Leidir Horta Monteiro

Licenciado em Ciências de
Engenharia Eletrotécnica e de Computadores

Sistema de simulação baseado em agentes usando o V-REP

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

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

Co-orientador: André Dionísio Bettencourt da Silva Parreira Rocha,
Mestre, CTS Uninova

Júri:

Presidente: Prof. Doutor Paulo da Costa Luís da Fonseca Pinto

Arguente: Prof. Doutor Tiago Oliveira Machado de Figueiredo Cardoso

Vogal: Prof. Doutor José António Barata de Oliveira

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2017

Sistema de Simulação Baseado em Agentes usando o V-REP

À minha família.

Agradecimentos

Nesta secção, gostaria de aproveitar a oportunidade para expressar os meus mais sinceros agradecimentos a todos os que contribuíram para que esta dissertação fosse realizada com sucesso.

Ao meu orientador Professor Doutor José Barata, pela oportunidade de desenvolver esta dissertação no campo da robótica, pela motivação que me deu desde do primeiro até ao último ano do curso e pela partilha da sua história nas ilhas de Cabo Verde.

Ao meu coorientador Mestre André Rocha, pelas sugestões apresentadas, paciência, orientação e tempo despendido, assim como toda a ajuda depositada que foi essencial para o desenvolvimento desta dissertação.

A todos os meus professores que tive em toda a minha vida e em especial ao professor Luís Bernardo por tudo e aos restantes professores do Departamento de Engenharia Eletrotécnica da Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa que contribuíram para minha formação.

À FACIT, pelo apoio disponibilizado nos últimos anos do curso.

Aos meus amigos e colegas, pelos momentos de diversão, sofrimentos, sacrifícios e apoios fornecidos e em especial aos que apoiaram diretamente nesta dissertação.

À minha tia pelo carinho, compreensão, aconchego, motivação e apoio disponibilizado durante todo estes anos.

Por último, não menos importantes, à minha família em especial aos meus pais pela educação, valores, compreensão, suporte, carinho, motivação e por tudo, pois sem eles não estaria aqui.

Resumo

Atualmente o mercado apresenta elevadas flutuações na procura por produtos personalizados, com baixo custo, alta qualidade e com ciclos de vida curtos. Portanto, as empresas são obrigadas a adaptarem-se aos novos paradigmas de produção para poderem dar resposta às elevadas flutuações na procura e as exigências do mercado. Estas exigências geram nas empresas necessidades de novas soluções para tornarem os sistemas de produção mais ágeis e flexíveis, uma vez que as abordagens tradicionais estão sendo inadequadas. Por forma a tornar os sistemas ágeis e flexíveis surgiram novos paradigmas de produção que oferecem às empresas uma elevada capacidade de adaptação na produção, o que possibilita as empresas dar resposta as elevadas flutuações na procura. Portanto, adicionalmente permite obter menor custo face as abordagens tradicionais.

Com o avanço das tecnologias surgiram várias propostas que visam tornar os sistemas de produções mais flexíveis, adaptáveis, evolutivas e escaláveis. Uma dessas propostas são os sistemas multiagentes. Estes sistemas têm carácter distribuídos, auto-organizáveis e capazes de reagir de forma rápida às perturbações ou situações de falhas iminentes das linhas de produção.

A inovação e a melhoria contínua na produção levam as empresas de manufaturas a recorrer às tecnologias de simulação para testar, validar e avaliar os seus processos de produção. Neste contexto, é proposta uma arquitetura que visa apresentar uma solução de simulação para sistemas controlados por sistemas multiagentes. Esta arquitetura foi testada usando uma topologia da célula NOVAFLEX. Os resultados comprovam que é possível integrar sistemas multiagente num simulador 3D, possibilitando adaptar os sistemas de produções.

Palavras-chave: Sistema Multiagente, Integração do Sistema, V-REP, Simulação, Sistema de manufatura.

Abstract

Currently, the market has high fluctuations in the demand for customized products, with low cost, high quality and with short life cycles. Just as companies are forced to adapt to new production paradigms in order to respond as high fluctuations in demand for market demands. These demands create a need in companies for new solutions to make production systems more flexible, since traditional approaches are inadequate. In order to develop and flexible systems new production paradigms have emerged for companies with a high capacity for adaptation in production, which enables companies to respond as high fluctuations in demand. Please get cheaper as traditional face.

With the advancement of technologies, several proposals have appeared that aim to make production systems more flexible, adaptable, evolutionary and scalable. One such proposal is multi-agent systems. These systems are distributed, self-organizing, and capable of reacting quickly to disturbances or situations of imminent failure of production lines.Innovation and the continuous improvement of technologies in the production areas have led manufacturing companies to use simulation technologies to test, validate and evaluate their production processes.

Innovation and continuous improvement in production lead manufacturers to use simulation technologies to test, validate and evaluate their production processes. In this context, it is proposed an architecture that aims to present a simulation solution for systems controlled by multi-agent systems. This architecture was tested using a NOVAFLEX cell topology. The results prove that it is possible to integrate multiagent systems in a 3D simulator, allowing to adapt the production systems.

Keywords: Multi-agent System, System Integration, V-REP, Simulation, Manufacturing system.

Acrónimos

ABS	Agent Based Simulation
ACL	Agent communication Language
ADACOR	Adaptive holonic control Architecture for Distributed Manufacturing Systems
AGV	Automatic Guided Vehicle
AHP	Analytical Hierarchical Process
API	Application Programming Interface
ASk	Atomic Skill Agent
BMS	Bionic Manufacturing Systems
CLA	Coalition Leader Agent
CSk	Complex Skill
DA	Deployment Agent
DES	Discrete Event Simulation
DF	Directory Facilitator
DML	Dedicated Manufacturing Systems
EPS	Evolvable Production Systems
FMS	Flexible Manufacturing Systems
HMS	Holonic Manufaturing Systems
HUA	Handover Unit Agent
IDE	Integrated Development Environment
IMS	Intelligent Manufacturing Systems
IP	Internet Protocol

IPC	Inter-Process Communication
JADE	Java Agent Development Plataform
MAS	Multi-Agent Sistems
PA	Product Agent
PROSA	Product-Resource-Order-Staff Architecture
PSiA	Product Sink Agent
PSoA	Product Source Agent
RA	Resource Agent
RMS	Reconfigurable Manufacturing Systems
SD	System Dynamic
SIMIO	Simulation Modeling Framework based on Intelligent Objects
TCP	Transmission Control Protocol
TEA	Transport Entity Agent
V-REP	Virtual Robot Experimentation Platform

Conteúdo

Índice de figuras

Índice de Tabelas

1

INTRODUÇÃO

1.1 Descrição do problema

A procura por produtos altamente personalizados e diversificados geraram várias necessidades às empresas de manufaturas. E, portanto, obrigando as empresas a reformularem os seus conceitos de produções, de modo a serem flexíveis, ágeis, adaptáveis e sensíveis a esta forma de procura. Isto é, as empresas sentem a necessidade de desenvolver novos modelos de produção de forma a dar a resposta às constantes alterações e flutuações na procura.

Para colmatar as flutuações dos mercados, surgiram vários paradigmas de produção. Estes paradigmas promoveram uma maior flexibilidade e dinamismo nos sistemas de produção. No entanto, atualmente, estes paradigmas não são suficientes para colmatar todas as necessidades, uma vez que ainda existe lacunas na adaptabilidade, modularidade e escalabilidade nos sistemas de manufatura e produção.

Para colmatar estas lacunas várias abordagens nos sistemas de manufaturas foram propostas. As soluções mais evolutivas são propostas devido à aplicação da informática aos sistemas de produção. Esta aplicação tem trazido uma nova visão para lidarem com os sistemas mais complexos. Os sistemas multiagente são umas das soluções propostas. Estes sistemas têm carácter distribuído e altamente flexível e capaz de reagir de forma rápida as perturbações ou situações de falhas nos sistemas de manufatura.

A dimensão e complexidade dos sistemas de produção obrigaram as empresas a procurarem ferramentas que permitam testar, avaliar e otimizar os seus processos de produção de forma seguro, rápido, eficaz e com custo reduzido. A solução mais evidente passa pela simulação das linhas de produção. A simulação oferece possibilidade em averiguar todas as condições dos sistemas ou até novas metodologias que se pretende implementar ou otimizar.

Nesta dissertação apresenta-se e valida-se uma implementação baseada em sistemas multiagente e um simulador. Esta arquitetura permite integrar essas duas tecnologias e averiguar os seus comportamentos em diversas situações na simulação das linhas de produções.

1.2 Pergunta de Investigação e hipóteses

Tendo em conta os problemas mencionados, nesta dissertação pretende-se analisar as seguintes questões:

Será possível simular um sistema de produção altamente flexível e dinâmico, controlado por um ambiente multiagente com as tecnologias existentes?

Recorrendo ao V-REP com uma camada de integração implementada para o efeito, será possível integrar e simular o controlo com um sistema multiagente num ambiente fabril.

1.3 Visão Geral do Trabalho Realizado

Nesta dissertação é proposta uma arquitetura capaz de integrar um sistema baseado em novas abordagens de produção, sistemas multiagentes e um simulador V-REP. Este simulador é muito versátil, escalável para simulação em 3D. O objetivo deste trabalho é provar que é possível garantir a interoperabilidade entre o sistema multiagente e o V-REP.

A arquitetura proposta é composta por três camadas: execução, integração e simulação. A camada de execução é baseada na arquitetura IADE, responsável por manter toda a inteligência, a dinâmica, agilidade e flexibilidade do sistema. A camada de execução é constituída por conjuntos de agentes divididos em três grupos: de suporte, de execução e transportes. O grupo de suporte é responsável pelo lançamento, inscrições e pesquisa dos agentes na plataforma. O grupo de execução é responsável pelas ações sobre os sistemas de produção, como a decisão sobre o plano de produção, o controlo de produção de cada produto, execução das habilidades e pela abstração das componentes físicas (como por exemplo os robôs, operadores humanos, etc.). Por último, o grupo de transporte é responsável pela abstração das componentes físicas numa linha de produção como as passadeiras, os encaminhadores, pontos de entrada e de saída. Os objetivos desses agentes são garantir o dinamismo, agilidade, eficiência e organização nas deslocações dos produtos desde a sua entrada até a saída do sistema de transporte.

A camada de integração permite a interação entre as duas tecnologias distintas de forma clara e sucinta. Para garantir esta interação é utilizado o mecanismo de comunicação, *socket*. Este mecanismo permite conectar e implementar um modelo cliente/servidor, onde o cliente e o servidor são os agentes e o simulador respetivamente. No que diz respeito ao cliente é implementada uma interface para cada agente onde são chamadas através de funções API's que permitem interagir diretamente com o servidor. A comunicação é feita da seguinte forma, inicialmente, o cliente conecta ao servidor e posteriormente, cada agente comunica com o

servidor através de um ID fornecido durante a fase de conexão. Este ID varia de acordo com o porto de conexão.

A camada de simulação permite simular a linha de produção, onde os componentes virtuais inseridos no ambiente do simulador são organizados em dois módulos: de recursos, constituídos pelos robôs, operadores humanos etc.; e os módulos de transporte, são constituídos pelos modelos de passadeiras, encaminhadores, pontos de entrada e de saída. Os comportamentos de cada módulo foram programados num *child script* de cada modelo utilizado no simulador.

As tecnologias de suporte utilizadas para a implementação da arquitetura foi o Java, JADE e V-REP. Para validação desta arquitetura foi implementada uma topologia da célula NOVAFLEX que se encontra instalada na Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa.

1.4 Principais contribuições

As principais contribuições deste trabalho incluem uma arquitetura que permite integrar duas tecnologias distintas, isto é, o sistema multiagente e o simulador V-REP, que é um simulador moderno, versátil muito utilizado nas áreas da robótica.

Esta dissertação oferece uma solução que visa resolver a falta de simuladores existentes no mercado, para simular novos sistemas de controlo como os sistemas multiagente. A integração permite simular as ações dos agentes sobre as linhas de produção dando a possibilidade de testar as capacidades e os limites dos sistemas de produção.

A integração entre o MAS e V-REP permite facilmente adicionar e integrar outros modelos no simulador com ou outro agente constituinte no MAS, graça às API's oferecidas pelo simulador. Na arquitetura proposta, a camada de execução foi implementada baseada na arquitetura IADE (Ribeiro, Barata, Onori, & Hoos, 2015), que permite adaptar outros tipos de arquitetura baseadas em agentes com o simulador V-REP.

1.5 Estrutura da Dissertação

Esta dissertação encontra-se organizada em sete capítulos: Introdução, Estado da Arte, Arquitetura, Implementação, Validação e Testes, Conclusões e Trabalhos Futuro e Referencias Bibliográficas.

No primeiro capítulo, **Introdução**, contextualiza-se o problema dos sistemas de manufatura, e de seguida apresenta-se algumas perguntas de investigação, visão geral da dissertação e as principais contribuições.

No segundo capítulo, **Estado da Arte**, apresenta-se os estudos realizados por outros autores relacionados com sistema de manufatura e a sua simulação. Na parte final do capítulo

são apresentados alguns paradigmas de produção e os principais simuladores utilizados em sistemas de manufatura.

No terceiro capítulo, **Arquitetura**, descreve-se a arquitetura proposta nesta dissertação. Esta arquitetura baseia-se num sistema multiagente e a sua integração com um simulador V-REP.

No quarto capítulo, **Implementação**, descreve-se detalhadamente as implementações efetuadas da arquitetura proposta.

N quinto capítulo, **Validação e Testes**, ilustra-se vários testes e resultados obtidos da simulação do projeto.

No sexto capítulo **Conclusões e Trabalho Futuro**, são apresentadas as principais conclusões e na parte final são apontados alguns estudos futuros.

E por último, o sétimo capítulo, **Referências Bibliográficas**, são apresentados todas as fontes bibliográficas usadas para os estudos relacionados com este documento.

2

ESTADO DA ARTE

Neste capítulo apresenta-se alguns estudos relacionados com os sistemas de produções e manufaturas. Inicialmente, descreve-se os principais paradigmas de produção, posteriormente descreve-se o sistema multiagente e alguns simuladores utilizados num ambiente de produção e manufatura. Na parte final do capítulo apresenta-se a conclusão geral.

2.1 Paradigmas de Produção e Manufatura

Nos últimos anos os sistemas de produção e manufatura enfrentaram várias mudanças significativas devido à globalização e à competitividade dos mercados. Estas mudanças provocam novas tendências e exigências na procura de variedades e produtos padronizados, com alta qualidade e baixo custo. Face à esta exigência do mercado e ao aumento da procura, as empresas são obrigadas a manter competitividade com melhor flexibilidade e agilidade na produção.

Umas das primeiras abordagens para dar respostas as exigências foi a produção em massa protagonizada no início do século XX por Henry Ford. Nesta abordagem provou-se que a utilização da linha de montagem com tarefas específicas e sistematizadas, permite aumentar a qualidade e rapidez na fabricação de produtos e também aumentar quantidade significativa de produtos não-diversificado e a diminuição dos preços (Ribeiro & Barata, 2011). A desvantagem desta abordagem é a não flexibilidade na diversidade ou variedades de produtos. Segundo a filosofia do Henry Ford é que os clientes deveriam adaptar ao produto e não vice-versa. A prova disso a sua famosa declaração "*have a car of any colour they wanted as long as it was black*". Os produtos são produzidos com poucas variedades, deixando os consumidores com poucas ou nenhuma opção de escolha, numa era em que a procura por produtos altamente personalizados era cada vez maior.

