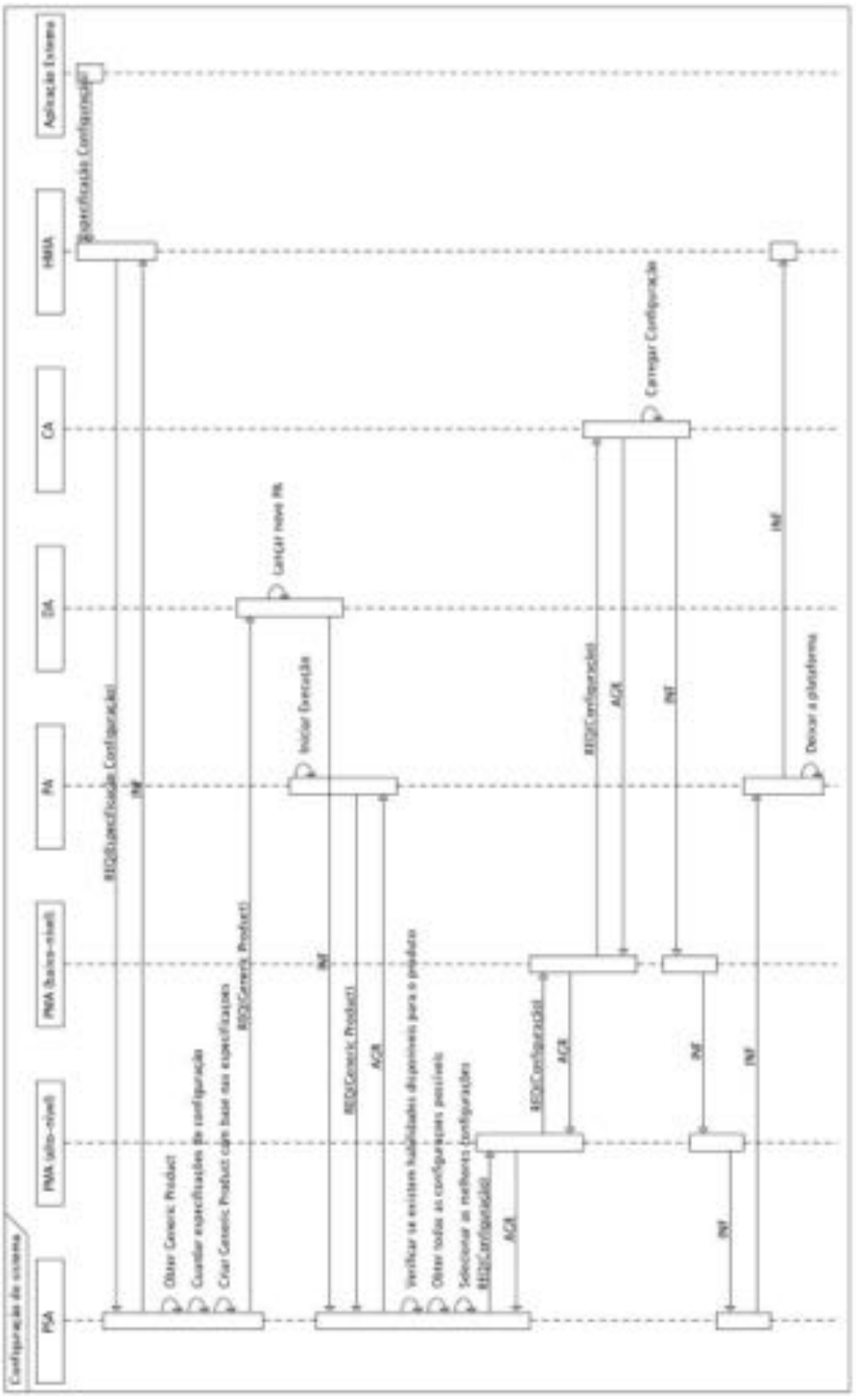

Figura 4.20: Diagrama de sequência da (re)configuração do sistema.

selecionando todos os recursos. Neste caso a escolha dos componentes a (re)configurar é feita com base numa prioridade associada à configuração.

A configuração é representada por um objeto do tipo *Configuration* (Figura 4.1), e está sempre associada a uma habilidade. Como se pode verificar no diagrama de classes da Figura 4.1, a configuração tem entre outras, uma *List* de *Parameters* e uma *Collection* de *Prime_Subsystem_OR_Components*. A *List* de *Parameters* permite associar parâmetros como o parâmetro de configuração e a de prioridade, a *Collection* de *Prime_Subsystem_OR_Components* permite associar subsistemas ou componentes para a qual se destina a configuração.

A noção de prioridade tornou-se imperativa, pois uma habilidade pode ser executada por diferentes componentes e esses componentes podem requerer configurações diferentes. Assim uma configuração é escolhida quanto maior for a sua prioridade.

O HMIA disponibiliza serviços que permite receber de uma aplicação externa (HMI) especificações de configuração. Por sua vez, envia um *Request* com essas especificações serializadas ao PSA. O PSA está constantemente á espera do *Request*, como se pode verificar no *Behaviour AchieveREResponder*, esquematizado na Figura 4.21.

Antes de se dar início ao processo de (re)configurações é necessário lançar um PA. Este PA abstrai o produto e é lançado com todas as informações que o descrevem.

À semelhança do que aconteceu no subcapítulo 4.2.13, recebe um *Request* com especificações para (re)configurar o sistema. Ao deserializar o conteúdo a mensagem obtém um *ArrayList* de *Parameters*, onde está especificado o *id* do produto, habilidades de teste e componentes desejados.

Como demostrado no *Behaviour PlugNewProductResponder* (Figura 4.21) o PSA guarda as especificações recebidas, e envia um *Agree* para o HMIA, a fim de notificar

que o pedido foi recebido e irá ser processado, criando um *resisterPrepareResultNotification* com um *SequentialBehaviour* onde posteriormente é enviando uma resposta com o resultado.

No *SequentialBehaviour PlugNewProductManagemt*, Figura 4.21, pretende-se obter um *GenericProduct* fazendo uma *query* à base de dados com o *id* do produto recebido na lista de especificações. Manipulando o resultado da *query* obtem-se um *GenericProduct* consoante as habilidades de teste também especificadas na lista.

Tendo o *GenericProduct* à imagem das especificações exigidas, é enviado um *Request* com o *GenericProduct* serializado num *AchieveREInitiator* para o DA lançar um PA. Assim que o DA lança o PA, responde com um *Inform* terminando assim o *Behviour*. Terminado este *Behaviour* é inicializado um *OneShotBehaviour* que envia um *Inform* ao HMIA, caso ocorra algum problema que impeça o PA de ser lançado é enviado um *Failure*.

Assim que o PA é inicializado este envia um *Request* para o PSA para dar início a processo de (re)configuração do sistema.

Ao deserializar o conteúdo do *Request* o PSA obtém o *GenericProduct* que descreve o produto, obtendo as habilidades de execução e de teste, caso existam. De seguida envia um *Agree* ao PA para notificar que o pedido foi recebido e irá ser processado e cria um *resisterPrepareResultNotification* com um *SequentialBehaviour*.

Visto que o processo de seleção de configurações pode ser demorado, foi implementado um *ParallelBehavior* que cria dois *SimpleBehaviors*, uma para as habilidades de execução e outro para as de teste, tal como se pode verificar na Figura 4.21.

Com a implementação deste dois *Behaviours* em paralelo pretende-se reduzir não só o tempo de processamento, como também, a utilização de recursos, pois pretende-se fazer *querys* à base de dados a fim de obter todas as configurações possíveis para cada uma das

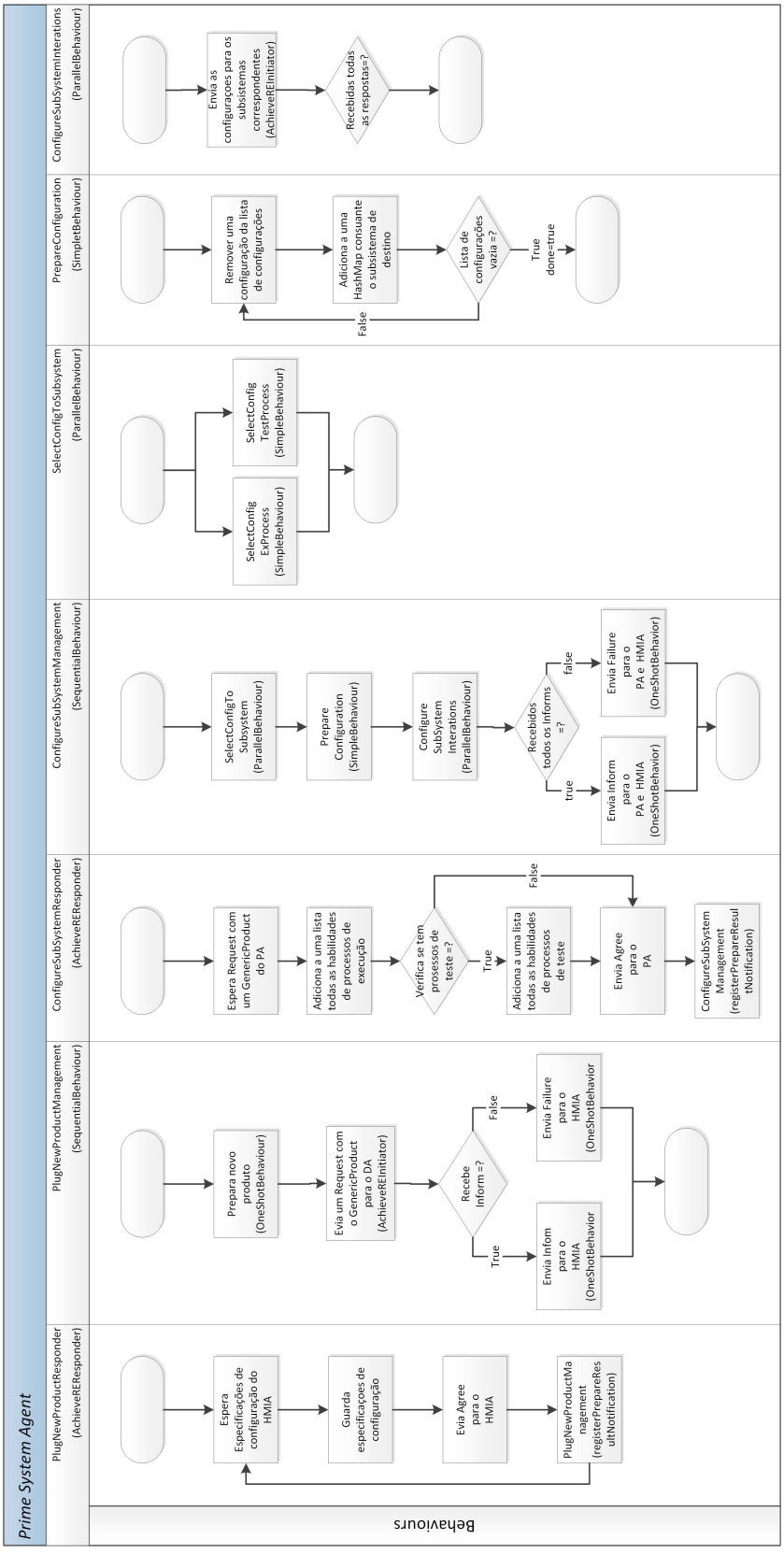

Figura 4.21: Comportamento do PSA na (re)configuração do sistema.