A necessidade de produzir pequeno lote de produtos altamente personalizados é cada vez maior para dar respostas as flutuações de mercado. Portanto as empresas sentem a

necessidade de inovar a linha de produção através da customização em massa popularizada por Joseph Pine II (Ribeiro & Barata, 2011). O núcleo desta inovação é aumentar rapidamente a produção dos produtos variados e personalizados mantendo os custos baixo.

Só ser flexível não é suficiente para dar resposta a procura e, portanto há a necessidade de ser ágil na produção que provoca a instabilidade e imprevisibilidade dos mercados heterogéneos e fragmentados. As empresas lidam com as mudanças imprevisíveis, por isso ser ágil para lidar com as incertezas (Ribeiro & Barata, 2011). Dentro do sistema ágil surgiram novos paradigmas de produção tais como *Reconfigurable Manufacturing Systems* (RMS), *Bionic Manufacturing Systems* (BMS), *Holonic Manufacturing Systems* (HMS) e *Evolvable Production Systems* (EPS) além do FMS que já tinha sido desenvolvido na década 80. Na Figura 2.1 encontra-se a cronologia da evolução do paradigma da manufatura.

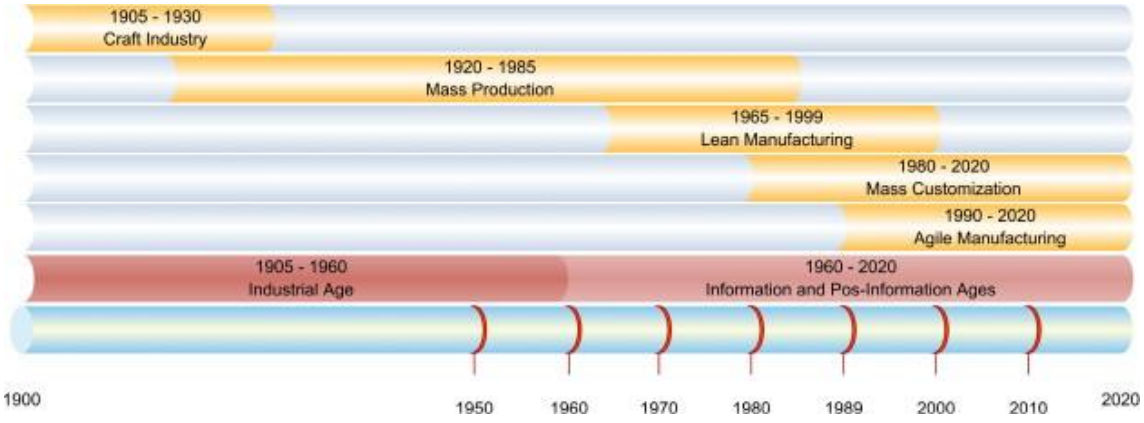


Figura 2.1 Cronologia da evolução de sistema de manufatura (Ribeiro & Barata, 2011)

Segue-se a discrição detalhada de novos paradigmas de produções.

2.1.1 Sistemas Flexíveis de Manufatura

Os FMS's (*Flexible Manufacturing Systems*) surgiram devido ao aumento da competitividade e a exigência do mercado, com um elevado número de procura de diversos tipos de produtos padronizados onde o custo, a qualidade e a velocidade de entrega tornaram-se fatores necessários aos clientes. Face às necessidades dos consumidores, surgiu uma nova estratégia de customização em massa. As empresas tiveram de se adaptar de forma a corresponder às demandas, então para isso os sistemas tinham que ser altamente flexível.

O FMS é composto por conjunto de máquinas interligadas entre si através de um sistema de transporte de materiais. A sincronização entre as máquinas e o sistema de transporte é controlado pelo sistema central (Shivanand, Benal, & Koti, 2006). Desta forma as máquinas serão versáteis e permitirem a produção de diferentes tipos de produtos em curto intervalo de tempo.

Estas versatilidades das máquinas são obtidas através da habilidade de trocas de ferramentas, que permite inúmeras diferentes combinações, aumentando o número de operações possíveis.

Em comparação com outra abordagem, como por exemplo o DML (*Dedicated Manufacturing Systems*), que baseia-se na automação fixo onde normalmente cada linha dedicada é projetada para produzir um único produto, e portanto, é considerado relativamente barata o custo por peça. No entanto no FMS é permitido a produção de diferentes tipos de produtos num único sistema (Y. Koren et al., 1999).

Para tornar o FMS mais viável, vários estudos foram realizados como por exemplo, em (Jahromi & Tavakkoli-Moghaddam, 2012) foi proposto um modelo de programação para resolver o problema da seleção dinâmica da máquina-ferramenta e alocação de operação no ambiente FMS. O objetivo deste modelo é determinar uma combinação da máquina-ferramenta para cada tipo de operação de modo a minimizar os custos de produção, nomeadamente os custos de instalação, de movimentação das ferramentas e de manuseamento.

Em (Pach, Berger, Sallez, & Trentesaux, 2015) é proposto um modelo de controlo de FMS baseada em campos potenciais para alocar e encaminhar dinamicamente os produtos para os recursos existentes no sistema de produção. O objetivo deste modelo é minimizar o tempo total de produção e otimizar o consumo de energia dos recursos. O desperdício de energia é limitado através do controlo dos estados de recursos em tempo real.

Este paradigma apesar de proporcionar a produção de diversos tipos de produtos no sistema de manufatura e acomodar algumas mudanças na demanda, que é só praticável quando essas mudanças forem previsíveis. Portanto, este paradigma apresenta algumas fragilidades como a incapacidade de lidar com a incerteza e a evolução do ciclo de vida do sistema de produção. Não obstante, é adequado para lidar com as flutuações dos mercados, uma vez que um dos principais objetivos do sistema de manufatura é fornecer produtos de alta qualidade praticamente instante (Ribeiro & Barata, 2011). Devido a esta lacuna do FMS surgiram novos paradigmas de produção.

2.1.2 Sistemas Reconfiguráveis de Manufatura

A necessidade de lidar com as rápidas mudanças dos sistemas de produção, levaram ao surgimento de um novo paradigma de produção, o RMS (*Reconfigurable Manufacturing Systems*). Este paradigma permite a rápida implementação, reconfiguração de equipamentos e/ou componentes, como por exemplo máquinas, células etc. Estas componentes podem ser adicionadas, removidas ou modificadas de modo a dar resposta as novas funções do sistema, para responder a procura (El Maraghy, 2006).

O RMS a contrário do DMS e FMS, as suas capacidades e funcionalidades não são fixos, uma vez que o sistema pode adaptar ou mudar ao longo do tempo de acordo com as necessidades do mercado. O RMS além de permitir a flexibilidade na produção de vários

produtos diferentes num curto intervalo de tempo, permite alterar o seu próprio sistema. O RMS altera o sistema se existir as seguintes caraterísticas como a modularidade, integrabilidade, customização, convertibilidade e diagnosticabilidade. O principal objetivo do RMS é permitir a reconfiguração dos seus componentes de software e hardware em simultâneo (Y. Koren et al., 1999) (Yoram Koren & Shpitalni, 2010).

A eficiência e dinamismo do RMS foram estudados, com objetivo de melhorar a capacidade de resposta do sistema. (Kapil Kumar & Pramod Kumar, 2014) e (Mittal & Jain, 2014) apresentam várias medidas do desempenho do RMS, nomeadamente o tempo de reconfiguração, tempo de espera, tempo de execução, tempo do *rump-up,* custos, confiabilidade e disponibilidade. Estas medidas apresentam grandes efeitos na eficiência do sistema. Também em (Nazarian, Ko, & Wang, 2010) é apresentado um modelo que permite otimizar a eficiência do sistema reconfigurável.

Em (Abdi, 2005) foi proposto um modelo AHP (*Analytical hierarchical Process*) para a seleção de configurações de *layout* do RMS. Para configurar diferentes tipos de *layout* durante a alteração dos produtos.

Embora um RMS seja considerado um sistema altamente automatizado, não quer dizer que a presença humana é indispensável. Pelo contrário é inevitável neste paradigma a presença do homem no processo de reconfiguração, uma vez que este processo envolve a interação entre homem e a máquina tanto aos níveis físicos como cognitivos. Em (Eldardiry, 2000) apresenta o efeito da contribuição humana na flexibilidade e dinâmica do sistema reconfigurável.

2.1.3 Sistemas Biónicos de Manufatura

Os sistemas biónicos de manufatura são inspirados nos sistemas biológicos (Kanji Ueda, 1992). Este paradigma faz um paralelo entre as propriedades de sistemas biológicas e sistemas de manufatura. Onde as mudanças imprevisíveis no ambiente de produção são feitas com base nos fundamentos biológicos, nomeadamente auto-organização, adaptação e evolução (Kanji Ueda, Hatono, Fujii, & Vaario, 2000). Este paradigma baseia na hierarquia das camadas, tal como no organismo vivo (composto por um conjunto de órgãos diferentes). Estes órgãos interagem e trocam dinamicamente as informações entre as camadas da hierarquia. As informações trocadas fazem com que os organismos evoluam de geração em geração.

O BMS (*Bionic Manufacturing System*) foi desenvolvido na década 90, onde se pretendeu adaptar as características de um sistema biológicos tais como autonomia e comportamento espontâneo para um sistema de manufatura, essas caraterísticas eram desenvolvido como uma combinação de vários sistemas autónomo nomeadamente robôs, máquinas-ferramentas, AGVs (*Automatic Guided Vehicle*) etc. que eram relacionados e ordenados hierarquicamente e assim formavam um sistema global (Tharumarajah, Wells, & Nemes, 1998).

Segundo (Kanji Ueda et al., 2000) os organismos biológicos são capazes de adaptar-se às mudanças ambientais em que estão sujeitas e auto-sustentar através de funções como auto-reconhecimento, auto-crescimento, auto-recuperação e evolução. Estes organismos são caraterizados por dois tipos de informações biológicos *DNA-type* que evolui de geração a geração (hereditária). O *BN-type* adquire informações aprendidas individualmente ao longo do tempo. A combinação dessas informações para uma entidade individual torna os sistemas vivos, autónomos e adaptativos. Todos os elementos no BMS como robôs, máquinas-ferramentas, AGVs e ferramentas de trabalho devem ser visto como um organismo autónomo e de informações do *BN-type,* e os produtos são desenvolvidos a partir das matérias-primas que contém as informações do *DNA-type*. O produto final já contém os dois tipos de informações biológicas (Figura 2.2).

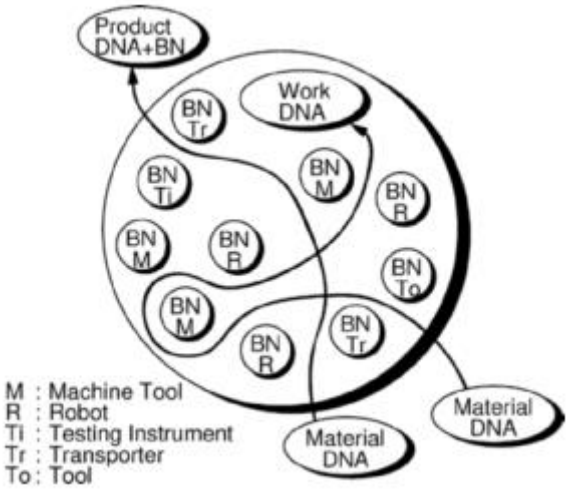


Figura 2.2 Exemplo do conceito de BMS na fábrica (K

2.1.4 Sistemas Holónicos de Manufatura

A competitividade e a concorrência global têm estado a aumentar nos últimos anos, onde os mercados estão constantemente a mudar, as exigências de produtos altamente customizados, o volume do produto de baixa e alta variedade, evolução da tecnologia, baixo ciclo de vida dos produtos obrigaram as empresas a melhorar a produção. A fim de sobreviver à essas mudanças surgiu um novo paradigma HMS (*Holonic Manufaturing Systems*).

O termo *holon* foi apresentado no livro do autor Arthur Koestler (Koestler, 1967). Este autor desenvolveu os conceitos básicos de sistemas holónicos, para explicar as tendências de auto-organização dos sistemas sociais e biológicos. Este termo teve origem na combinação da palavra grega "*holos",* que significa *"todo",* com o sufixo *"on"* que significa *"parte"* (Koestler, 1967).

Os investigadores do IMS (*Intelligent Manufacturing Systems*) adaptaram o conceito *holon* de Koestler para o sistema de manufatura (Suda, 1989) (Christensen, 1994) e (van Leeuwen & Norrie, 1997):

- **Holon**: Um bloco de construção autónomo e cooperativo de um sistema de produção. Este bloco permite transformar, transportar, armazenar e/ou validar as informações de objetos físicos. O *holon* consiste em uma parte de processamento de informação e parte de processamento físico. Um *holon* pode fazer parte de outro *holon.*
- **Autonomia**: A capacidade de uma entidade criar e controlar a execução dos seus planos e/ou estratégias.
- **Cooperação**: o processo pelo qual um conjunto de entidades desenvolve e executa planos de processos aceitos mutuamente.
- **Holarquia**: um sistema de *holon* que podem cooperar para um determinado objetivo. O holarquia define as regras básicas para a cooperação dos *holons* e, assim limita a sua autonomia.
- **Sistemas holónicos de Manufatura (HMS)**:um holarquia que integra toda a gama de atividades de fabricação a partir da reserva em ordem através de conceção, produção e marketing.

O HMS é um sistema de produção capaz de ser virtualmente reconfigurável, onde os humanos, máquinas, células e módulos de programa podem interagir dinamicamente formando grupos virtuais. Este sistema pode ser modelado como inteligente, distribuído, autónomo e elementos (*holons*) cooperativos, no entanto estes formam um sistema extensível e escalável. (van Leeuwen & Norrie, 1997).

O HMS aborda uma arquitetura de um sistema de produção altamente descentralizada, construída a partir de uma mistura modular padronizada, autónomo, cooperativo e elementos inteligentes (van Leeuwen & Norrie, 1997). Este conceito combina as melhores características de organização hierárquicas e heterárquicas para preservar a estabilidade de uma hierarquia e proporcionando flexibilidade dinâmica de uma heterárquia (Van Brussel, Wyns, Valckenaers, Bongaerts, & Peeters, 1998).

Na literatura existem várias arquitetura baseada em HMS como a *Product-Resource-Order-Staff Architecture* (PROSA) (Van Brussel et al., 1998). A PROSA é constituída por três tipos de *holons* básicos (Figura 2.3) denominados de *resource holons, product holons e order holons.* Cada holons desempenha uma função diferente no sistema de produção, utilizando os conceitos de orientação a objetos como a agregação e a especialização. Posteriormente o *Staff holons* pode ser adicionado à estrutura da arquitetura, esta entidade não é obrigatória mas tem funções importantes na cooperação com os *holons* (Van Brussel et al., 1998). A PROSA carateriza-se por dois conceitos básicos, a agregação e a especialização. Onde a agregação

consiste num conjunto de *holons* que interagem entre si, agrupando e formando um *holon* maior com a sua própria identidade. Dependendo do contexto a agregação pode ser divida em *sub-holon* ou considerada como um todo. A especialização permite a separação dos diferentes tipos de *holons* básicos de acordo com as características (Van Brussel et al., 1998).

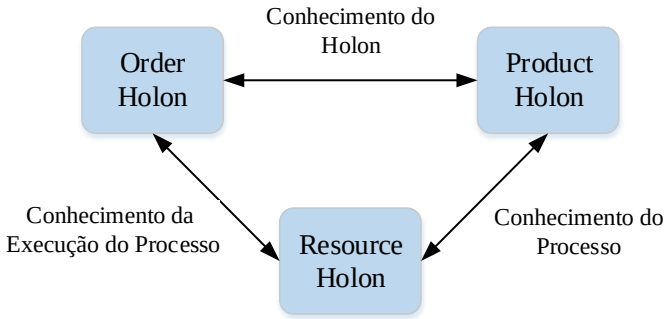


Figura 2.3 Relações entre holons básicos adaptado do (Van Brussel et al., 1998)

A ADACOR (*Adaptive holonic control Architecture for Distributed Manufacturing Systems*) consiste num conjunto de *holons* autónomos e inteligentes que têm como objetivo melhorar o desempenho do sistema de produção, principalmente quando ocorre perturbações inesperadas (Leitão, Colombo, & Restivo, 2003).

A ADACOR é constituída por quatros tipos de *holons*: *product holon, task holon, operational holon* e *supervisor holon.* O *product holon* representa cada produto a ser produzido e contém todo o conhecimento e é responsável pelo plano de produção. O *task holon* é responsável pelo controlo e supervisão da ordem da execução do sistema e contém a informação dinâmica. O *operational holon* representa os recursos físicos de produção tais como robôs, operadores. E por fim, o *supervisor holon* coordena as atividades dos *holons* no dominio, é responsável pela otimização do sistema (Leitão et al., 2003).

A implementação do HMS faz com que os sistemas de produção sejam capazes de reconfigurarem de forma rápida, eficaz e ágil. O sistema torna-se altamente flexível e dinâmico capaz de se organizar de acordo com as suas necessidades e as variáveis imprevistas do sistema.

2.1.5 Sistemas Evolutivos de Produção

Numa fábrica a mudança de operação é inevitável, devido a globalização do mercado em permanente evolução. As variedades, padronização e custos baixos dos produtos são algumas das exigências do mercado. As empresas de produções atuais, além de procurar uma solução eficaz às demandas, também não deixam de preocupar com as suas sustentabilidade, economia e até com ecologia. Neste sentido o EPS (*Evolvable Production Systems)* foi desenvolvido com objetivo de encontrar soluções tecnológicas e mecanismo (Neves & Barata, 2009).

Os princípios de modularidade, integrabilidade, flexibilidade, convertibilidade e diagnosticabilidade levaram ao desenvolvimento de novas abordagens. Uma vez que a adição ou remoção dinâmica dos componentes são necessários dependendo do plano de produção. A adaptação do sistema quando um módulo é removido ou adicionado e a capacidade de diagnóstico global, contribuiu para o desenvolvimento do EPS. Este paradigma garante a execução adequada de um sistema onde os componentes e as condições do ambiente de trabalho podem ser alterados de forma dinâmica. (Onori & Barata, 2009)

O EPS é caracterizado por vários módulos inteligente com controlo integrado. Estes módulos são capazes de se adaptarem dinamicamente as mudanças de acordo com as suas necessidades de produções, devido a capacidade de auto-controlo, auto-diagnóstico, auto-reconfigurável, auto-evolutivo, auto-ajuste, reutilização do sistema mecânico e interoperabilidade dos módulos. Estes fatores contribuíram para a diminuição do tempo de implementação, eficiência e rapidez na produção (Neves & Barata, 2009) (Onori & Barata, 2009). Este paradigma foi desenvolvido para tornar um sistema de produção mais dinâmico, para lidar com a mudança ou troca de produto tendo sempre em conta o tempo *rump-up* e também os picos de demanda (Onori & Barata, 2009).

2.2 Sistemas Multiagente

Nos últimos anos houve crescimento considerável no desenvolvimento de sistemas de produção distribuídos e inteligentes, devido ao MAS (*Multi Agent Systems*). O MAS é constituído por agentes inteligentes e autónomos uma vez que os agentes colaboram dinamicamente entre si para alcançar objetivos. Através destas características, os MAS são adaptados aos sistemas de produções, onde são divididos em produtos, ferramentas, máquinas, etc.

Em (Abdelkader, Bakhta, & Abdelkader, 2012) o MAS é definido como um rede acoplada de agentes que colaboram entre si para dar respostas aos problemas que estão além das suas capacidades ou conhecimentos individuais. Estes agentes são autónomos e funcionam em ambientes distribuídos, onde são estabelecidas objetivos locais ou globais. Esta abordagem apresenta algumas vantagens como a sua natureza de controlar a complexidade de sistemas distribuído, a construção de sistemas escaláveis. Uma vez que a adição de mais agentes torna uma tarefa fácil, mais robusto e tolerante as falhas.

Apesar de não existir um consenso no que diz respeito à definição dos agentes em (Monostori, Kumara, & Váncza, 2006) definiu o agente como um sistema computacional que se encontra localizado num ambiente dinâmico e é capaz de reproduzir um comportamento autónomo e inteligente no sistema. O MAS é originário através das interações do conjunto de agentes, que funcionam como um todo.

Um agente tem a capacidade de compreender e observar o ambiente onde encontra-se integrado. O agente apresenta algumas propriedades básicas tais como a autonomia capacidade de tomar as suas próprias decisões para controlar tanto o estado interno como o comportamento

no ambiente, é racional age da maneira mais apropriada para cada situação, fazendo o melhor que pode de si, apresenta algum tipo de inteligência para poder aplicar as regras fixas para o raciocínio, assim tendo capacidade para o planeamento e aprendizagem. Os agentes possuem outras caraterísticas como a mobilidade, genuinidade e transparência (Monostori et al., 2006).

O MAS paradigma tem sido utilizado noutro vertente como. Em (Saba, Laallam, Hadidi, & Berbaoui, 2015) o MAS é usado no gerenciamento de sistemas de energia renovável multi-fonte, que eram gerenciados por abordagens centralizados. O objetivo da integração do MAS era obter um alto nível de flexibilidade, não só durante a operação, mas também durante todo o ciclo de vida do sistema. Os autores definiram cada elemento que compõe o sistema de energia como um agente do MAS.

O MAS tem sido utilizado para resolver problemas de escalonamento. Em (Kanaga & Valarmathi, 2012) é apresentada uma implementação desenvolvida numa plataforma de multiagente, JADE, (Bellifemine, Poggi, & Rimassa, 1999) aplicaram o conceito do MAS para reduzir o tempo de espera do paciente no hospital.

O MAS é uma tecnologia que atrai muitas áreas distintas de pesquisas, por causa das suas características como a descentralização, adaptabilidade, robustez, escalabilidade, autonomia, etc. Em (Khan & Wang, 2017), o MAS é utilizado para controlar e otimizar o uso ideal de energia elétrica em *MicroGrid*.

Em sistemas de manufatura é essencial ter dois processos, como o planeamento e escalonamento da produção. O planeamento tem como função especificar quais os recursos existentes na linha de produção e que operações são necessárias à produção de um determinado produto. Por outro lado, o escalonamento tem como função atribuir as tarefas entre as máquinas do sistema de produção. Como é possível reparar, existe pouca diferença entre eles, sendo assim, em (Li, Zhang, Gao, Li, & Shao, 2010) foi desenvolvida uma abordagem baseada em agentes que propõe a integração entre essas duas tarefas de forma a melhorar o desempenho do sistema. Esta abordagem baseada em agentes inteligentes apresentaram grandes importâncias para o sistema de manufatura por causa da sua alta flexibilidade, autonomia, descentralização e adaptabilidade sobre os sistemas.

2.3 Simulação dos Sistemas de Manufatura

A concorrência na indústria de produção e as exigências do mercado, rapidez na entrega, qualidade dos produtos levaram as empresas a recorrer à sistema de simulação de modo a prever o erro, analisar, testar e melhorar os seus produtos. Apresentando assim a vantagem de não desperdício dos materiais ou produtos nos testes. A simulação é seguro, mais barata e mais rápida do que as experiências. A simulação assume cada vez mais um dos principais papéis na criação, inovação e otimização de processos industriais.

2.3.1 Definições e conceitos

Existem muitas formas de definir a simulação dos sistemas:

- Em (Banks, 1999) a simulação é a imitação do funcionamento de um processo ou sistema do mundo real ao longo do tempo e envolve a criação e observação de uma história artificial do sistema. A simulação é usada para descrever e analisar o comportamento do sistema (Banks, 1999).
- Em (Gebus, Martin, Soulas, & No, 2004) a simulação significa imitação do comportamento de um sistema dinâmico através do tempo
- Em (Shannon, 1998) a simulação é definida como o processo de desenhar um modelo de sistema real com o propósito de compreender o comportamento do sistema e/ou avaliar várias estratégias para o funcionamento do sistema.
- Em (Maria, 1997) a simulação é uma ferramenta para avaliar o desempenho de um sistema, sob diferentes configurações de interesse e durante longos períodos de tempo.

O sistema e o modelo são termos indispensáveis na simulação e, portanto, no parágrafo seguinte encontra-se as respetivas definições.

Em (Shannon, 1998) o sistema é um grupo de elementos inter-relacionados que cooperam entre si para atingir um objetivo. O modelo é a representação de um grupo de objetos ou ideias. Em (Maria, 1997) defende que um modelo é uma representação da construção e do funcionamento de um sistema, é uma simplificação do sistema.

Numa simulação, inicialmente um sistema real é modelado os comportamentos de forma fidedigna, e posteriormente o modelo é simulado, após são recolhidos os resultados da simulação para as análises. A partir desses resultados são formulados conclusões e a tomada de decisão para alteração do sistema e o ciclo repete até garantir uma melhoria contínua do sistema. Na Figura 2.4 observa-se a interação entre mundo real e a simulação.

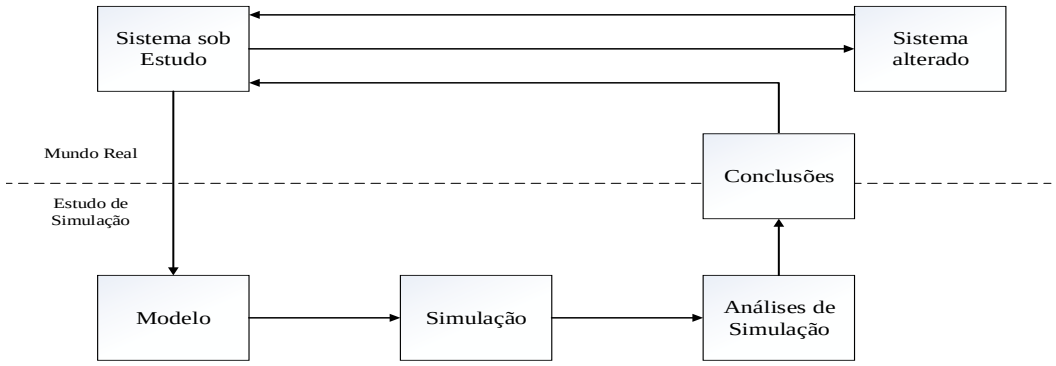


Figura 2.4 Esquema de um estudo de simulação adaptado de (Maria, 1997)

2.3.2 Modelos de Simulação

Neste subcapítulo são apresentados três modelos utilizados na simulação, nomeadamente Simulação de Evento Discreto ou (DES - *Discrete Event Simulation*), Dinâmica de Sistema ou (SD - *System Dynamic*) e Simulação Baseada em Agente ou (ABS - *Agent Based Simulation*).

2.3.2.1 Dinâmica de Sistema

O SD foi desenvolvido por Jay Forrester na década de 50. O SD foi considerado como estudo das características de *feedback* da informação de atividade industrial para mostrar como a estrutura organizacional, amplificação e o tempo de atraso (nas decisões e ações) interagem para influenciar o sucesso de uma empresa (Forrester, 1958). O SD é uma forma de compreender o comportamento de um sistema complexo ao longo do tempo. Este método é focado em *feedback loop* para ver a causa e a razão que acontece com algum evento (Sumari & Ibrahim, 2013). Este modelo é conhecido como um modelo que tem mudança no seu estado e pode ser visto através do uso de séries de estoques e fluxos (Sumari & Ibrahim, 2013).

O SD é em geral uma visão macroscópica de um sistema para explorar a estrutura do sistema afeta o comportamento do sistema (Morgan, Howick, & Belton, 2017). Na Tabela 2.1 encontra-se alguns softwares e packages que suportam este tipo de comportamentos.

Tabela 2.1 Software/Package para o SD

Software/package SD	**Contacto**
Analytica	http://www.lumina.com
AnyLogic	http://anylogic.com
ASCEND	http://ascend4.org
Berkeley Madonna	http://berkeleymadonna.com
Consideo	http://www.consideo-modeler.de
Dynaplan Smia	http://www.dynaplan.com
Forio Simulations	http://www.forio.com
GoldSim	http://www.goldsim.com
Insight Maker	http://www.insightmaker.com
Powersim	http://www.powersim.com
RecurDyn	http://www.recurdyn.de
Simantics System Dynamics	http://sysdyn.simantics.org
Simile	http://www.simulistics.com
Simulink	http://www.mathworks.com/products/simulink
Vensim	http://vensim.com
Ventity	http://ventity.biz
VisSim	http://www.vissim.com
Wolfram System Modeler	http://www.wolfram.com/system-modeler

2.3.2.2 Simulação de Eventos discretos

O DES foi introduzido na década 60 por Geoffrey Gordon. Este modelo foi desenvolvido para, a GPSS (*General Purpose Simulation System*) (Gordon, 1961). O DES é constituído por quatro tipos de conceitos distintos principais, nomeadamente os eventos, entidades, atributos e recursos. O evento é algo que acontece num determinado momento num ambiente que pode afetar recursos e/ou entidades. As entidades são objetos que possuem atributos e consumem recursos enquanto está à decorrer um evento. Este consumo de energia não é afetado pelo comportamento ao nível individual. Os atributos são característicos únicos de uma entidade, que podem ou não mudar ao longo do tempo. Os recursos são objetos que fornecem um serviço a uma entidade.

As filas são outro conceito importante do DES e ocorrem quando várias entidades competirem por um específico recurso (Marshall et al., 2015).

No modelo DES a dinâmica do sistema é trigonométrica, gerado por eventos, permitindo aos utilizadores simular os eventos individuais no sistema. Este modelo é muitas vezes usado para o sistema a nível operacional, onde a interação individual e a variação das experiências das

entidades ao longo do tempo é importante (Morgan et al., 2017). Na Tabela 2.2 apresenta-se algumas ferramentas de simulação que suportam este tipo de comportamentos.

Tabela 2.2 Software do DES

Software DES	**Companhia**
Arena	Rockwell Software
AutoMod	Brooks Automation (AutoSimulations)
DE3	BYG Systems
Dosimis3	Simulations Dientleistungs Zentrum GmbH
Enterrise Dynamics (Taylor ED)	Incontrol Enterprise Dynamics
Extend	Imagine That
Factor/Aim	Symix (Pritsker)
FlexSim (Taylor II)	Flexsim Software Products, Inc
GPSS/H	Wolverine Software
G2 Rethink	Gensym
Micro Saint	Micro Analysis and Design
MMS	nHance Technologies
ProModel	Production Modeling of Utah
Quest	Delmia Corp.(Deneb Robotics)
Schedula	Codework
Ses/Workbench	HyPerformix
Show Flow	Incontrol Simulation Software B.V
SIMAS II	CIMPACT Sarl
SimBax	AICOS Technologies AG
SimFlex	Flextronics
Em-Plant (Simple++)	UGS (Technomatrix)
Simprocess	CACI Products Company
SIMUL8	SIMUL8 Corporation Products
SLX	Wolverine Software
Spar	Clockwork Solutions
Witness	Lanner Group

2.3.2.3 Simulação Baseada em Agente

O ABS é modelo de simulação para sistemas dinâmicos, adaptativos, e autónomos (Gunal, 2012). O núcleo do ABS é constituído por agentes inteligentes, autónomos que interagem entre si. Estes agentes usam raciocínio dedutivo e indutivo sobre o sistema e adaptam-se aos ambientes em que se encontram. Os comportamentos dos agentes é obitido de acordo com as regras de decisão definidas e as ações são baseadas no estado atual do ambiente. Os agentes

podem ter objetivos explícitos para maximizar ou minimizar, aprender e se adaptar de acordo com a experiência.