habilidades e selecionar qual a mais indicada. Dependendo da quantidade de informação a ser processada este processo pode ser demorado. Com esta abordagem torna-se mais rápido pois o processamento é dividido por cada *Behaviour*.

O processo de seleção de uma configuração consiste em verificar se o(s) *Prime_Component(s)* presente(s) na *Collection* de *Prime_Subsystem_OR_Components* constam na lista de especificações recebida. No final da verificação se para uma habilidade existirem duas ou mais configurações destinadas ao mesmo componente, então é selecionada a que tem maior prioridade. A habilidade que levou à obtenção da configuração é guardada na *Skill* do objeto *Configuration* (Figura 4.1).

Terminado o *ParallelBehaviour* é inicializado um *SimpleBehaviour* (*PrepareConfiguration*), Figura 4.21, que consiste em ordenar as configurações pelo subsistema à qual se destina.

No subcapítulo 4.2.6, foi referido que quando o sistema é atualizado independentemente da causa, em cada camada de PMAs, é guardado tanto nos *Prime_Components* como nas *Skills* o objeto *Prime_Subsystem*, a fim de se obter o caminho inverso até chegar ao subsistema que os disponibiliza.

Assim o PSA acede ao objeto *Skill* guardado no objeto *Configuration*, a fim de obter a *Collection* de *Prime_Subsystem_OR_Components*, onde estão guardados todos os *Prime_Subsystems* por onde a habilidade passou até chegar ao PSA.

As configurações são organizadas numa *HashMap* onde a *Key* é o nome do PMA de destino, ou seja, o *id* do primeiro *Prime_Subsystem* da *Collection*, o value um *Arraylist* de *Configuration* que são todas as configurações destinadas ao PMA.

Este método reduz o número de mensagens trocadas entre agentes, pois agrupa as configurações numa lista permitindo que sejam envias em conjunto.

Organizadas as configurações é lançado um outro *ParallelBehaviour* responsável por enviar as configurações para os subsistemas correspondentes. É iniciado um *AchieveREInitiator* que cria e envia um *Request* por cada entrada presente na *HashMap*, em que o destinatário é a *Key* e o conteúdo o *value* serializado.

Visto que o PSA envia um *Request* para os subsistemas a ele associados (Figura 4.20), são desencadeados nos PMAs os comportamentos demonstrados na Figura 4.22.

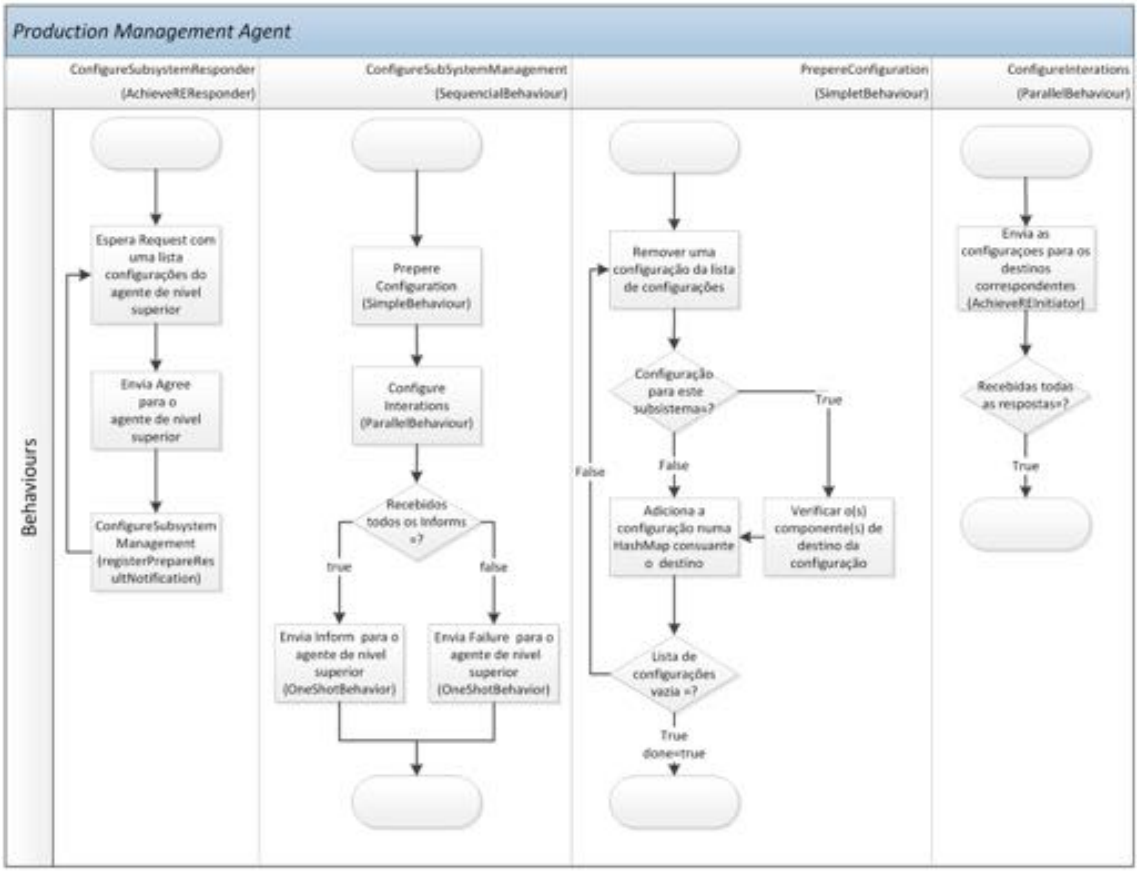

Figura 4.22: Comportamento do PMA na (re)configuração do sistema.

Assim quando o PMA deserializar o conteúdo do *Request* recebido obtém um *Arraylist* com todas as configurações destinadas ao seu subsistema. Foi implementado um *SimpleBehaviour* similar ao implementado no PSA, mas neste caso o PMA acede à *Collection*

de *Prime_Subsystem_OR_Component* presente no objeto *Skill* guardado em cada um dos objetos *Configuration* recebidos e organiza-os também por destinatário.

Para isso percorre os *ids* dos *Prime_Subsystems* guardados e no caso do *id* de um dos *Prime_Subsystems* ser igual ao seu *LocalName* então o nome do destino é o *id* do *Prime_Subsystem* seguinte. Criando também uma *HashMap* onde a *Key* é o nome do destino e o value um *Arraylist* de *Configuration* com todas as configurações destinadas ao agente com esse nome.

No caso do seu *LocalName* coincidir com o último *Prime_Subsystem* da *Collection* significa que aquela habilidade é disponibilizada pelo subsistema por ele representado. Assim na *HashMap* a *key* são os *ids* do *Prime_Components* presentes na *Collection* de *Prime_Subsystem_OR_Component* do objeto *Configuration*, pois representa os componentes a serem (re)configurados.

Criada a *HashMap* é iniciado um *AchieveREInitiator* que cria e envia um *Request* por cada entrada presente na *HashMap*, em que o destinatário é a *Key* e o conteúdo o *value* serializado. No caso da ultima camada de PMAs os destinatários são os CAs a ser (re)configurados.

Depois dos CAs estarem (re)configurados respondem com um *Inform* para o PMA que enviou o *Request*, sendo que este processo se repete até ao PSA como se pode verificar na Figura 4.20. Caso a configuração falhe são replicados ao longo da árvore *Failures*.

Uma vez que o PSA é notificado do resultado de todo este processo, envia o resultado para o PA e para o HMIA, terminando assim o processo de (re)configuração do sistema.

Capítulo 5

Validação e Resultados

Com o intuito de validar o trabalho apresentado no capitulo 4, este foi subido a dois testes distintos. Um com base sistema real e o outro com base num sistema virtual.

Assim, no subcapítulo 5.1 é feita uma descrição do sistema real e, posteriormente, a apresentação da execução do sistema e resultados. No subcapítulo 5.2 é apresentado um sistema virtual à qual o trabalho desenvolvido foi submetido e respectivos resultados. Por fim, o subcapítulo 5.3 é referente à discussão dos resultados apresentados nos subcapítulos 5.1 e 5.2.

5.1 Sistema Real

O sistema real denominado *Modutec* é uma plataforma de montagem altamente flexível, integrada no âmbito do projeto PRIME. O sistema *Modutec* foi criado pela *FEINTOOL* [FEI15], e cedida pela *TQC Automation & Test Solutions* [TQC15], uma empresa com 20 anos de experiência em automação industrial e equipamentos de teste.

5.1.1 Descrição do sistema real

A plataforma de montagem da Figura 5.1, é composta por oito estações (módulos) independentes, tanto ao nível da estrutura mecânica como da instalação elétrica. Um

sistema de transporte está ao alcance de todos os módulos da plataforma, usado para transportar o produto para as estações de trabalho.

Cada modulo tem o seu mecanismo de controlo, a fim de implementar uma arquitetura de controlo distribuído. Cada estação de trabalho é controlada por um *PLC Beckhoff CX5010* e o sistema de transporte por um *PLC Beckhoff CX2030* [ARC$^+$15].

Figura 5.1: *Modutec* plataforma de montagem altamente flexível da *FEINTOOL*, retirado de [ARC$^+$15].

Das oito estações disponíveis estão a ser utilizadas seis, duas com robôs, duas como espaço de trabalho, uma com um suporte de ferramentas e a última como estação de testes.

São utilizados dois robôs *Kuka KR5 Sixx R650* de seis eixos, em estações independentes, mas ambos com aceso ao suporte de ferramentas, que pode conter no máximo seis ferramentas, presente numa estação entre os dois robôs. As ferramentas são pinças e

podem ser utilizadas pelos robôs de forma assíncrona, sendo duas delas pneumáticas e as restantes quatro mecânicas. A estação de teste contem um teste mecânico de força exercido sobre o produto e um teste de visão onde é verificada a integralidade do produto [ARC+15].

5.1.2 Descrição do Produto

O produto é uma dobradiça de retenção de dez partes como demostrado na Figura 5.2, utilizada no interior da cabine de um camião.

Figura 5.2: Partes constituintes da dobradiça de retenção.

A dobradiça é composta por duas folhas separadas que são ligadas por um freio de metal; três esferas de metal, que são colocadas nas ranhuras cilíndricas adjacentes no centro da dobradiça; três molas que são colocadas nas mesmas ranhuras depois de colocada uma esfera nessa mesma ranhura; e um retentor, que é utilizado para fechar a dobradiça. Nem todas as partes da dobradiça têm de ser utilizadas na montagem. Consequentemente, são derivadas quatro variantes de produto, cada uma com força de retenção diferente.

A força de retenção é atribuída consoante o número de molas e esferas colocadas na dobradiça, sendo que quanto maior o número de molas e esferas, maior a força de retenção exercida.

Na Figura 5.3 é possível visualizar de diferentes perspetivas a dobradiça já montada.

Figura 5.3: Diferentes perspetivas a dobradiça já montada.

5.1.3 Execução do Sistema

Para que seja possível juntar todas as partes da dobradiça de modo a obter o resultado apresentado na Figura 5.3, é necessário modelar o sistema de modo a que este disponibilize habilidades capazes de montar a dobradiça.

Na Figura 5.4 é possível visualizar a forma com o sistema apresentado no subcapítulo 5.1.1 foi abordado.

Então é utilizado como entidade de mais alto nível e de caráter obrigatório um PSA, três PMAs, sendo que como era de esperar associado a eles estão três SMAs.

Ao PSA é associado um PMA, por sua vez a este é associado o PMA_1, o PMA_2 e o sistema de transporte. Assim o subsistema representado pelo PMA detém as CSK disponibilizadas pelos PMAs a ele associados, as SSK disponibilizadas pelo sistema de transporte e disponibiliza, caso existam, as CSK geradas pelo SMA_PMA com base nas

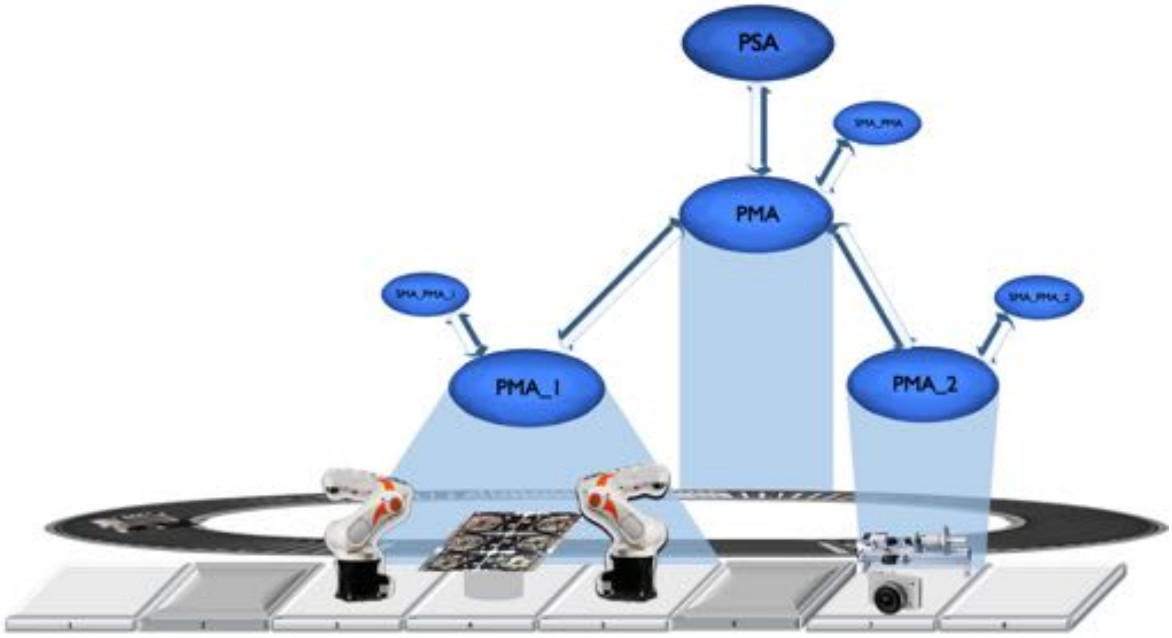

Figura 5.4: Arquitetura utilizada na execução do sistema.

regras impostas.

Ao PMA_1 estão associados os dois robôs e as ferramentas por eles utilizadas. Este subsistema detém as SSK fornecidas pelos componentes a ele associados e disponibiliza ao PMA as CSK geradas pelo SMA_PMA_1.

Ao PMA_2 estão associados os componentes de teste, camera e teste de força, onde este detém as SSK fornecidas pelos componentes a ele associados e disponibiliza ao PMA as CSK geradas pelo SMA_PMA_2.

Foram criados três subsistemas devido ao facto de, existirem três secções distintas. Como se pode verificar na Figura 5.4 o sistema de transporte tem a capacidade de transportar o produto de modo a que este fique ao alcance de todas as estações, assim este foi associado a um subsistema de mais alto nível ao qual foi associado outros subsistemas que representam as estações. Como as ferramentas são partilhadas entre os robôs estes fazem parte do mesmo subsistema, por fim como a camera e o teste de força são ambos

para verificar a integralidade da dobradiça foi criado um outro subsistema ao qual são associados.

Sendo cada um dos componentes abstraídos por um CA, quando são associados a um PMA, fornecem-lhe todas as suas caraterísticas. Esta caraterística para além do seu nome são também as SSK que estes conseguem oferecer sendo que cada SSK tem associados a ela parâmetros que descrevem os detalhe associados a cada uma delas, por exemplo uma *Tool* mecânica oferece uma SSK *Grasp* que permite pegar em objetos que compreendam os valores de *Lower_Bound* e *Upper_Bound* descritos no parâmetro *Width*.

Na Tabela 5.1 estão descritas as especificações associadas as cada um dos componentes descritos no sistema.

Para que seja possível criar CSK de modo a aumentar a granularidade do sistema e a simplificar a descrição do produto, são criadas regras com base na Tabela 5.2.

Uma vez criadas as regras o produto pode ser descrito por um conjunto de habilidades, mas para que seja possível configurar os componentes responsáveis pela execução de cada habilidade é necessário especificar as configurações para cada habilidade. Assim na Tabela 5.3 estão representadas essa especificações. Por exemplo, para configurar o *Robot 1* para fazer um *Pick and Place* com a *Tool1* é necessário enviar para o CA que o abstrai, um objeto *Configuration* (Figura 4.1) com um *Parameter* 541 (Tabela 5.3) .

Tabela 5.1: Características oferecidas pelos componentes.

Component	Type	Simple Skill	Parameter 1	L_B* 1	U_B* 1	Value 1	Parameter 2	L_B* 2	U_B* 2	Value 2	Parameter 3	L_B* 3	U_B* 3	Value 3
Robot 1	Robot	Move	From	--	--	Ws-1to10	To	--	--	Ws-11to17	--	--	--	--
		Press	Force	150	1050	--	--	--	--	--	--	--	--	--
Robot 2	Robot	Move	From	--	--	Ws-1to10	To	--	--	Ws-11to17	--	--	--	--
		Press	Force	150	1050	--	--	--	--	--	--	--	--	--
Tool 1	Mechanical gripper	Grasp	Width	7	11	--	Geom.	--	--	"Flat"	Pocket	--	--	False
		Grasp	Width	5	9	--	Geom.	--	--	"Cylinder"	Pocket	--	--	False
		Surface	Tip	--	--	"Hard"	Geom.	--	--	"Flat"	--	--	--	--
Tool 2	Mechanical gripper	Grasp	Width	12	21	--	Geom.	--	--	"Flat"	Pocket	--	--	True
		Grasp	Width	13,5	22	--	Geom.	--	--	"Cylinder"	Pocket	--	--	True
		Surface	Tip	--	--	"Hard"	Geom.	--	--	"Flat"	--	--	--	--
Tool 3	Pneumatic gripper	Suction	Vacuum	-83	0		Nozzle	1	3	--	--	--	--	--
		Release	--	--	--	--	--	--	--	--	--	--	--	--
		Surface	Tip	--	--	"Soft"	Geom.	--	--	"Point"	--	--	--	--
		Surface	Tip	--	--	"Hard"	Geom.	--	--	"Ring"	--	--	--	--
Tool 4	Mechanical gripper	Grasp	Width	11	20,5	--	Geom.	--	--	"Flat"	Pocket	--	--	True
		Grasp	Width	13,5	23	--	Geom.	--	--	"Cylinder"	Pocket	--	--	False
		Surface	Tip	--	--	"Hard"	Geom.	--	--	"Flat"	--	--	--	--
Tool 5	Pneumatic gripper	Suction	Vacuum	-85	0	--	Nozzle	1	2	--	--	--	--	--
		Release	--	--	--	--	--	--	--	--	--	--	--	--
		Surface	Tip	--	--	"Soft"	Geom.	--	--	"Point"	--	--	--	--
		Surface	Tip	--	--	"Hard"	Geom.	--	--	"Ring"	--	--	--	--
Tool 6	Mechanical gripper	Grasp	Width	9	18	--	Geom.	--	--	"Flat"	Pocket	--	--	True
		Grasp	Width	4	12,58	--	Geom.	--	--	"Cylinder"	Pocket	--	--	False
		Surface	Tip	--	--	"Hard"	Geom.	--	--	"Flat"	--	--	--	--
Camera	Camera	Photo	Aperture	0	2,8	--	--	--	--	--	--	--	--	--
Force Test	Force Test	Force	Force	0	150	--	--	--	--	--	--	--	--	--

*L_B -> Lower Bound *U_B -> Upper Bound

Tabela 5.2: Requisitos utilizados para gerar CSK.