O ABS é descentralizado e o comportamento global do sistema é dado pelo comportamento individual de cada agente (Borshchev & Filippov, 2004). Um modelo ABS possui três elementos importantes (Gunal, 2012):

- **Agentes -** possuem os atributos que podem ser dinâmicos ou estáticos (variáveis) e comportamentos que são ações condicionais ou incondicionais (métodos).
- **Interações -** define as relações entre agentes.
- **Ambiente -** são fatores que afetam os agentes e as suas interações.

Na Tabela 2.3 encontra-se algumas plataformas que são capazes de suportar este tipo de comportamentos do ABS.

Tabela 2.3 Plataforma para ABS

Plataforma ABS	**Contacto**
A3/AAA	http://joram.ow2.org/
AgentBuilder	http://www.agentbuilder.com
AgentSheets	http://www.agentsheets.com/
AnyLogic	http://www.anylogic.com
AgentService	http://www.agentservice.it/
Cormas	http://cormas.cirad.fr
FLAME	http://flame.ac.uk/
FLAME GPU	http://www.flamegpu.com/
JABM	http://jabm.sourceforge.net/
JADE1	http://jade.tilab.com/
Janus	http://www.janusproject.io/
JAS-mine	http://www.jas-mine.net/
JASA	http://jasa.sourceforge.net/
Jason	http://jason.sourceforge.net/wp/
JESS	http://herzberg.ca.sandia.gov/jess/
MaDKit	http://www.madkit.net/madkit/
MASON	http://cs.gmu.edu/~eclab/projects/mason/
NetLogo	https://ccl.northwestern.edu/netlogo/
Sugarscape	http://sugarscape.sourceforge.net/

A comparação entre SD, DES e ABS pode-se ser encontrado em (Borshchev & Filippov, 2004), (Marshall et al., 2015), (Gunal, 2012) e (Sumari & Ibrahim, 2013).

2.3.3 Ferramentas de Simulação

Em seguida apresenta-se quatros tipos de simuladores distintos utilizados num sistema de manufatura.

2.3.3.1 ARENA

A Arena surgiu em 1993 devido à integração do SIMAN (é uma linguagem de simulação) e CINEMA (é um pacote integrado ao SIMAN que permite uma apresentação animada do sistema) num único ambiente de simulação (Rockwell Software, 2017). A linguagem SIMAN através da Arena passou a ser representada em formato gráfico tornando-se mais simples e intuitiva. Esta ferramenta permite simular eventos discretos ou contínua. A sua aplicação é fácil

[1] Esta plataforma será usada nesta dissertação para a implementação dos agentes.

uma vez que não é necessário escrever o código de programação para a criação do modelo. O modelo é criado através da tecnologia de *drag and drop* dos blocos já modelados no software. A ARENA possui a capacidade de animação 2D e 3D, permitindo a interações com outros softwares como Visual Basic, Excel, etc (Gebus et al., 2004). É possível escolher o nível de complexidade usando recursos básicos ou recursos mais específicos permitindo a criação de ferramentas personalizadas no programa.

2.3.3.2 SIMIO

O SIMIO (*Simulation Modeling framework based on Intelligent Objects*) foi desenvolvido em 2007 (Vik, Dias, Pereira, Oliveira, & Abreu, 2010), esta ferramenta permite construir e executar modelos dinâmicos de um sistema (Simio, 2017). O SIMIO baseia-se nos objetos inteligentes usando o conceito de orientação a objetos. Os objetos representam a componente física do sistema (exemplo: passadeira, robôs, carros, barcos, mota, etc.). Estes objetos podem ser reutilizados em vários projetos de simulação nos quais podem ser facilmente armazenados e partilhados na biblioteca do SIMIO (Pegden, 2007; Sturrock & Pegden, 2010).

A animação do comportamento do seu sistema pode ser efetuada em 3D ao longo do tempo. O utilizador pode usar objetos pré-definidos da biblioteca ou construir o seu próprio objeto inteligente para a construção de um modelo hierárquico (Sturrock & Pegden, 2010). No entanto, o processo da criação do objeto é muito simples e não é necessário escrever código de programação. O Simio e Arena têm várias características em comum e a descrição detalhada pode ser encontrada em (Pegden, 2007; Simio, 2017; Sturrock & Pegden, 2010; Vik et al., 2010).

2.3.3.3 Flexsim

O Flexsim foi fundado em 1993 por Bill Nordgren, Roger Hullinger e Cliff King, originalmente com o nome de F&amp H Simulation (Flexsim, 2017). A primeira versão do Flexsim 1.0 foi lançado em 2003, para simulação de evento discretos. Flexsim é software de simulação 3D, orientado a objeto, tecnologia do *drag and drop* e OpenGL. Os utilizadores podem personalizar ou criar os seus próprios objetos e guardar numa biblioteca para uma futura reutilização na criação de um novo modelo. O Flexsim é totalmente integrado a sistema CAD e, é compatível com a indústria 4.0.

2.3.3.4 V-REP

O V-REP é um simulador versátil e escalável para a simulação 3D relativamente num curto período de tempo. Foi desemvolvida em 2010 e é um dos simuladores mais utilizados na área robótica para a educação (Peralta, Fabregas, Farias, Vargas, & Dormido, 2016).

Esta ferramenta contém o *integrated development environment (IDE)* que baseia numa arquitetura de controlo distribuído. Nesta arquitetura cada objeto ou modelo pode ser controlado individualmente através de um *script* incorporado, um plug-in, um nó ROS (*Robot Operating System*), um cliente remoto API ou uma solução personalizada. Estas condições fazem com que

o V-REP seja muito versátil e ideal para aplicação multi-robô. O V-REP pode ser programado em diferentes linguagens como por exemplo: C / C++, Python, Java, Lua, Matlab, Octave ou Urbi (Coppelia-Robotics, 2016).

No V-REP encontra-se implementado vários modelos de robôs, equipamentos, infraestruturas, componentes e outros exemplares que podem ser adicionados na plataforma.

2.4 Conclusões Gerais

Neste capítulo foi possível verificar que os sistemas de manufatura evoluíram ao longo do tempo, desde primórdios de produção até o presente. As abordagens tradicionais, como sistemas centralizados e baseadas em regras, tornaram-se obsoletos e inadequados para responder as necessidades do mercado com elevadas flutuações. Portanto, surgiram os vários paradigmas de produções. Alguns dos paradigmas são FMS, RMS, BMS, HMS e EPS. Estes paradigmas permitem o sistema de manufatura ser flexível, reconfigurável, ágil, sustentável e a diagnosticável. Estas características são importantes para dar resposta as turbulências dos mercados. Estes paradigmas permitem estudar sistemas distribuídos, descentralizados e auto-organizados.

O MAS permite melhorar as propriedades importantes como por exemplo a modularidade, escalabilidade, interoperabilidade, versatilidade e autonomia necessária para manter os sistemas adaptáveis as novas tecnologias ou alterações nas demandas. Este sistema funciona de forma descentralizados, onde as incertezas ou eventuais conflitos podem ser resolvidos através de comunicação, colaboração e cooperação entre agentes.

Por fim os simuladores são amplamente utilizados como objetivo de avaliar, testar, analisar comportamentos dos sistemas reais. Estes simuladores permitem estudar modelos complexos, intratáveis analiticamente.

Na Tabela 2.4 apresenta-se algumas características relevantes dos simuladores descritos.

Tabela 2.4 Comparações entre diferentes simuladores

		ARENA	SIMIO	Flexsim	V-REP
Simulação por evento discreto		✓	✓	✓	×
Flexibilidade de linguagem de programação		×	×	×	✓
Animação em 3D		✓	✓	✓	✓
Versão livre para estudante		×	×	✓	✓
Integração com componentes reais		×	×	×	✓
Tecnologia Drag and Drop		✓	✓	✓	✓
Sistema operativo	*Windows*	✓	✓	✓	✓
	Mac	×	×	×	✓
	Linux	×	×	×	✓

3

ARQUITETURA

Neste capítulo descreve-se a arquitetura geral proposto para a implementação deste projeto. Esta arquitetura baseia-se no paradigma EPS e é composto por agentes inteligentes, auto-organizáveis, autónomos que cooperam entre si para alcançar os objetivos pretendidos. Estas características proporcionam uma elevada versatilidade, dinamismo, flexibilidade e interoperabilidade nos sistemas distribuídos. A arquitetura encontra-se dividida em três camadas: i) a camada de execução; ii) camada de integração; iii) camada de simulação.

3.1 Arquitetura geral do Sistema

Embora a arquitetura é composta por três camadas, o núcleo incide principalmente na camada de integração. Esta camada é responsável pela interligação da camada de execução e a camada de simulação. Portanto, permite uma comunicação bilateral entre os clientes (agentes) da camada de execução e os servidores (simuladores) da camada de simulação. A camada de execução é o cérebro do sistema, onde encontra-se definida toda a inteligência do funcionamento da linha de produção. A camada de simulação permite simplesmente simular a linha de produção, ou seja, é o corpo do sistema. Esta camada realiza ações de acordo com os pedidos dos agentes e, portanto, não possui nenhum controlo ou decisão sobre o sistema.

A estrutura da arquitetura é composta por três camadas, onde cada camada é constituída por várias entidades diferentes e independentes. No entanto, para garantir as características de um sistema evolutivo é necessário existir uma interação entre as camadas. Na Figura 3.1 apresenta-se arquitetura geral inserido neste projeto.

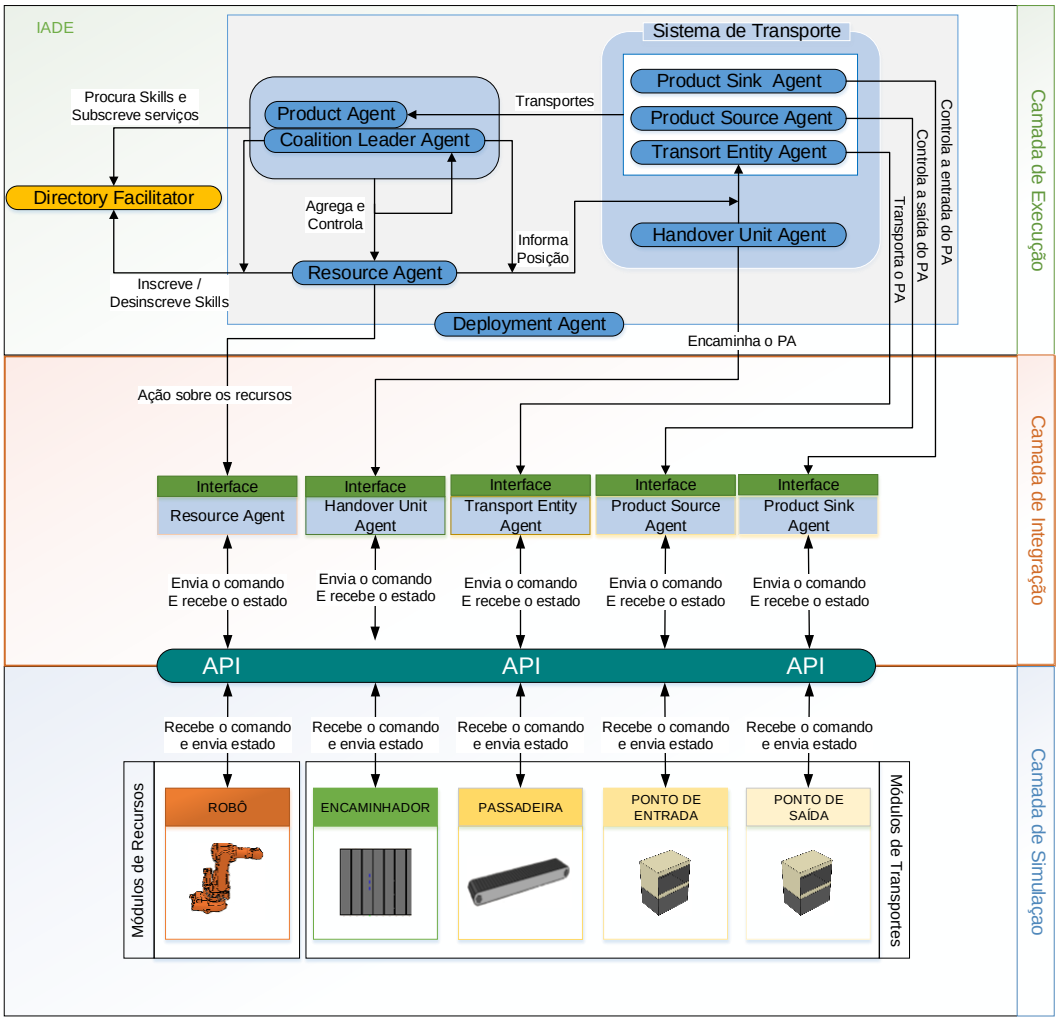


Figura 3.1- Arquitetura geral do sistema

3.2 Camada de Execução

A camada de execução baseia-se na arquitetura do IADE e é desenvolvida no projeto IDEAS na plataforma JADE. Neste projeto, esta camada é responsável pela execução do sistema. Qualquer decisão relativamente ao plano de produção é feita na camada de execução. Esta camada é composta por nove agentes autónomos, que estão distribuídos em três grupos: suporte (dois agentes); execução (três agentes) e transporte (quatro agentes).

Na Tabela 3.1 encontra-se as funções de cada agente e os seus respetivos grupos.

Tabela 3.1 Descrições das funções dos agentes mecatrónicos e os seus respetivos grupos

Grupo	Nome	Descrição
Suporte	DeploymentAgent (DA)	Permite ao utilizador lançar os agentes no sistema. Esses agentes são os de execução e dos transporte.
	DirectoryFacilitator (DF)	Responsável por inscrever ou desinscrever os agentes e as suas habilidades na plataforma. Permite os agentes inscritos consultar as habilidades ou serviços disponíveis de quaisquer agentes, para a execução de uma determinada tarefa.
Execução	ResourceAgent (RA)	Responsável por abstrair equipamentos físicos de um sistema de produção, como neste caso o robô e a garra. Tem como funçao a execuçao das habilidades atómicas.
	CoalitionLeaderAgent (CLA)	Responsável pela coordenaçao dos seus agentes. Suporta a composição de várias habilidades fornecidas por entidades de mais baixo nível (RA e CLA de nível inferior) e dá origem a uma nova habilidade de grau superior.
	ProductAgent (PA)	É o agente de mais alto nível do sistema. Cada produto é abstraído por um agente deste tipo, no qual tem a capacidade de negociar e decidir o melhor agente para um determinado tarefa como também o local para sua execução. Os PA's são independentes e, portanto, cada um gere o seu próprio plano de execução.
Transporte	ProductSourceAgent (PSoA)	É responsável por gerir a entrada do produto no sistema e lançar o PA no sistema. A PSoA associa cada produto ao um PA
	ProductSinkAgent (PSiA)	É responsável por gerir a saída do produto do sistema, isto é, quando um PA termina o seu plano de produção, o produto será removido da plataforma.
	TransportEntityAgent (TEA)	É responsável por abstrair as entidades de transportes no sistema como por exemplo a passadeira. Permite transportar o produto entre encaminhador, para uma determinada estação. Quando o PA alcança o destino o TEA informa-o da sua chegada.
	HandoverUnitAgent (HUA)	É responsável por encaminhar os PA's entre os TEA's, ou seja HUA verifica o destino do PA e consulta na sua tabela de vizinhos (que contém TEA's) e decide qual será o melhor TEA para o destino.