Complex Skill	Parts	Simple Skill	Parameter 1	L_B* 1	U_B† 1	Value 1	Parameter 2	L_B* 2	U_B† 2	Value 2	Parameter 3	L_B* 3	U_B† 3	Vallue 3
Pick and Place	Double leaf	Move	From	--	--	"Ws-10"	To	--	--	"Ws-11"	--	--	--	--
		Grasp	Width	15	--	--	Geom.	--	--	"Flat"	Pocket	--	--	True/False
	Single leaf	Move	From	--	--	"Ws-9"	To	--	--	"Ws-13"	--	--	--	--
		Grasp	Width	13	--	--	Geom.	--	--	"Flat"	Pocket	--	--	True/False
		Move	From	--	--	"Ws-9"	To	--	--	"Ws-13"	--	--	--	--
		Grasp	Width	2	--	--	Geom.	--	--	"Flat"	Pocket	--	--	True/False
	Tube	Move	From	--	--	"Ws-5"	To	--	--	"Ws-12"	--	--	--	--
		Grasp	Width	14	--	--	Geom.	--	--	"Cylinder"	Pocket	--	--	True/False
	Spring 1	Move	From	--	--	"Ws-2"	To	--	--	"Ws-14"	--	--	--	--
		Grasp	Width	5	--	--	Geom.	--	--	" Cylinder"	Pocket	--	--	True/False
	Spring 2	Move	From	--	--	"Ws-3"	To	--	--	"Ws-15"	--	--	--	--
		Grasp	Width	5	--	--	Geom.	--	--	" Cylinder"	Pocket	--	--	True/False
	Spring 3	Move	From	--	--	"Ws-4"	To	--	--	"Ws-16"	--	--	--	--
		Grasp	Width	5	--	--	Geom.	--	--	" Cylinder"	Pocket	--	--	True/False
	Retainer	Move	From	--	--	"Ws-1"	To	--	--	"Ws-17"	--	--	--	--
		Grasp	Width	10	--	--	Geom.	--	--	"Flat"	Pocket	--	--	True/False
		Move	From	--	--	"Ws-1"	To	--	--	"Ws-17"	--	--	--	--
		Grasp	Width	17	--	--	Geom.	--	--	"Flat"	Pocket	--	--	True/False
Insert	Tube	Move	From	--	--	"Ws-12"	To	--	--	"Ws-12"	--	--	--	--
		Press	Force	200	--	--	--	--	--	--	--	--	--	--
		Surface	--	--	--	--	Tip	--	--	"Hard"	Geom.	--	--	"Ring"
	Retainer	Move	From	--	--	"Ws-1"	To	--	--	"Ws-17"	--	--	--	--
		Press	Force	400	--	--	--	--	--	--	--	--	--	--
		Surface	--	--	--	--	Tip	--	--	"Soft"	Geom.	--	--	"Point"
Pneumatic Pick and Place	Ball 1	Move	From	--	--	"Ws-6"	To	--	--	"Ws-14"	--	--	--	--
		Suction	Vacuum	--	-50	--	Nozzle	--	5	--	--	--	--	--
		Release	--	--	--	--	--	--	--	--	--	--	--	--
	Ball 2	Move	From	--	--	"Ws-7"	To	--	--	"Ws-15"	--	--	--	--
		Suction	Vacuum	--	-50	--	Nozzle	--	5	--	--	--	--	--
		Release	--	--	--	--	--	--	--	--	--	--	--	--
	Ball 3	Move	From	--	--	"Ws-8"	To	--	--	"Ws-16"	--	--	--	--
		Suction	Vacuum	--	-50	--	Nozzle	--	5	--	--	--	--	--
		Release	--	--	--	--	--	--	--	--	--	--	--	--
	Retainer	Move	From	--	--	"Ws-1"	To	--	--	"Ws-17"	--	--	--	--
		Suction	Vacuum	--	-60	--	Nozzle	--	6	--	--	--	--	--
		Release	--	--	--	--	--	--	--	--	--	--	--	--
Compliance	Finished	Photo	Aperture	2	--	--	--	--	--	--	--	--	--	--
Force	Finished	Force	Force	80	--	--	--	--	--	--	--	--	--	--

*L_B -> Lower_Bound †U_B -> Upper_Bound

Tabela 5.3: Configurações associadas às CSK.

Complex Skill \ Component		Robot 1 / Robot 2						Camera	Force Test
		Tool1	Tool2	Tool3	Tool4	Tool5	Tool6		
Pick and Place	Single Leaf	541	542	-	544	-	-	-	-
	Double Leaf	-	522	-	524	-	-	-	-
	Tube	-	562	-	564	-	-	-	-
	Spring 1	721	-	-	-	-	726	-	-
	Spring 2	741	-	-	-	-	746	-	-
	Spring 3	761	-	-	-	-	766	-	-
	Retainer	841	842	-	844	-	846	-	-
Insert	Tube	581	582	583	584	585	586	-	-
	Retainer	861	862	863	864	865	866	-	-
Pneumatic Pick and Place	Ball 1	-	-	603	-	605	-	-	-
	Ball 2	-	-	623	-	625	-	-	-
	Ball 3	-	-	643	-	645	-	-	-
	Retainer	-	-	843	-	845	-	-	-
Compliance		-	-	-	-	-	-	910	-
Force		-	-	-	-	-	-	-	920

Na Tabela 5.4 estão descritos as diferentes variantes de produtos para o qual os componentes foram configurados, para isso no produto são descritas as habilidades necessárias à sua montagens. O facto de ser possível gerar habilidades de mais alto nível, permite aumentar a granularidade do sistema e assim associá-las a um produto tornando a sua descrição mais simples.

Tabela 5.4: Descrição das diferentes variantes de produto.

Product	Description	Skill														
		Pick and Place							Insert		Pneumatic Pick and Place				Compliance	Force
		Single leaf	Double leaf	Tube	Spring 1	Spring 2	Spring 3	Retainer	Tube	Retainer	Ball 1	Ball 2	Ball 3	Retainer		
Product A	0 ball / 0 springs Retainer with Mechanic Pick and place	☑	☑	☑	☒	☒	☒	☑	☑	☑	☒	☒	☒	☒	Op.	Op.
Product B	1 ball / 1 springs Retainer with Mechanic Pick and place	☑	☑	☑	☒	☑	☒	☑	☑	☑	☒	☑	☒	☒	Op.	Op.
Product C	2 ball / 2 springs Retainer with Mechanic Pick and place	☑	☑	☑	☑	☒	☑	☑	☑	☑	☑	☒	☑	☒	Op.	Op.
Product D	3 ball / 3 springs Retainer with Mechanic Pick and place	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☒	Op.	Op.
Product E	0 ball / 0 springs Retainer with Pneumatic Pick and place	☑	☑	☑	☒	☒	☒	☒	☑	☑	☒	☒	☒	☑	Op.	Op.
Product F	1 ball / 1 springs Retainer with Pneumatic Pick and place	☑	☑	☑	☒	☑	☒	☒	☑	☑	☒	☑	☒	☑	Op.	Op.
Product G	2 ball / 2 springs Retainer with Pneumatic Pick and place	☑	☑	☑	☑	☒	☑	☑	☑	☑	☑	☒	☑	☑	Op.	Op.
Product H	3 ball / 3 springs Retainer with Pneumatic Pick and place	☑	☑	☑	☑	☑	☑	☒	☑	☑	☑	☑	☑	☑	Op.	Op.

5.1.4 Resultados

Depois de estabelecidas todas as especificações apresentadas no subcapítulo 5.1.3, procedeu-se à montagem das diferentes variantes do produto. Para validar o desempenho da arquitetura foram analisados diferentes casos.

De seguida será apresentado alguns desses casos quando se pretende configurar o sistema para produzir uma dobradiça do tipo G, com duas molas e duas esferas na qual o *Retainer* é colocado através de um *Pneumatic Pick and Place*.

Como é através do HMI que o utilizador toma conhecimento da informação do sistema e escolhe as especificações de configuração, é possível nas Figuras 5.5 e 5.6 verificar o estado do sistema depois de selecionado a dobradiça do tipo G e os testes de visão e de força.

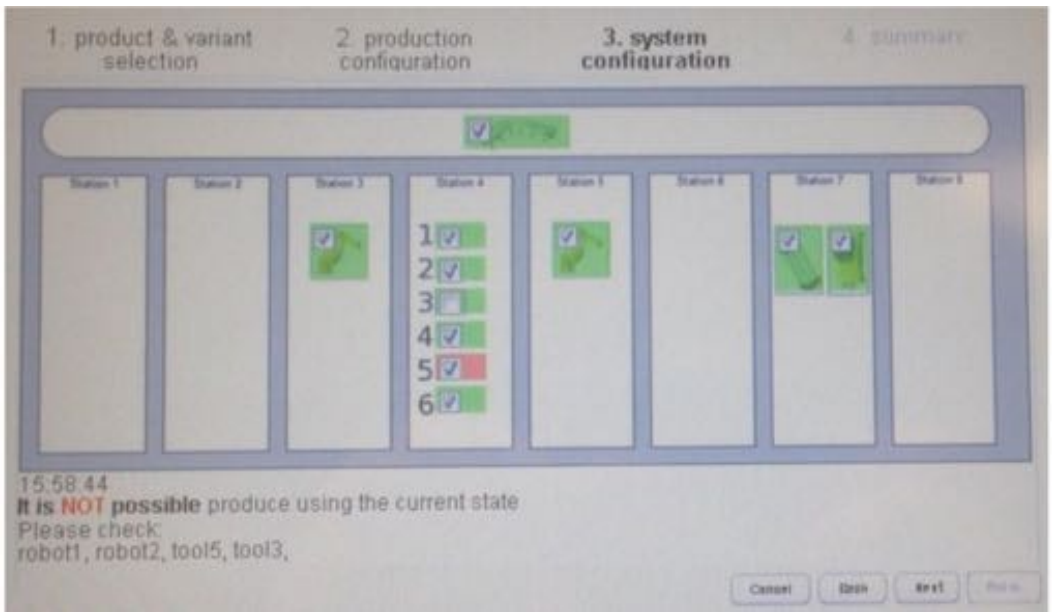

Figura 5.5: Estado do sistema no HMI depois de selecionado o produto e os testes.

Como se pode verificar na Figura 5.5 não é possível produzir o produto com base no estado atual do sistema. Esta informação é em tempo real pois o PSA está constantemente a ser requisitado, como se pode verificar na Tabela 5.4 uma dobradiça do tipo G requer um *Pneumatic Pick and Place* e como para esta CSK é necessário ter um robô e a *Tool3* ou a *Tool5*, neste caso não é possível configurar o sistema.

Ao analisar a Figura 5.5 é possível verificar que a *Tool3* está ligada (verde) e a *Tool5* desligada (vermelho), no entanto, o utilizador pode escolher dos componentes disponíveis quais os que pretende utilizar na produção do produto e como se pode verificar a *Tool3* não está selecionada. Assim, o PSA dá informação ao utilizador que para produzir uma dobradiça do tipo G para além das especificações exigidas é necessário um robô uma *Tool3* ou uma *Tool5*, isto para que o sistema seja capaz de disponibilizar um *Pneumatic Pick and Place*.

Para produzir esta dobradiça é ligada a *Tool5*, tal como se pode verificar na Figura 5.6.

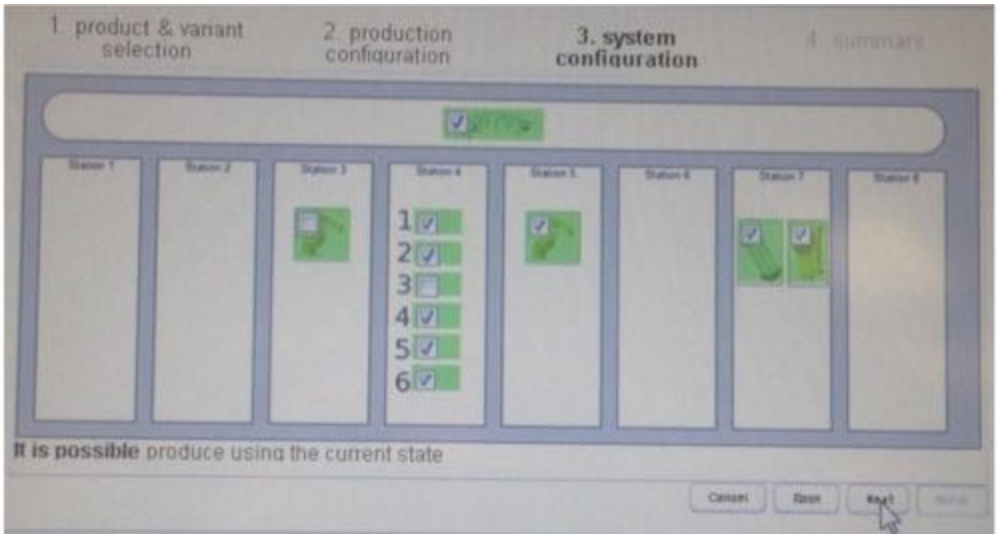

Figura 5.6: Estado do sistema no HMI depois de selecionado o produto e os testes.

Ligada a *Tool5* é possível verificar que o estado atual do sistema permite produzir a dobradiça. Neste caso é selecionado o robô 2 que se encontra na estação cinco, caso fossem selecionados os dois robôs, a primeira metade do produto era produzida no robô 1 e a outra metade no robô 2. Isto deve-se ao facto de, às configurações serem atribuídas prioridades, onde para a primeira metade do produto têm preferência as configurações associadas ao robô 1 e para a segunda metade as configurações associadas ao robô 2.

Uma vez que é possível produzir a dobradiça são enviadas as configurações para os *PLCs* correspondentes.

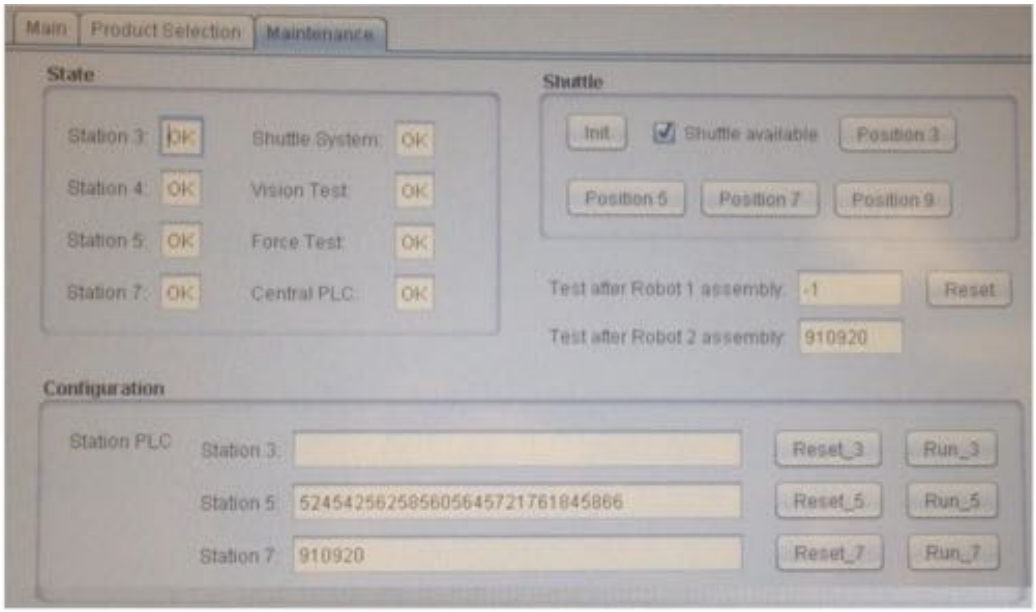

Figura 5.7: Interface gráfica que mostra as configurações enviadas para as estações correspondentes.

Na Figura 5.7 é possível visualizar uma interface gráfica desenvolvida pela Universidade de *Nottingham* que permite visualizar as configurações que são envidas para os *PLCs*, depois de estipuladas as especificações de configuração.

Pode-se verificar que para o robô 1 (*Station 3*) não são enviadas configurações, pois este não foi selecionado, para o robô 2 (*Station 5*) são enviadas todas as configurações necessárias para a montagem de uma dobradiça do tipo G. Como previamente foi selecionado o teste de visão e de força é enviado para a *Station 7* as configurações que as descrevem.

5.2 Sistema Virtual

A fim de validar e analisar a *performance* da arquitetura desenvolvida esta foi testada em ambiente virtual. Para este teste foi utilizado um computador com um processador *Intel Core i7* a *2,2 GHz, 8 Gigabytes* de memória *RAM*.

Com o intuito de avaliar a *performance* do trabalho implementado, foram criados três casos de teste distintos. Estes casos de teste diferem no número de subsistemas e no número de componentes a eles associados. Para cada caso de teste foram criados diferentes níveis de agregação, por exemplo, no caso de teste um foi criado um sistema com base na Figura 5.8, onde é criada uma camada de PMAs e a cada um deles associados seis CAs. Nos seguintes casos de teste são adicionados dois PMAs a cada PMA existente e posteriormente associados à ultima camada de PMAs seis CAs por PMA.

Assim, o número de subsistemas associados a cada caso de teste é dado pela fórmula $u_n = 2^{n+1} - 2$, tal como referido anteriormente a cada PMA da última camada são associados seis CAs, o número de componentes associados a cada caso de teste é dado por $u_n = 2^n 6$, em ambos os casos n é o número do caso de teste.

Para cada um dos casos de teste foi analisado o tempo que demora o sistema estabilizar no seu lançamento, no adicionar e remover de uma componente num PMA e o tempo que demora o sistema a reconfigurar os componentes para um determinado produto. Para cada caso o sistema foi testado vinte vezes, obtendo-se os valores médios e de desvio padrão para cada conjunto de ensaios.

5.2.1 Casos de testes e simulação do sistema

De modo a tornar esta validação mais realista, os agentes utilizados são os mesmos da arquitetura do PRIME, à exceção do HMIA onde foi criado um agente que fornece uma *GUI* para que utilizador possa escolher o sistema a ser lançado, adicionar/remover componentes e lançar um produto para configurar os CAs. O protocolo utilizado entre o

PSA e o HMIA foi mantido.

Assim na Figura 5.8 está esquematizada a arquitetura do primeiro caso de teste, onde são criados dois subsistemas e a cada um eles associados seis CAs, tal como referido anteriormente, os restantes casos de testes consistem na adição dois de subsistemas à última camada de PMAs, sendo que os CAs são sempre associados a essa ultima camada.