3.2.1 Definição de Habilidade

Os agentes genéricos que se encontrem na pilha do IADE expressam as suas funcionalidades e dinamismo em habilidades ou skills (Ribeiro, Rocha, & Barata, 2012). As habilidades de cada agente são posteriormente disponibilizadas e registadas no DF, onde podem ser consultadas por outros agentes que se encontrem registado no DF. Estas habilidades podem ser executadas mediante os pedidos efetuados pelos agentes.

As habilidades podem ser classificadas da seguinte forma:

- **Atomic Skill (ASk)** é um mecanismo que permite associar informação de um skill ao um controlador específico. Este mecanismo encontra-se integrado no RA
- **Complex Skill (CSk)** é responsável pela execução do processo de alta complexidade. Este mecanismo pode ser constituído por um conjunto de ASk e/ou CSk, e encontra-se integrado na CLA.

Na Figura 3.2 apresenta-se um exemplo da composição do CSk que se pretende ser implementado neste trabalho. Neste exemplo, Figura 3.2, encontra-se as skills que serão executadas sequencialmente da esquerda para a direita. Esta sequência inicia-se na ASk1 e termina na CSk2. A CSk1 termina de ser executada quando ASk2, ASk3 e ASk4 terminar as suas execuções. De imediato inicia-se a execução da ASk5 e de seguida inicia-se a execução da CSk2. A CSK2 é composta por um conjunto de skills como por exemplo CSk1, ASk6 e ASk7. É possível observar que a CSk1 está a ser reutilizada para definir um novo skill, CSk2, que segue a mesma lógica de execução da CSk1. É de salientar que só as ASk's podem interagir diretamente com o hardware.

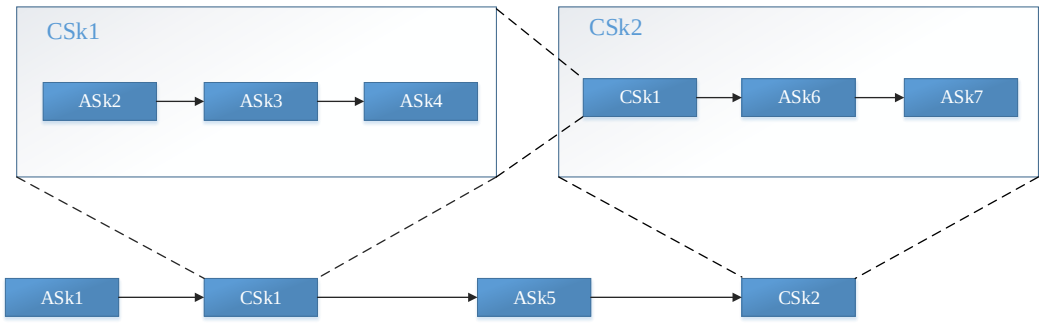


Figura 3.2 Exemplo da composição da CSk

3.2.2 Definição da Área

A arquitetura IADE permite a definição das áreas[2]. Esta definição é útil para evitar situação como por exemplo, um CLA quer alocar um RA para uma determinada execução da habilidade, e este encontra-se fora do alcance físico do CLA, portanto, o CLA não pode alocar RA fora do alcance físico mesmo que abstração da RA produzisse a mesma habilidade. A utilização das áreas delimita o escopo de ação de um agente e também da negociação do CLA, isto é, impede os agentes de alocar habilidades de outros agentes pertencentes as áreas distintas.

Para ilustrar a definição das áreas apresenta-se na Figura 3.3, um exemplo com duas áreas distintas (Área1 e Área2). Estas áreas contêm o CLA1 e o CLA2, e os seus recursos (RA). A CSk do CLA1 será constituída somente por RAs (verdes) porque estão localizados fisicamente na mesma área. A mesma regra será aplicada ao CLA2 que neste caso a sua CSk será constituída somente por RAs (azuis). A definição das áreas neste exemplo impede que o CLA1 e o CLA2 troquem de recursos (RA), com essa restrição deixa o sistema mais organizado.

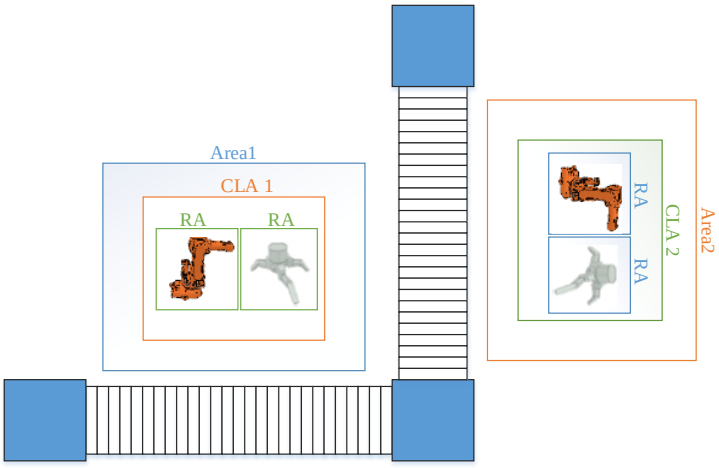


Figura 3.3 Exemplo da definição das áreas

3.2.3 Topologia do Sistema de Transporte com os Agentes

Na Figura 3.4 apresenta-se todo o mapeamento entre a topologia do sistema e os agentes. Este mapeamento contém todos os componentes físicos e lógicos necessários para execução adequado da linha de produção.

[2] É usada para definir a partição do sistema nos quais os agentes são capazes de inter-operar fisicamente entre eles em cada partição (Ribeiro et al., 2015).

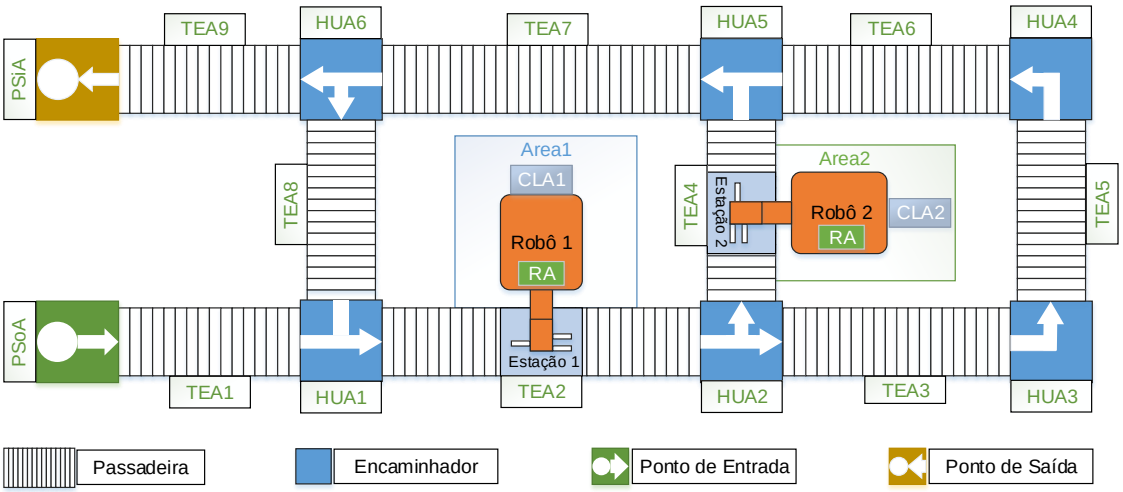


Figura 3.4 Exemplo da Topologia de um Sistema

O ponto de entrada e de saída são abstraídos pelo PSoA e PSiA, respetivamente. O ponto de entrada é responsável por lançar os produtos diretamente no sistema físico, sendo assim o PA terá o seu primeiro contato com o sistema. O ponto de saída é responsável por remover o produto do sistema.

A passadeira é responsável por transportar o produto e acoplar os recursos (ex: TEA2 e TEA4). No entanto, cada TEA é uma abstração da passadeira (Figura 3.4). Cada passadeira move no sentido das setas apresentadas no encaminhador adjacente à passadeira. Este sistema contém nove TEA e as suas respetivas passadeiras.

O encaminhador é abstraído pelo HUA, tem como objetivo encaminhar o produto para as passadeiras. Existe seis encaminhadores neste sistema e cada um tem os seus vizinhos de entrada e saída. Os vizinhos de entrada fornecem os produtos ao encaminhador e os vizinhos de saída recebem os produtos do encaminhador.

No sistema apresentado na Figura 3.4 existe duas áreas distintas. Cada área contém um robô (com uma garra integrado) abstraído pelo RA e coordenado pelo CLA. A estação está ancorada na TEA2 e TEA4, onde os recursos executam as suas habilidades.

3.2.4 Agentes de Execução

Os agentes de execução (RA, CLA e PA) inseridos na camada de execução têm como objetivo garantir o funcionamento dinâmico, eficaz e inteligente do sistema de manufatura. As interações entre RA, CLA e PA consistem em várias situações ou casos, desde a negociação até ao pedido de execução de uma determinada da tarefa. Estas interações permitem garantir uma melhor decisão perante os objetivos pretendidos.

3.2.4.1 Resource Agent

O RA é a unidade mais básica da arquitetura IADE que interagi diretamente com os controladores ou módulos do hardware do sistema de produção. Nesta arquitetura, RA é responsável por abstrair equipamentos ou componentes físicos inseridos no ambiente de manufatura. No caso de estudo considerado nesta dissertação particular, RA é abstraído pelos módulos de recursos, como por exemplo robôs, garras, postos humanos etc. O controlo dos equipamentos são asseguradas através das habilidades de cada agente com objetivo de traduzir as suas habilidades em código nativo dos seus respetivos equipamentos, permitindo uma eventual ação física sobre os equipamentos.

O RA ao ser lançado deve inscrever todas as suas habilidades nos serviços das páginas amarelas, assim permitem aos outros agentes solicitar-lhe para uma eventual execução das habilidades. Além da inscrição nas páginas amarelas, o RA também deve informar aos transportes das suas localizações físicas e onde serão executadas as suas habilidades. Na Figura 3.5 ilustra-se as primeiras interações com outros agentes.

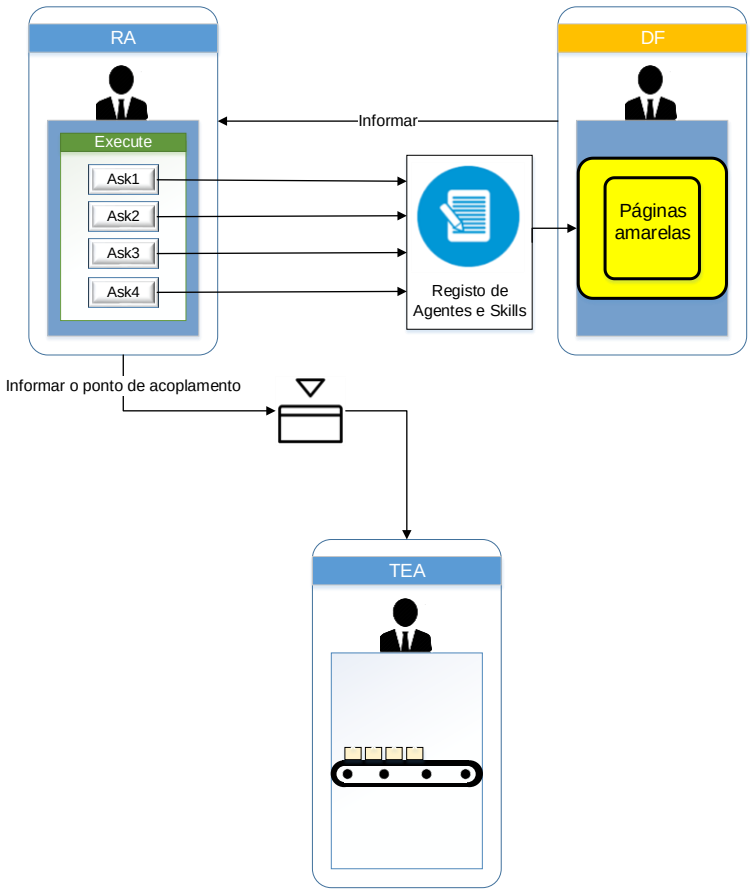


Figura 3.5 Interação dos recursos com os agentes

3.2.4.2 Coalidition Leader Agent

Nesta arquitetura pretende-se que cada CLA coordena os seus agentes associados e os processos em cada área em que pertencem a fim de garantir organização e gestão das atividades dos agentes no sistema. Isto é, a implementação do CLA garante que um sistema fica mais adaptável e escalável, garantindo assim maior flexibilidade na linha de produção.

O CLA, ao contrário do RA, não interage diretamente com os módulos do hardware mas sim coordena a execução dos RA's ou outros CLA's de níveis mais baixo. No entanto o CLA é um puro agente de software. O CLA suporta a composição de várias habilidades oferecidas por entidades de mais baixo nível (RA e/ou CLA) dando origem à uma nova habilidade de grau superior (CSk). O CLA é normalmente invocado para a execução sequencial e condicional das habilidades fornecidas pelos membros dos agentes que o compõe.

Na Figura 3.6 pode observar que o CLA pode ter vários níveis ou granularidade consoante a complexidade do sistema. Neste exemplo tem três CLA's, onde CLA1 é de nível superior e coordena CLA2 e CLA3 de nível mais baixo. CLA2 e CLA3 e estas por sua vez coordenam o RA1, RA2 e RA3, RA4. Os RA's atuam diretamente no hardware sabendo que o CLA não pode interagir com os módulos do hardware. Todos os agentes da camada azul são de níveis mais baixo e são coordenados pelos CLA's que se encontra na camada laranja e verde. Estes são responsáveis por todos tipos de comunicações com os agentes de níveis superiores (CLA ou PA). Para a negociação ou execução de skills dos seus agentes associados uma vez que ele gera e lidera as suas operações.

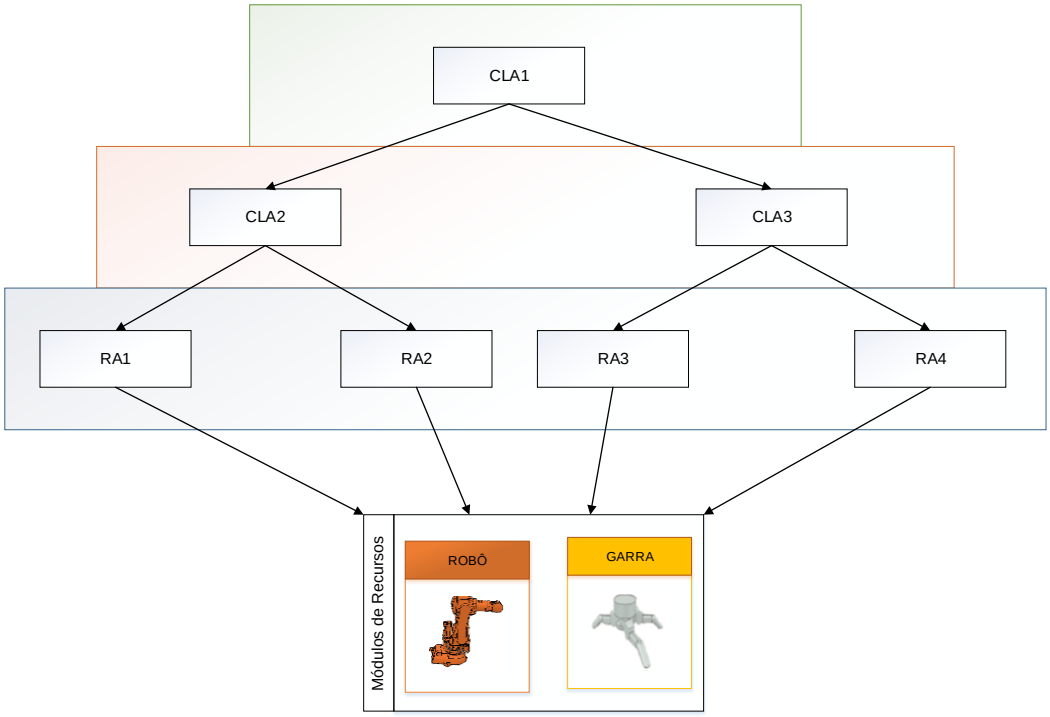


Figura 3.6 Hierarquia do CLA

O CLA ao ser lançado deve inscrever as suas habilidades nas páginas amarelas e também deve informar aos transportes das suas localizações físicas para execuções das habilidades. O CLA traz dinamismo e flexibilidade à linha de produção, uma vez que muitos recursos não possuem capacidades para negociações ou raciocínio por causa das suas pequenas capacidades computacional. Portanto estes recursos coordenados pelo CLA maximizam a interoperabilidade e autonomia do sistema. Na Figura 3.7 é possível observar a rotina que o CLA pode desempenhar durante a execução do sistema.

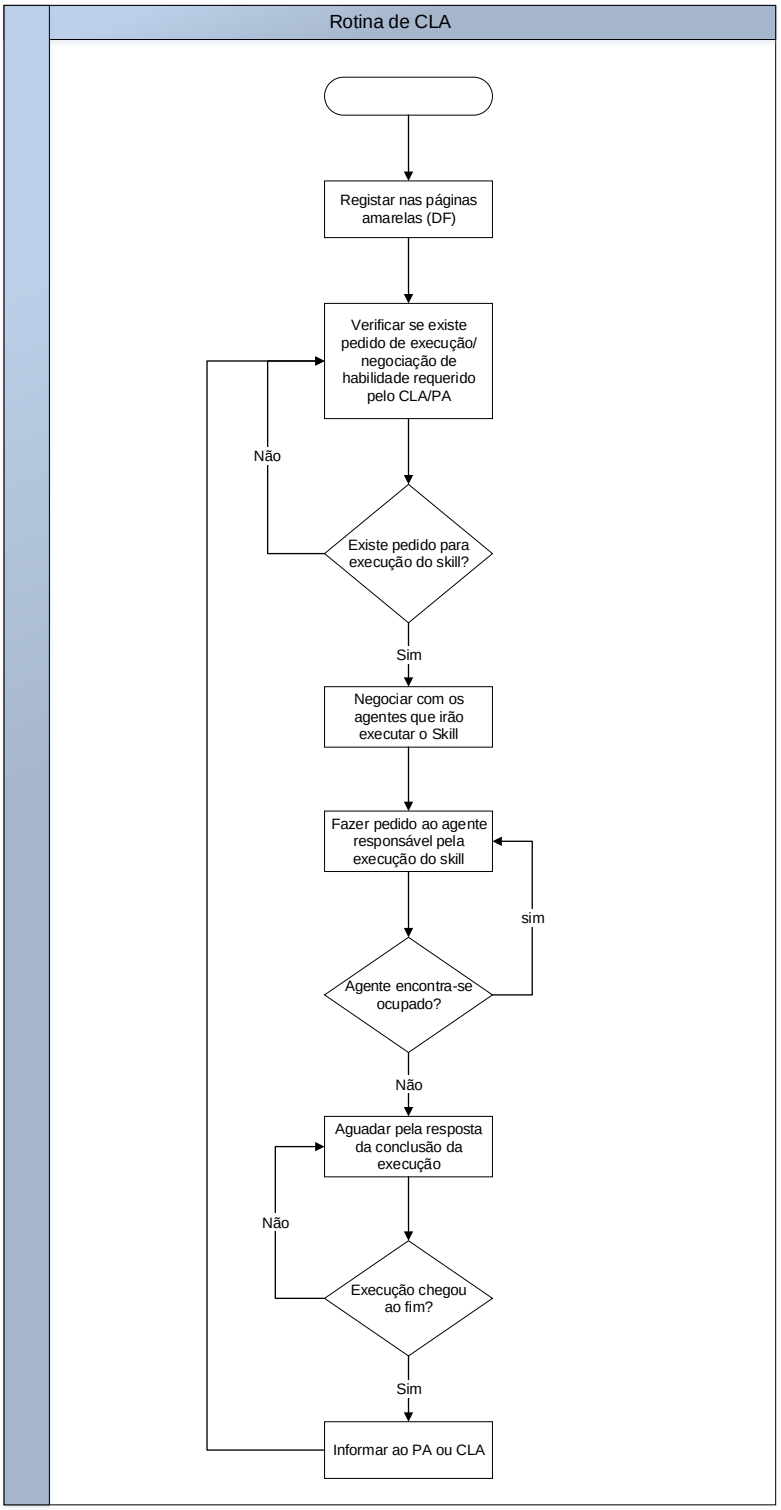


Figura 3.7 Diagrama de atividades do CLA

3.2.4.3 Product Agent

O PA é a entidade de mais alto nível do sistema, portanto, é uma extensão do CLA a um nível superior. Cada produto no sistema é abstraído pelo PA, no qual possui a capacidade de

negociar e decidir o melhor agente e o local para executar as tarefas. Os PA's são independentes e não interagem entre si, uma vez que cada PA gere os seus próprios planos de execução. O PA não troca informação com os produtos, isto é, não interage diretamente com os produtos, uma vez que o objetivo do PA é controlar adequadamente os comportamentos lógicos dos produtos na linha de montagem. Os produtos comportam-se de acordo com as decisões tomadas dos seus agentes, que por sua vez age de forma diferente devido à lista de tarefas que cada um compõe. Isto é, numa linha de produção pode existir vários tipos de produtos diferentes, e essas diferenças estão presente na lista das tarefas dos PA's. A interoperabilidade entre agentes permite ao PA decidir adequadamente o comportamento que cada produto pode ter na produção e este comportamento rege-se de acordo com a lista de tarefas.

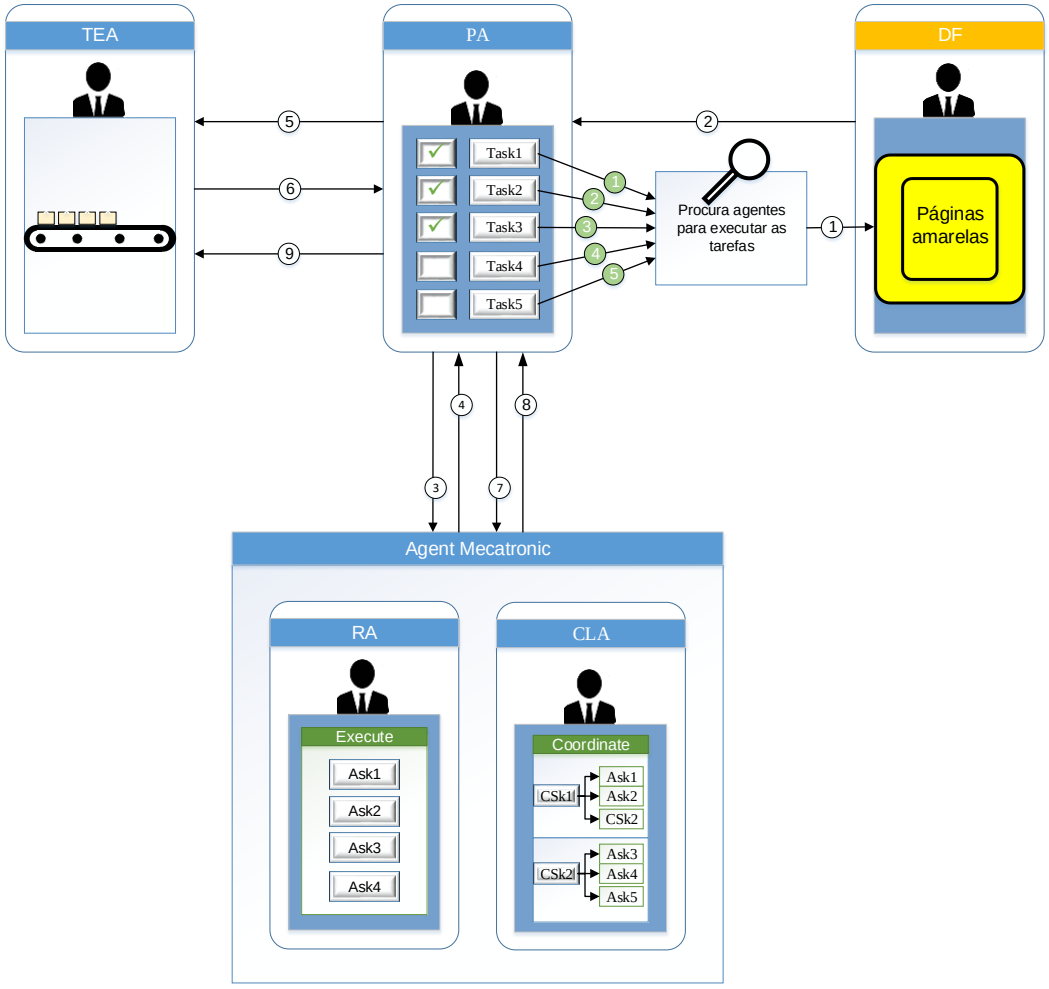


Figura 3.8 Interações sequenciais do PA

Os PA's são os agentes mais importantes da arquitetura, todos os agentes operam dos PA's em torno dele, eles cooperando entre si para satisfazer adequadamente os pedidos dos PA's. Portanto, os PA's adaptam as linhas de produção de acordo com objetivo do produto. Uma vez que, as linhas de montagem são configuradas e montadas consoantes as necessidades da

produção. A adaptação dinâmica dos PA's é necessário para o ambiente em que este se encontram, têm que possuir capacidade de comunicar e interagir com outros agentes inseridos no sistema e organizar-se adequadamente para ponderar melhor a decisão perante as necessidades da produção.

Na Figura 3.8 estão representados possíveis interações e comunicações do PA durante a sua execução. Essas comunicações são compostas por várias fases. A primeira fase o PA procura agentes disponíveis nas páginas amarelas para a execução das suas tarefas (1 e 2). Na segunda fase, o PA negocia com esses agentes mecatrónicos (RA ou CLA) para futura execução das tarefas (3 e 4). Na terceira fase o PA decide o destino onde será executado e posteriormente pede ao TEA para transportar o produto ao destino e o PA é informado pelo TEA aquando este chega ao destino pretendido (5 e 6) e por fim, a última fase o PA peça ao agente anteriormente negociado para a execução das tarefas (7 e 8). Este é o ponto culminante na execução das tarefas.

O número à verde na Figura 3.8, indicam que cada tarefa é executada sequencialmente, isto é, para executar qualquer tarefa é necessário repetir todo o processo descrito na sequência de 1 a 9. No fim da execução o PA pede ao TEA (9) para transportar o produto para a estação terminal.

Na Figura 3.9 apresenta-se uma rotina comportamental do PA.

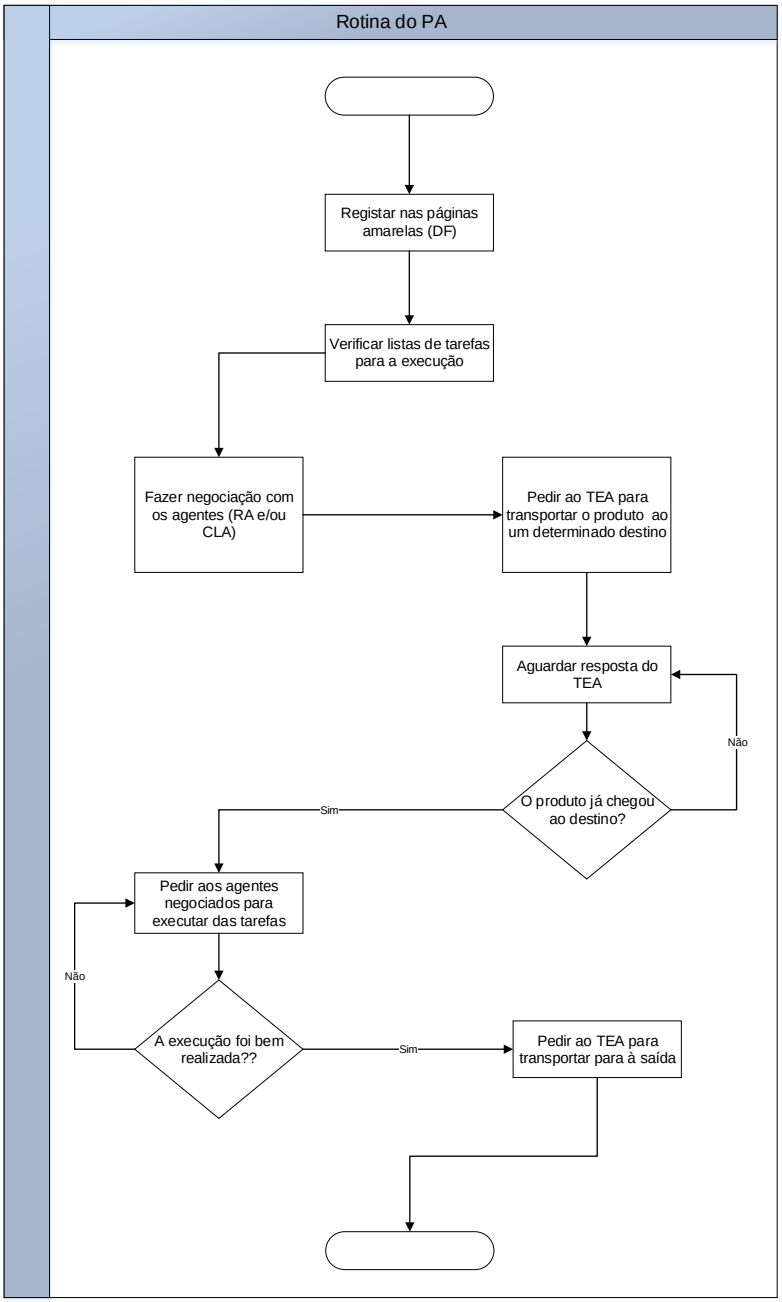


Figura 3.9 Diagrama de atividades do PA

3.2.5 Agentes de Transportes

Os agentes dos sistemas de transportes são responsáveis pela mobilidade dos produtos na linha de produção. Esta mobilidade tem de ser eficaz e flexível para garantir dinamismo ao sistema. Além disso, este sistema tem que ter capacidade de transferir diferentes produtos contido nos respetivos destinos. Para tal existe na arquitetura quatro tipos de agentes que são capazes de produzir estas características nomeadamente PSoA, PSiA, HUA e TEA. As duas últimas características são entidades principais do sistema de transporte.

3.2.5.1 Product Source Agent

O PSoA é responsável por gerir a entrada do agente do tipo PA no sistema de produção e de seguida associar ao produto o seu e respetivo agente PA. O PSoA também atribui o ID a cada PA e ao produto físico que se encontra no sistema de produção, assim permite a identificação de cada produto. O PSoA pode ter mais de que um PA para lançar ao sistema, sendo assim, tem que gerir suas entradas na linha de produção para evitar colisão ou acumulação dos produtos no ponto de entrada. Essa gerência adequa-se mais ao nível físico do que lógico, uma vez que o lógico os produtos são adicionados numa lista dinâmica e o físico são adicionados diretamente ao sistema de produção.