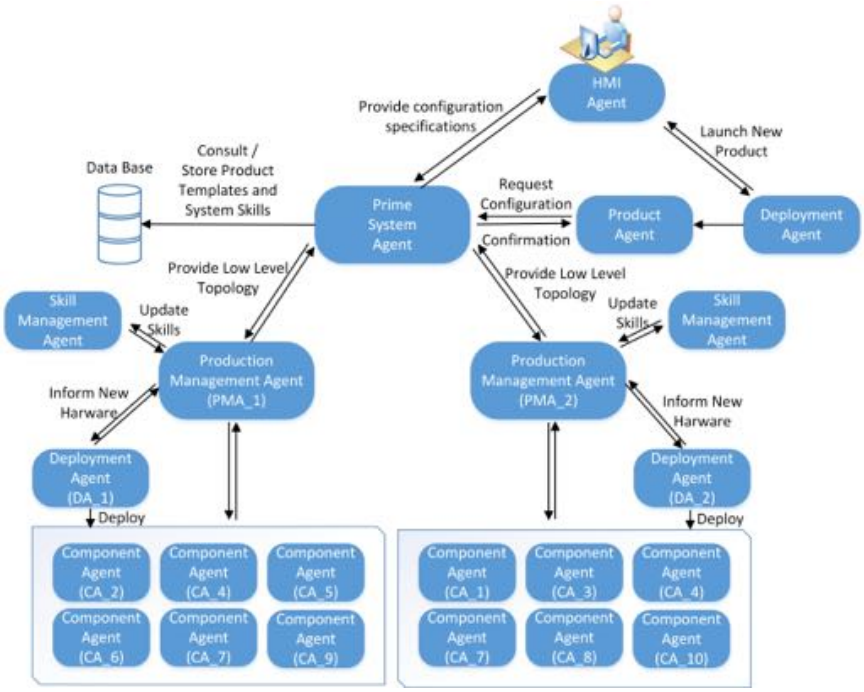

Figura 5.8: Arquitetura implementada em ambiente virtual.

Foram implementados dez CAs que disponibilizam SSK de A a J. Na Tabela 5.5 é possível visualizar as SSK disponibilizadas por cada um dos CAs bem com as regras utilizadas para gerar as CSK, onde por exemplo AB é gerada quando um subsistema detém as SSK A e B.

Tabela 5.5: Habilidades associadas a cada CA e regras para gerar CSK.

Component	Skill	A	B	C	D	E	F	G	H	I	J
CA_1	A										
CA_2	B	AB									
CA_3	C	AC	BC								
CA_4	D	AD	BD	CD							
CA_5	E	AE	BE	CE	DE						
CA_6	F	AF	BF	CF	DF	EF					
CA_7	G	AG	BG	CG	DG	EG	FG				
CA_8	H	AH	BH	CH	DH	EH	FH	GH			
CA_9	I	AI	BI	CI	DI	EI	FI	GI	HI		
CA_10	J	AJ	BJ	CJ	DJ	EJ	FJ	GJ	HJ	IJ	

A cada habilidade disponibilizada no sistema foi associada uma configuração para que seja possível caso pedido pelo PA, enviar uma configuração para o CA correspondente.

A fim de configurar os componentes e obter o tempo necessário para reconfigurar o sistema desde que é lançado um PA até este receber um *inform* a notificar que os componentes foram configurados com sucesso, foi criado um produto que é descrito por dez habilidades necessárias para a sua execução, tal como se pode verificar na Tabela 5.6.

Tabela 5.6: Descrição do produto utilizado no caso de teste.

	Skills									
Produto	BG	DF	BF	CE	AD	AE	BI	AG	CG	HJ

5.2.2 Resultados

Tal como referido anteriormente foi objeto de estudo o tempo necessário para lançar um sistema por completo. Assim é obtido o tempo que demora o sistema desde o lançamento do PSA (entidade de mais alto nível), até que o sistema estabilize depois de lançados todos os outros agentes, em cada caso de teste.

Depois de lançado cada sistema vinte vezes obtiveram-se os valores apresentados na Tabela 5.7, onde os resultados obtidos são expressos em milisegundos.

Tabela 5.7: Tempos de lançamento do sistema em cada caso de teste.

Caso de teste	1	2	3
Média	14326,9	41197,4	118589,25
Desvio padrão	134,47	348,94	1269,99
Pior resultado	14648	41780	119995
Melhor resultado	14132	40445	114257

A fim de facilitar a análise dos resultado apresentados na tabela 5.7, a Figura 5.9 apresenta os valores de média obtidos.

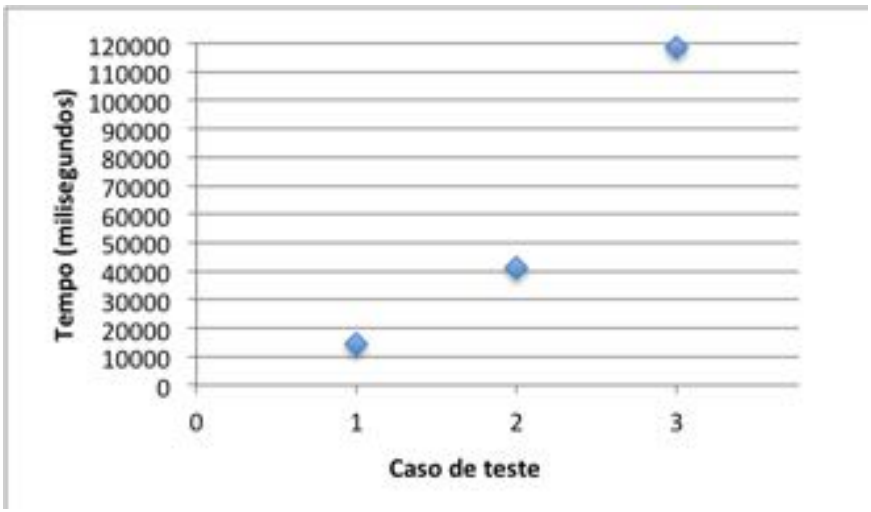

Figura 5.9: Tempos de lançamento do sistema em cada caso de teste.

À medida que se aumenta o número de PMAs e CAs no sistema o tempo associado ao seu lançamento aumenta, isto deve-se ao facto de, com o aumento do número de CAs no sistema este disponibiliza um maior número de habilidades, resultantes da combinação das habilidades disponibilizadas por cada componente. É possível verificar que para o maior subsistema com catorze PMAs e quarenta e oito CAs o sistema necessita no pior caso 119995 milisegundos, que corresponde a cerca de dois minutos.

Uma vez que o sistema se encontra estável, é objeto de estudo o tempo associado ao adicionar e remover de um componente num subsistema. Na Tabela 5.8 é possível observar os valores médios em milisegundos depois de adicionar um CA e na Figura 5.10 o gráfico com os valores de média e desvio padrão a fim de facilitar a análise dos resultados.

Tabela 5.8: Tempos associados ao adicionar de um CA no sistema, em cada caso de teste.

Caso de teste	1	2	3
Média	660,1	671,4	762,7
Desvio padrão	30,61	17,85	17,66
Pior resultado	761	710	815
Melhor resultado	615	641	739

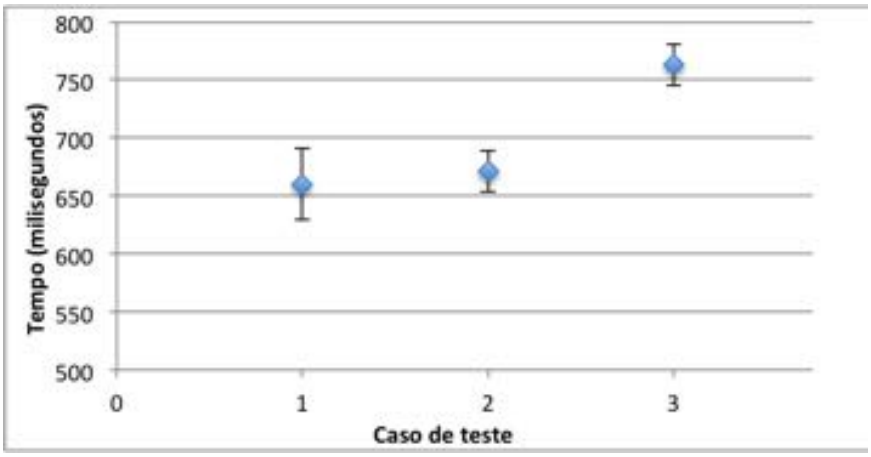

Figura 5.10: Tempos associados ao adicionar de um CA no sistema, em cada caso de teste.

Depois de adicionado e de o sistema estabilizar, este mesmo CA é removido e os resultados associados a este processo são descritos na Tabela 5.9 e à semelhança do caso anterior na Figura 5.11 está representado um gráfico com os valores de média e desvio padrão.

Tabela 5.9: Tempos associados ao remover de um CA no sistema, em cada caso de teste.

Caso de teste	1	2	3
Média	730,8	734	836,95
Desvio padrão	23,80	14,10	17,85
Pior resultado	787	764	879
Melhor resultado	684	716	813

Ao analisar os resultados obtidos anteriormente é possível verificar que independentemente do número de PMAs associados o tempo que demora o sistema a estabilizar depois de adicionado ou removido um CA é idêntico. Isto deve-se ao facto de, o CA ser adicionado a um PMA na qual estão associados seis CA e apesar de o sistema disponibilizar mais habilidades devido ao número total de CAs associados, isto não interfere no seu desempenho, pois estão divididos em vários subsistemas.