Um novo produto só pode ser introduzido no sistema a partir do momento em que exista espaço no ponto de entrada, e posteriormente será alocado na passadeira. Portanto, o PSoA é responsável pela primeira decisão do sistema de transportes.

3.2.5.2 Product Sinck Agent

O PSiA, ao contrário do PSoA, é responsável por gerir a saída do agente PA do sistema, tanto ao nível lógico como físico. Quando um PA termina o seu plano de produção o produto é encaminhado ao ponto de saída. O ponto de saída ao receber um produto o PSiA informa ao PA que a execução chegou ao fim, sendo assim o PA remove-se da plataforma, e por fim o produto é removido do sistema de produção.

3.2.5.3 Handover Unit Agent

O HUA é um dos agentes mais importante no sistema de transporte, a sua principal responsabilidade é decidir corretamente qual a passadeira que deverá transportar produto para o destino requerido. A decisão correta é obtida com auxílio da tabela de encaminhamento. Esta tabela contém os seus vizinhos de entrada e de saída (passadeiras abstraídas pelo TEA's) e informações contidas nas etiquetas de cada produto como nome, origem, destino e outras informações auxiliares. Através dessas informações HUA decide qual o caminho mais curto e eficaz para transportar o produto definido pelo PA. Para garantir a decisão mais adequada o HUA verifica se o destino está conectado diretamente nos vizinhos de saída. No caso contrário, HUA procura na tabela de encaminhamento o próximo destino mais perto que possa transportar o produto alcançar o seu destino final.

O encaminhador quando recebe o produto, o seu agente (HUA) verifica qual é o destino do produto, para decidir qual é o caminho que o produto deve prosseguir. Esta decisão é obtida através das informações contida na tabela de encaminhamento (Figura 3.10).

Cada encaminhador deve estar ligado ao máximo de quatro vizinhos e deve ter pelo menos um vizinho de entrada e um de saída. Caso contrário não haverá nenhuma necessidade de o ter no sistema. Assume-se que cada encaminhador só pode ter na posse um produto de cada vez. Esta condição garante maior flexibilidade na distribuição dos produtos para os seus

respetivos destinos. Pretende-se que este encaminhador seja multidirecional, esta característica permite ao seu agente HUA decidir qual é a direção e sentido correta que deve receber e encaminhar o produto.

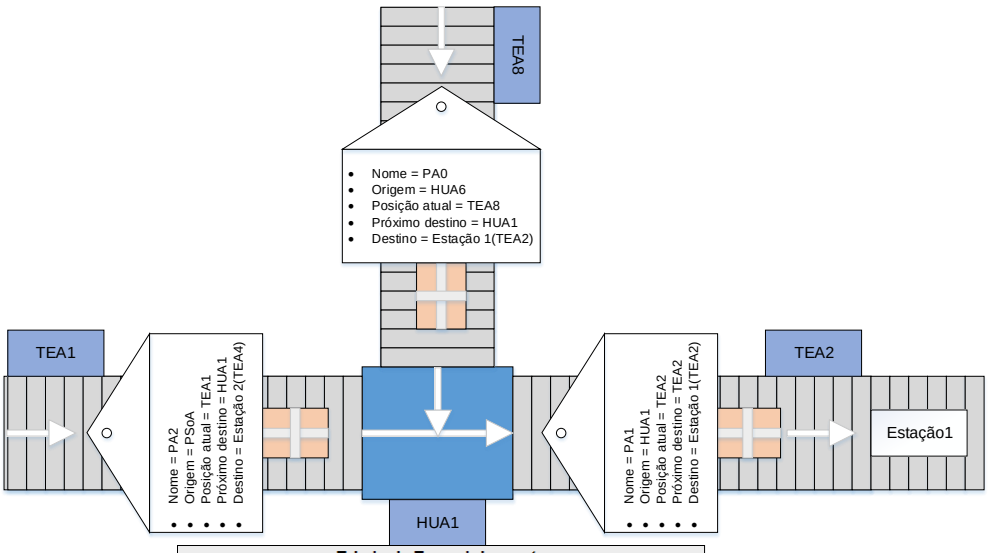


Tabela de Encaminhamento

Entrada		Saída	Destino		
			Destino	Próximo destino	Próximo salto
TEA1	TEA8	TEA2	Estação1	TEA2	TEA2
			Estação 2	HUA2	TEA2

Figura 3.10 Esquema do HUA

3.2.5.4 Transport Entity Agent

O TEA é também um dos principais agentes do sistema de transportes. Este agente abstrai a passadeira que se encontra na linha de produção. O TEA controla as entradas e as saídas dos produtos da passadeira, assim como guardar as posições de cada produto, atualizando-a quando for necessário. O TEA verifica quantos produtos a passadeira pode suportar, e este varia de acordo com o comprimento da passadeira, portanto, um sistema de transporte pode ter várias passadeiras de tamanhos diferentes. As entradas e saídas dos produtos são sequenciais, funcionando como fila, primeiro a entrar é o primeiro a sair FIFO (*First In, First Out*). Portanto a passadeira pode transportar vários produtos em simultâneos e estes são entregues um a um ao encaminhador.

A passadeira permite ancorar as estações, sendo assim o seu agente deve saber a sua posição para poder entregar os produtos para a execução das habilidades. Na Figura 3.11 apresenta-se um exemplo de uma passadeira abstraída pelo TEA2. Esta passadeira encontra-se localizado entre dois encaminhadores abstraídos pelos (HUA1 e HUA2) e nesta passadeira existe uma estação acoplada (Estação 1).

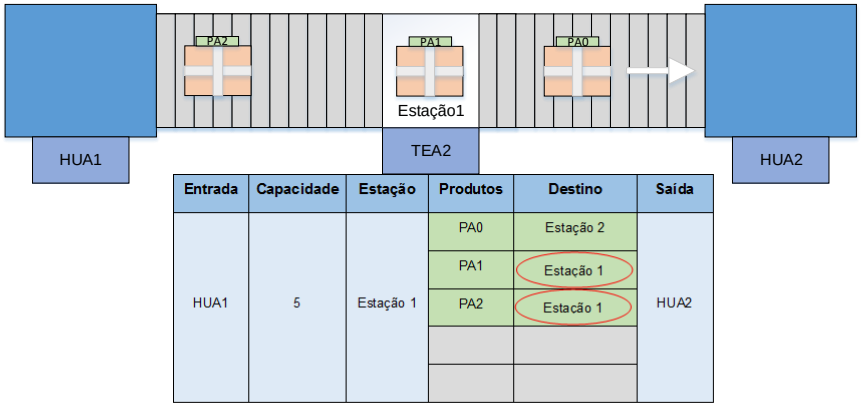


Figura 3.11 Esquema representativo do TEA

Na Figura 3.11 encontra algumas informações importantes do TEA2. Estas informações são: o seu vizinho de entrada (HUA1) e de saída (HUA2), quantidade de produtos na passadeira (três produtos como PA0, PA1 e PA2), capacidade de produto que pode ser transportado e neste caso são cinco e falta dois para lotar a passadeira. Como nesta passadeira está acoplada à estação1, através das etiquetas dos produtos o TEA2 deve-se saber o destino de cada produto, se o destino for estação1 o agente pára a passadeira para a execução. Com as informações da Figura 3.11 o agente TEA é capaz de controlar a passadeira de forma dinâmica, ágil e eficaz.

3.2.6 Interações entre Agentes

Na Tabela 3.2 estão representados todas as interações possíveis entre os agentes da arquitetura IADE implementada nesta dissertação. Essas interações proporcionam maior flexibilidade, dinamismo, agilidade e organização da linha de produção. O sistema de transporte é controlado por quatro agentes (TEA, HUA, PSoA e PSiA), como mencionado antes, estes agentes garantem o gerenciamento adequado do produto no sistema. Na Tabela 3.2 o símbolo ✓ significa que os agentes interagem e × significa que não interagem.

Tabela 3.2 Possíveis interações entre os agentes

	Executores			Sistema de Transporte				
Agentes	PA	CLA	RA	TEA	HUA	PSoA	PSiA	DF
PA	×	✓	✓	✓	×	✓	✓	✓
CLA	✓	✓	✓	✓	×	×	×	✓
RA	✓	✓	×	✓	×	×	×	✓
TEA	✓	✓	✓	×	✓	✓	✓	✓
HUA	×	×	×	✓	×	×	×	✓
PSoA	✓	×	×	✓	×	×	×	✓
PSiA	✓	×	×	✓	×	×	×	✓

Os PA's não interagem entre si devido ao facto que cada PA é independente e autónomo, com o seu próprio plano de produção. As interações de PA com CLA e RA são para a negociação ou pedido de execução de uma determinada habilidade. A interação do PA com o TEA é para o pedido de transportação do produto até ao destino ou então o TEA informa ao PA a chegada do produto à estação requerida. Por fim a interação do PA com PSoA e PSiA é para entrada e saída, respetivamente, do produto no sistema de transporte.

O CLA interage entre si para negociação ou execução de serviços requerido pelo próprio CLA ou PA. A sua interação com RA e TEA permite acoplar à estação, que se encontra ancorado no TEA. Isto é, RA's ou CLA's pede ao TEA para ancorar um determinado.

O TEA interage com HUA para encaminhar produto para outro TEA uma vez que TEA não comunica diretamente entre si, e HUA é usado como intermediário na comunicação entre eles. O HUA encaminha o produto para um dos seus vizinhos, isto é, para um TEA. A sua interação com PSoA e PSiA é para informar a entrada e a saída do produto, respetivamente.

O DF interage com qualquer agente registado na plataforma quer para inscrever/desinscrever os serviços. Na Figura 3.12 está representada as interações direta entre os agentes referidos na Tabela 3.2.

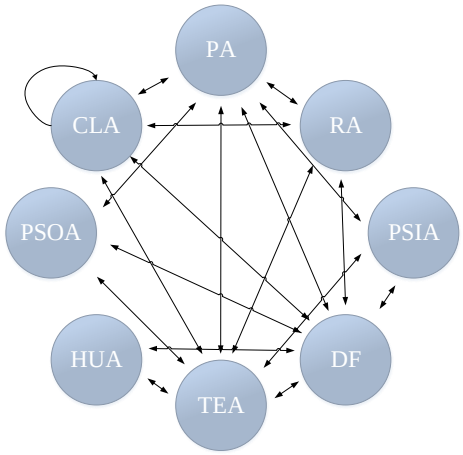


Figura 3.12 Interações diretas entre agentes

3.3 Camada de Integração

A camada de integração é o núcleo desta arquitetura. Nesta camada incide a responsabilidade de interagir a camada de execução com a camada de simulação. Isto é, a camada de integração faz uma ligação intermediária entre as camadas, no entanto envia os comandos dos clientes (agentes) para os servidores (robô, garra e passadeira) para serem atualizados e estes por sua vez reenvia os seus estados. Qualquer alteração ou atualização feita de um lado para outro é garantida através de um canal de comunicação.

3.3.1 Comunicação entre a camada de execução e de simulação

A comunicação entre estas camadas é fundamental, uma vez que a camada de execução funciona como cliente e a camada de simulação age como servidor. A interação entre eles é feita através da comunicação que garante a interoperabilidade. Esta comunicação tem que ser transparente de forma a permitir troca de informações corretas e instantâneas, e por isso, existem vários tipos de protocolos de comunicações em (Forouzan A., 2010). A comunicação entre processos (IPC –*Inter-Process Communication*) (Mitchell, Oldham, & Samuel, 2001) permite as interações entre o cliente e servidor. É de salientar que as interações só ocorrem após a conexão estiver estabelecida. Os protocolos de comunicação terão de ser escolhidos de acordo com as características das tecnologias utilizadas na implementação.

Para tornar a arquitetura mais flexível as duas extremidades de camada de execução e de simulação estão ligadas por uma API (*Application Programming Interface*) Figura 3.13. A API facilita o desenvolvimento de aplicação ao programador e diminui a quantidade de linhas de códigos desenvolvidas, uma vez que programador não precisa envolver-se com a implementação do software mas sim com o uso das suas funcionalidades através das API's. Estas API's são funções ou rotinas que interagem ou atuam diretamente com os módulos de recursos e/ou de transportes que são feitas através das interfaces. Muitos softwares já contêm

inúmeras funções de API's e estes podem ser desenvolvidos num qualquer linguagem de programação e ainda podem ser reutilizados por outros tipos de aplicações.

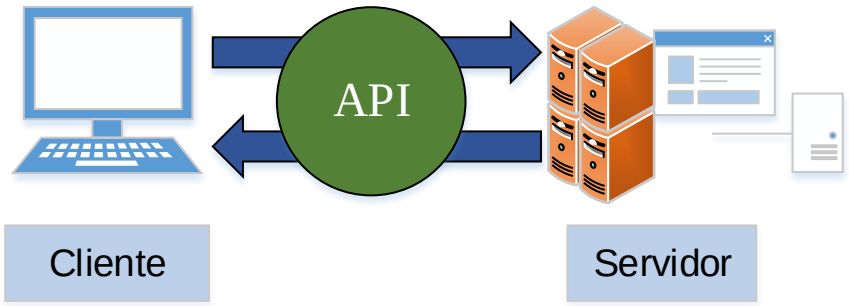


Figura 3.13 Comunicação entre o cliente e servidor

3.3.2 Integração de Recursos

3.3.2.1 Integração entre ResourceAgent e Módulos de Recursos

O RA é um dos agentes mais importante no sistema, visto que ele interage diretamente com os módulos de recursos a fim de permitir execução dinâmica dos seus skills atómicas. Isso acontece devido à integração entre agente e o seu recurso. Tendo em conta que os recursos (modelos físicos) não têm capacidades para comunicar e decidir, isto é, não possui nenhuma inteligência lógica e a integração com seu agente permite-o adquirir essas caraterísticas e outras, como autonomia, flexibilidade e interoperabilidade com outros agentes, portanto essas características deixa o sistema mais inteligente e organizável.

Na Figura 3.14 apresenta-se as rotinas do RA e os módulos de recursos.