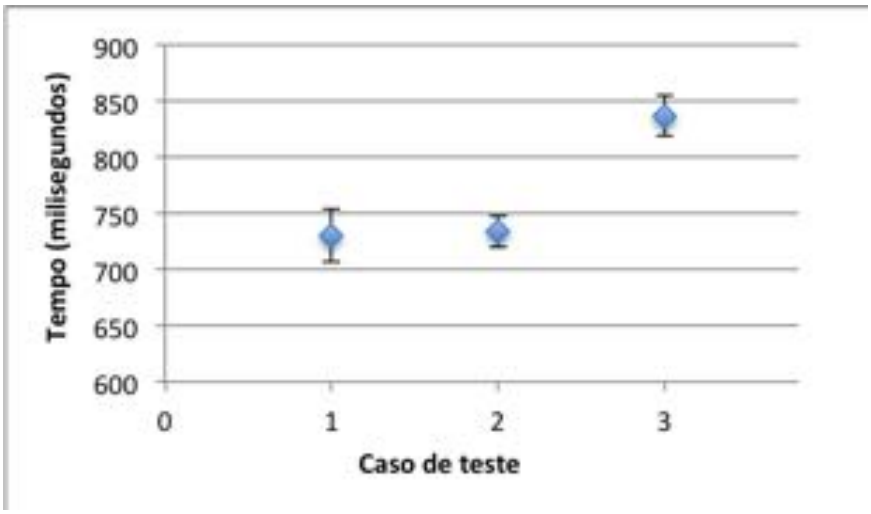

Figura 5.11: Tempos associados ao remover de um CA no sistema, em cada caso de teste.

Também se pode verificar que quando comparado com a adição de um CA, o tempo associado à sua remoção é superior isto deve-se ao facto de, quando este CA é adicionado o número de habilidades disponibilizadas pelo PMA é inferior. Quando este CA é removido este proporciona um número maior de habilidades, justificando assim o acréscimo no tempo, pois o número de habilidades a ser processadas é superior.

Por fim, é estudado o tempo associado à configuração dos componentes do sistema. Assim é lançado um PA com a descrição apresentada na tabela 5.6 e analisados dois resultados.

No primeiro caso, é analisado o tempo consumido pelo PSA para processar e preparar as configurações a enviar para os PMAs e posteriormente para os CAs correspondentes. A Tabela 5.10 descreve o tempo em cada caso de teste, que é necessário para o PSA preparar as configurações desde que é lançado o PA.

Tabela 5.10: Tempo consumido pelo PSA na configuração do sistema, em cada caso de teste.

Caso de teste	1	2	3
Média	522,1	507,15	499,2
Desvio padrão	14,57	15,19	16,85
Pior resultado	550	533	524
Melhor resultado	491	481	470

Para facilitar a análise dos resultados obtidos são apresentados os valores médios e de desvio padrão associados a cada caso de teste, na Figura 5.12.

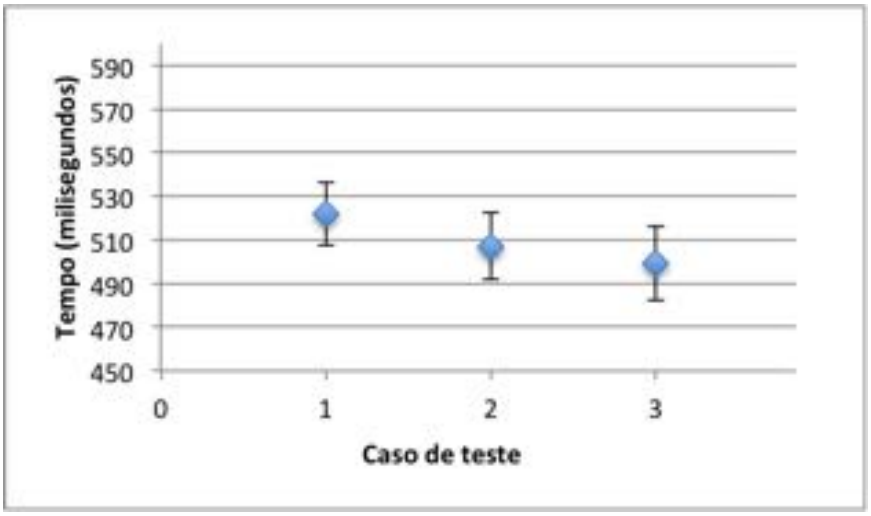

Figura 5.12: Tempo consumido pelo PSA na configuração do sistema, em cada caso de teste.

Neste caso é notório a semelhança de resultados obtidos nos três casos de teste, pois visto que o produto é sempre o mesmo o processamento exigido ao PSA em cada um dos casos é idêntico.

No segundo caso, é analisado o tempo necessário para configurar todos os CAs desde que o PA é lançado. Para os diferentes casos o sistema foi planeado de modo a que o produto fosse divido de forma equitativa pelos CAs presentes no sistema.

A tabela 5.11 e na figura 5.13 estão apresentados os resultados obtidos nesta etapa.

Caso de teste	1	2	3
Média	533,45	527,3	511,35
Desvio padrão	15,95	16,68	17,75
Pior resultado	562	581	539
Melhor resultado	495	510	484

Tabela 5.11: Tempo necessário para reconfigurar os CAs, em cada caso de teste.

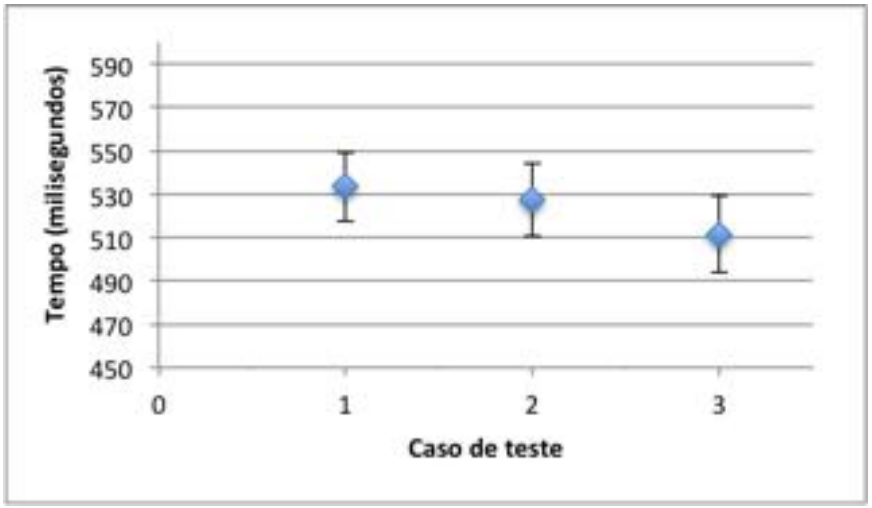

Figura 5.13: Tempo necessário para reconfigurar os CAs, em cada caso de teste.

Analisando os resultados obtidos na configuração dos CAs, é visível que grande parte do tempo utilizado para reconfigurar os CAs é consumido no PSA, pois esta entidade é responsável pela análise e preparação das configurações a enviar para os subsistemas adjacentes.

Com os resultados obtidos consegue-se verificar que o tempo necessário para reconfigurar todos os CAs é, inversamente proporcional ao número de PMAs (subsistemas) existentes no sistema. Visto que, foi projetado um sistema de modo a que a configuração fosse distribuída de forma equitativa pelos CAs, o facto de existirem mais subsistemas e visto que, a configuração é efetuada paralelamente em todo o sistema, diminui o tempo necessário à sua reconfiguração.

5.3 Discussão de Resultados

Com as simulações apresentadas nos subcapítulos 5.1 e 5.2 foi possível validar a arquitetura implementada num ambiente real onde foi configurado um sistema baseado em tecnologia padrão e num ambiente virtual onde foi possível configurar um sistema utilizando as mesma entidades, para um produto e configurações diferentes. Nesta última simulação foi possível analisar a *performance* do sistema a fim de validar a utilidade da arquitetura na redução significativa do tempo de *ramp-up*.

No sistema real a forma como o sistema foi modelado, em que os robôs têm acesso a vários tipos de pinças, faz com que a *Modutec* seja uma plataforma de montagem altamente robusta, flexível e configurável, capaz montar uma dobradiça não só recorrendo apenas a um robô com uma pinça, como também aos dois robôs com mais do que uma pinça [ARC$^+$15].

Dada a complexidade do sistema, a elevada granularidade apresentada foi essencial, pois o fato do sistema conseguir agregar diferentes níveis de complexidade de processos, permitiu simplificar não só a descrição do produto bem como a de todo o sistema. Esta característica tornou-se de extrema importância para que o resultado seja suficientemente expressivo para atender a sistemas de manufatura de maior dimensão [ARC$^+$15].

O facto das entidades responsáveis por abstrair as entidades físicas, os CAs, implementarem uma interface genérica que contém todas as especificações necessárias à interação com qualquer tipo de dispositivos independente do tipo de tecnologia utilizada, evita qualquer tipo de restruturação de sistemas já existentes [RBO$^+$15].

No sistema virtual foi possível submeter o trabalho proposto a diferentes condições de produção e avaliar o seu desempenho, validando-o para diferentes ambientes, independentemente do seu tamanho ou volumes de produção.

Ao fazer uma breve análise nos resultados obtidos pode-se concluir que o sistema permite uma rápida adaptação às necessidades inerentes a um sistema de manufatura ágil, flexível e reconfigurável.

A possibilidade de dividir o sistema em vários subsistemas permite que este tenha uma elevada escalabilidade e independentemente da quantidade de informação a ser processada, o sistema consegue reagir de forma mensurável.

Analisando os tempos associados à reconfiguração do sistema, o pior caso é de 562 milisegundos, o que comparando com sistema de manufatura convencionais é um valor bastante positivo. Também bastante positivo, foi o facto de aumentar do número de subsistemas não comprometer o desempenho da reconfiguração, pelo contrário, devido à escalabilidade conferia pela arquitetura permite que os tempos de reconfiguração diminuam à medida que o sistema aumenta.

Apesar de, a dada altura, para que o desempenho do sistema não fosse comprometido, ter sido imperativo criar uma nova entidade, o PSA, é notório nos resultados obtidos que a carga exigida a esta entidade na reconfiguração ainda é alta. Apesar de os resultados obtidos terem sido bastante positivos é nesta entidade que grande parte do tempo associado à reconfiguração é utilizado.

Tal como referido em [RBO+15], esta implementação permite a reconfiguração de um grande número de sistemas de manufatura e também a sua adaptação às mudanças na demanda, visto que, permite adicionar ou remover em tempo real respondendo às necessidades exigidas pela produção.

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Conclusões

A grande evolução das tecnologias de informação e comunicação levaram ao surgimento de novos paradigmas. Nos últimos anos têm sido apresentados muitos trabalhos que visam tornar os sistemas de manufatura mais robustos, flexíveis e reconfiguráveis, apesar das soluções apresentados solucionarem muitos dos problemas encontrados, de adaptabilidade, de reconfigurabilidade, etc..., estes exigem uma reformulação dos sistemas atuais, pois tornaram-se obsoletos e não suportam novos conceitos de produção.

Devido à conjuntura atual é impossível exigir a empresas em plena crise financeira um restruturação das suas estruturas, que apesar de resolver os seus problemas, o retorno financeiro não é garantido e tarda em aparecer. Assim a solução consiste em desenvolver um conceito capaz de utilizar as estruturas já existentes de modo a formá-las mais robustas e flexíveis.

A tecnologia de agentes tem sido parte da solução pois confere versatilidade e interoperabilidade necessária às estruturas existentes que são rígidas e de controlo hierárquico. Uma arquitetura capaz de se adaptar a diferentes sistemas, de lidar com a incerteza e reconfigurar de forma rápida linhas de produção, aliada a conceitos associados aos MAS, permite que sistemas de manufatura convencionais possam adaptar-se de forma rápida e flexível às constantes necessidades do mercado, produzindo uma grande diversidade de

produtos com reduzidos volumes, sem que isso implique a paragem por grandes períodos de tempo de linhas de produção completas.

O trabalho realizado demonstra que é possível implementar uma arquitetura auto-organizada e escalável capaz de reconfigurar de forma rápida linhas de montagem independentemente da tecnologia e do tipo de dispositivos utilizados.

Uma implementação baseada em conceitos MAS confere ao sistema o carácter distribuído e a autonomia exigida para que este possa responder a todas as expectativas exigidas, sem que seja comprometido o desempenho do sistema.

A escalabilidade característica do sistema permite que este seja utilizado em diferentes sistemas de manufatura, independentemente da informação a ser processada, pois o facto de, ser um sistema completamente distribuído, possibilita a divisam de um sistema em vários subsistema permitindo adaptar de forma simples o sistema às necessidades de produção.

A complexidade caraterística dos sistemas de manufatura podem tornar a descrição de todo o sistema e dos produtos complexa, mas devido à elevada granularidade do trabalho implementado esta complexidade pode ser reduzida.

Os testes realizados em ambiente real permitiram validar e simular a arquitetura na montagem de um produto num sistema de manufatura real e assim comprovar a possível utilização do trabalho realizado noutro ambiente de produção.

Em suma, ao submeter a arquitetura a um ambiente virtual foi possível comprovar a sua utilização condições de produção diferentes, a sua *performance* quando submetida a alterações de topologia e a capacidade de reconfigurar linhas de produção num período de tempo que outrora era impossível.

6.2 Trabalho Futuro

Como trabalho futuro seria importante aproveitar as valências inerentes à estrutura da arquitetura implementada, na monitorização de sistemas de manufatura. Através da interação estabelecida pelas diferentes entidades é possível extrair informações relevantes dos diferentes dispositivos e através de uma análise fornecer dados ao operador a fim de melhorar o desempenho de todo o sistema.

Introduzido um sistema de monitorização a forma como toda a informação é guardada pode ser alvo de estudo, afim de melhorar o desempenho da base de dados desenvolvida uma vez que o acesso à mesma resultou num processo pesado quando a quantidade de informação era elevada.

Com a validação de resultados ficou provado que de todas as entidades o PSA é a entidade à qual é exigido uma maior esforço de processamento na reconfiguração. Assim este poder ser alvo de melhoramentos afim de permitir que o processamento a ele exigido posso ser distribuído por outras entidades.

Outro ponto essencial de revisão é o conceito de auto-organização exigida a este tipo de sistemas, em que, aspetos relacionados com a agregação de competências pode ser melhorado de modo a torna o processamento associado a esta agregação mais simples.

Durante a realização deste trabalho foram escritos dois artigos. Um que permitiu a exposição da arquitetura desenvolvida e a sua capacidade de reconfigurar sistemas de manufatura, independentemente, do tipo de tecnologia utilizada [RBO+15], este artigo irá ser apresentado na conferência DoCEIS (Doctoral Conference on Computing, Electrical and Industrial Systems) em Abril de 2015 [doc15]. O outro artigo permitiu apresentar o trabalho desenvolvido na validação da arquitetura em ambiente real [ARC+15], este artigo irá ser apresentado na conferência INCOM (IFAC Symposium on Information Control in Manufacturing) em Maio de 2015 [inc15].

Referências Bibliográficas

[AAM+00] T Arai, Y Aiyama, Y Maeda, M Sugi, and J Ota. Agile assembly system by "plug and produce". *CIRP Annals-Manufacturing Technology*, 49(1):1–4, 2000.

[ARC+15] Nikolas Antzoulatos, Andre Rocha, Elkin Castro, Lavindra de Silva, Tiago Santos, Svetan Ratchev, and José Barata. Towards a capability-based framework for reconfiguring industrial production systems. *Information Control Problems in Manufacturing (INCOM), 2015 IEEE International Conference*, 2015.

[AVH04] Grigoris Antoniou and Frank Van Harmelen. Web ontology language: Owl. In *Handbook on ontologies*, pages 67–92. Springer, 2004.

[BBG06] Hendrik Bohn, Andreas Bobek, and Frank Golatowski. Sirena-service infrastructure for real-time embedded networked devices: A service oriented framework for different domains. In *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*, pages 43–43. IEEE, 2006.

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

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

[BLAT14] José Barbosa, Paulo Leitão, Emmanuel Adam, and Damien Trentesaux. Dynamic self-organization in holonic multi-agent manufacturing systems: The adacor evolution. *Computers in Industry*, 2014.

[CJdOC10] Gonçalo Cândido, François Jammes, José Barata de Oliveira, and Armando W Colombo. Soa at device level in the industrial domain: Assessment of opc ua and dpws specifications. In *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, pages 598–603. IEEE, 2010.

[doc15] Doceis'15. Available: http://sites.uninova.pt/doceis, 2015.

[DSSG+08] Luciana Moreira Sá De Souza, Patrik Spiess, Dominique Guinard, Moritz Köhler, Stamatis Karnouskos, and Domnic Savio. Socrades: A web service based shop floor integration infrastructure. In *The internet of things*, pages 50–67. Springer, 2008.

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

[FDMSB08] Regina Frei, G Di Marzo Serugendo, and Jose Barata. Designing self-organization for evolvable assembly systems. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO'08. Second IEEE International Conference on*, pages 97–106. IEEE, 2008.

[FEI15] Feintool. Available: http://www.feintool.com, 2015.

[FIP15a] Fipa acl message structure specification. Available : http://www.fipa.org/specs/fipa00061/SC00061G.html, 2015.

[FIP15b] Fipa request interaction protocol specification. Available: http://www.fipa.org/specs/fipa00026/SC00026H.html, 2015.

[H2d14] H2 database engine. Available: http://www.h2database.com/html/main.html, 2014.

[HWS11] Chung-Yuan Huang, Sheng-Wen Wang, and C-T Sun. Modeling agent self-awareness, individual performance and collaborative behavior. In *Intelligent Control and Automation (WCICA), 2011 9th World Congress on*, pages 759–763. IEEE, 2011.

[inc15] Incom. Available: http://www.incom2015.org, 2015.

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

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

[LM06] Stefan-Helmut Leitner and Wolfgang Mahnke. Opc ua–service-oriented architecture for industrial applications. *ABB Corporate Research Center*, 2006.

[MM05] Vladimir Marik and Duncan McFarlane. Industrial adoption of agent-based technologies. *IEEE Intelligent Systems*, 20(1):27–35, 2005.

[MNTW01] Brent A Miller, Toby Nixon, Charlie Tai, and Mark D Wood. Home networking with universal plug and play. *Communications Magazine, IEEE*, 39(12):104–109, 2001.

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

[OAB05] Mauro Onori, Henric Alsterman, and José Barata. An architecture development approach for evolvable assembly systems. In *Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005.(ISATP 2005). The 6th IEEE International Symposium on*, pages 19–24. IEEE, 2005.

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

[OBF06] Mauro Onori, José Barata, and Regina Frei. Evolvable assembly systems basic principles. In *Information Technology for Balanced Manufacturing Systems*, pages 317–328. Springer, 2006.

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

[pri15] Plug and produce intelligent multiagent environment based on standard technology. Available: http://www.prime-eu.com, 2015.

[RB11] Luis Ribeiro and Jose Barata. Re-thinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging it based production paradigms. *Computers in Industry*, 62(7):639–659, 2011.

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

[RBO+15] Andre Rocha, Diogo Barata, Giovanni Di Orio, Tiago Santos, and José Barata. Prime as a generic agent based framework to support pluggability and reconfigurability using different technologies. 2015.

[RBP11] Luis Ribeiro, José Barata, and Joao Pimentao. Where evolvable production systems meet complexity science. In *Assembly and Manufacturing (ISAM), 2011 IEEE International Symposium on*, pages 1–6. IEEE, 2011.

[RDOB+14] Andre Rocha, Giovanni Di Orio, Jose Barata, Nikolas Antzoulatos, Elkin Castro, Daniele Scrimieri, Svetan Ratchev, and Luis Ribeiro. An agent based framework to support plug and produce. In *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*, pages 504–510. IEEE, 2014.

[sim15] Simplan. Available: https://www.simplan.de/en/?lang=en, 2015.

[TQC15] Tqc automation and test solutions. Available: http://tqc.co.uk, 2015.

[Ued92] Kanji Ueda. A concept for bionic manufacturing systems based on dna-type information. In *Proceedings of the IFIP TC5/WG5. 3 Eight International PROLAMAT Conference on Human Aspects in Computer Integrated Manufacturing*, pages 853–863. North-Holland Publishing Co., 1992.

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

[UNI15] Uninova - instituto de desenvolvimento de novas tecnologias. Available: http://www.uninova.pt, 2015.

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