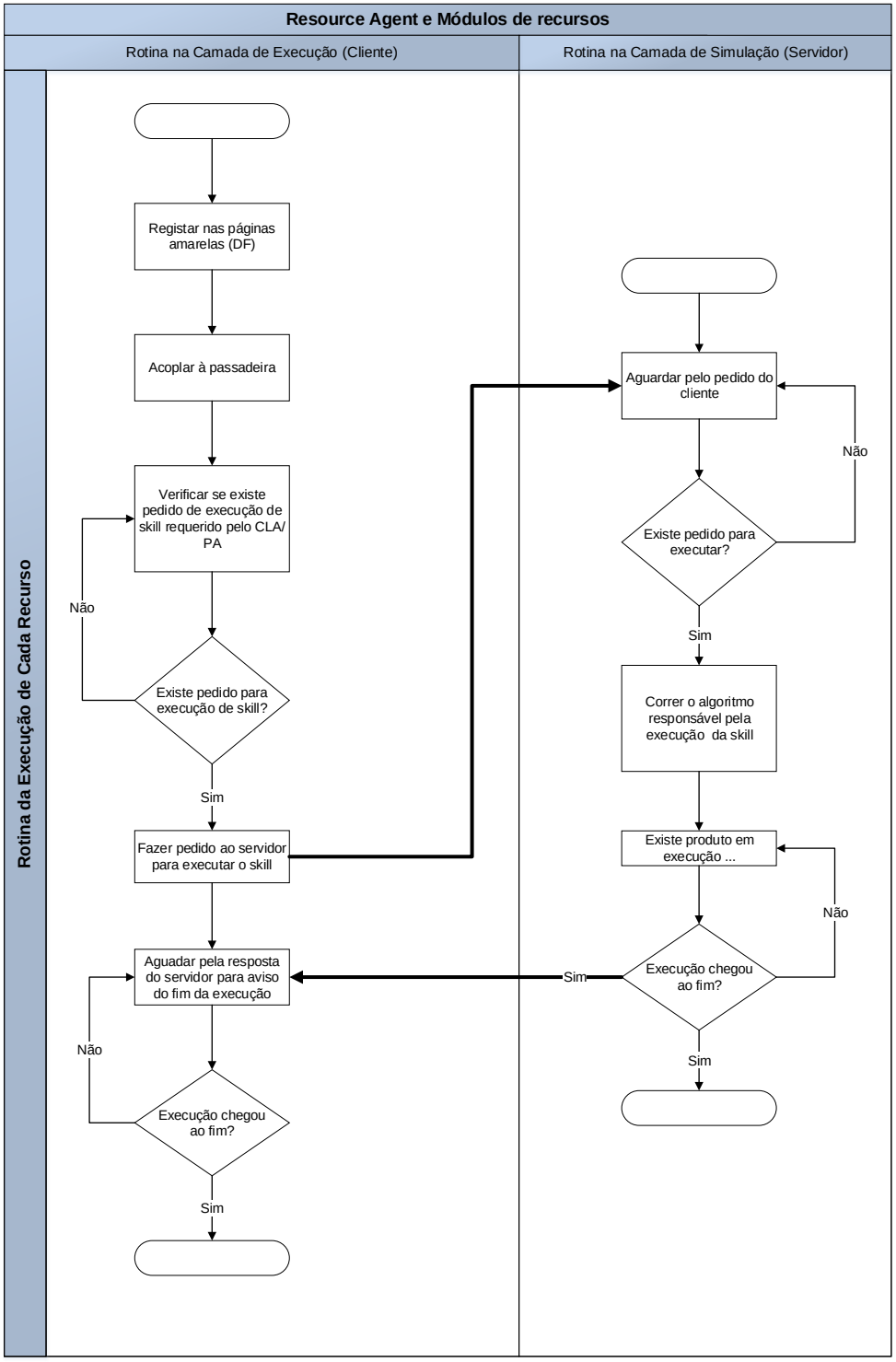


Figura 3.14 Diagrama de atividades entre RA e módulos de recursos

3.3.3 Integração de Transportes

3.3.3.1 Integração entre TEA e Passadeira

A necessidade de integrar TEA e passadeira ocorre quando TEA recebe o pedido do PA para a transportação de um produto. Sendo TEA um agente lógico que controla a passadeira, não possui capacidade de transportar fisicamente o produto, então neste sentido é que este é integrado com a passadeira. Esta passadeira tem a caraterística de transportação física, no entanto, a combinação dessas caraterísticas faz com que o sistema de transporte seja dinâmica, eficaz e inteligente ao ponto de decidir a transportação do produto adequadamente.

Na Figura 3.15 apresenta-se as rotinas do TEA e da passadeira.

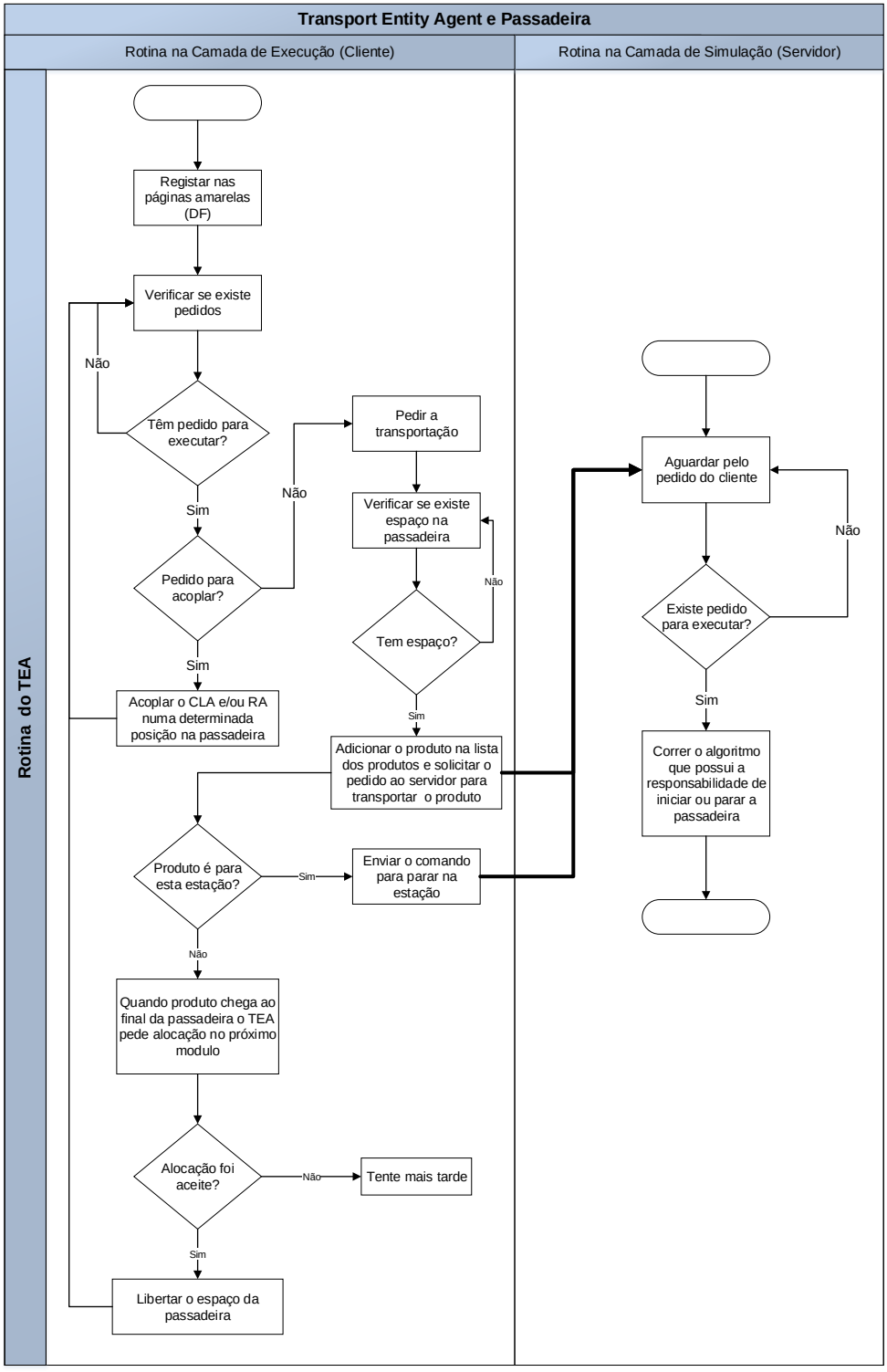


Figura 3.15 Diagrama de atividades entre o TEA e a passadeira

3.3.3.2 Integração entre o HUA e a Encaminhador

A integração entre HUA e o encaminhador é feita na mesma lógica que no caso da integração entre TEA e a passadeira (Figura 3.16).

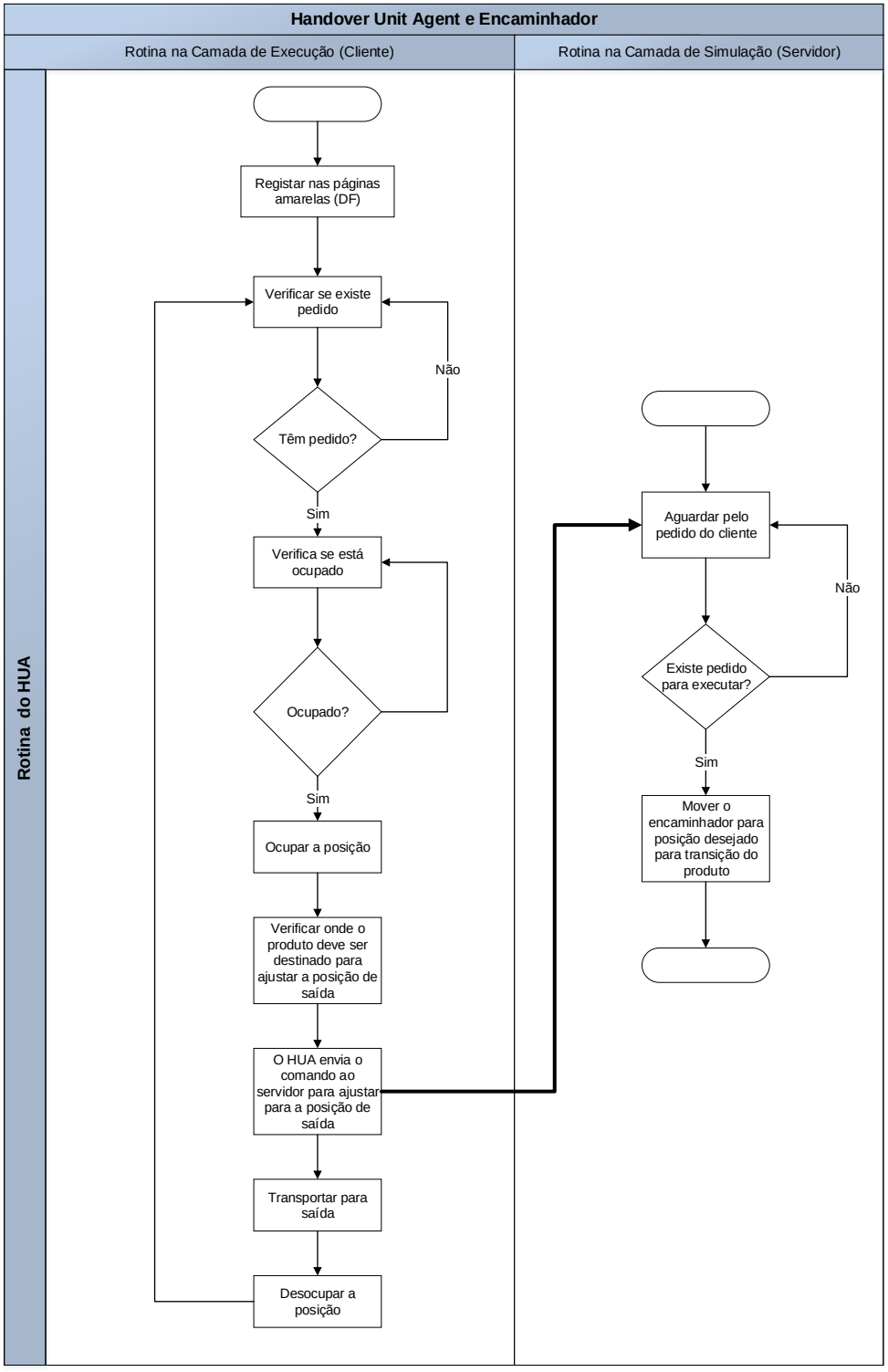


Figura 3.16 Diagrama de atividade entre HUA e o encaminhador

3.3.3.1 Integração entre PSoA e Ponto de Entrada

Para o lançamento do produto no sistema físico de produção é necessário que PSoA e ponto de entrada estejam integrados entre si. Esta integração permite manter atualização do

estado em ambos os lados. O ponto de entrada é responsável por introduzir o produto no sistema físico ao mandado de PSoA, isto é, o ponto de entrada não possui nenhuma inteligência lógica sobre o sistema. Qualquer decisão tomada perante o sistema é decidido pelo seu agente, sendo assim a sua integração faz com que PE simplesmente executa a ordem do PSoA.

Após a conexão estar estabelecida, o PSoA fica à espera de uma ordem de lançamento do produto. Em seguida, o PSoA atribui o nome ao produto em simultâneo com o PA e adiciona numa lista para ser lançado posteriormente. Na Figura 3.17 observa-se a rotina de lançamento do produto entre camadas. Nesta figura encontra-se duas colunas, observa-se a coluna do lado esquerdo ilustra a rotina do PSoA na camada de execução e do lado direita apresenta a rotina do PE na camada de simulação.

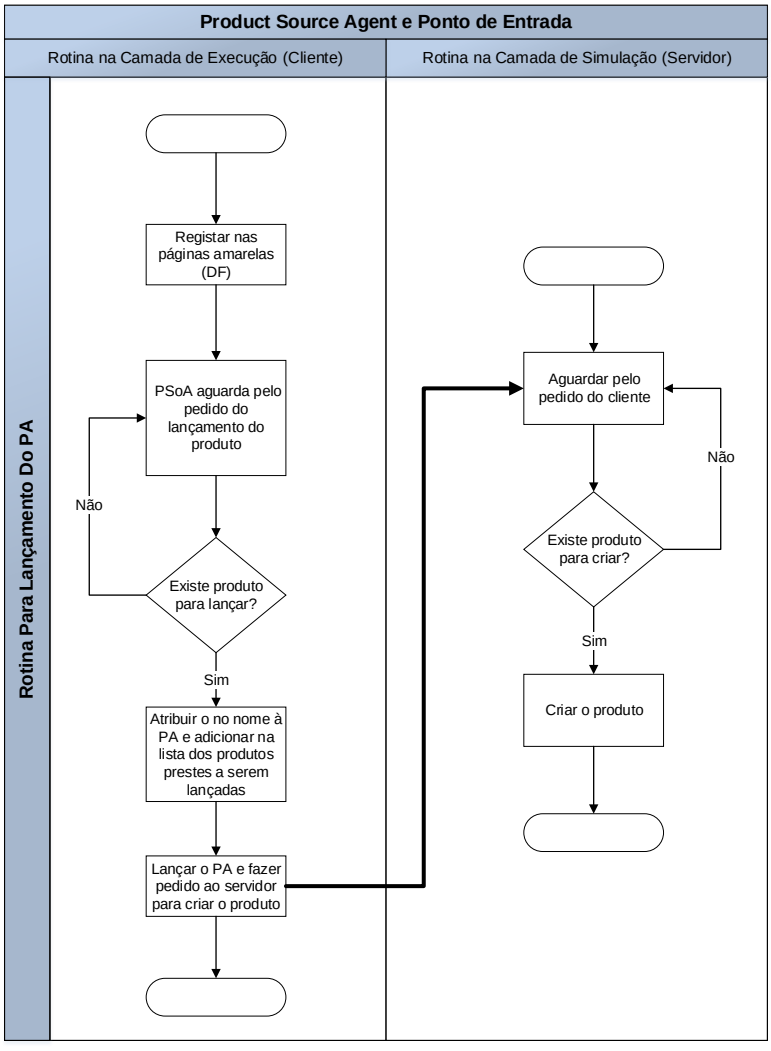


Figura 3.17 Diagrama de atividades da Integração entre PSoA e PE

3.3.3.2 Integração entre PSiA e o ponto de saída

Quando o produto termina a sua execução no sistema será eliminado através do PSiA e ponto de saída. Para garantir esta eliminação é necessária garantir a integração entre PSiA e ponto de saída, portanto, a eliminação do produto se realiza em ambas camadas (lógica e física). A da camada lógica, PSiA informa ao PA que o seu produto chegou ao fim e o PA será eliminado da plataforma, por outro lado na camada física o PSiA envia o comando para o PS para eliminar o produto do sistema.

Na Figura 3.18 apresenta-se as rotinas que PSiA e PS desempenham sobre as suas camadas. Após estar registado nas páginas amarelas o PSiA aguarda pela chegada do produto a ser eliminado do sistema.

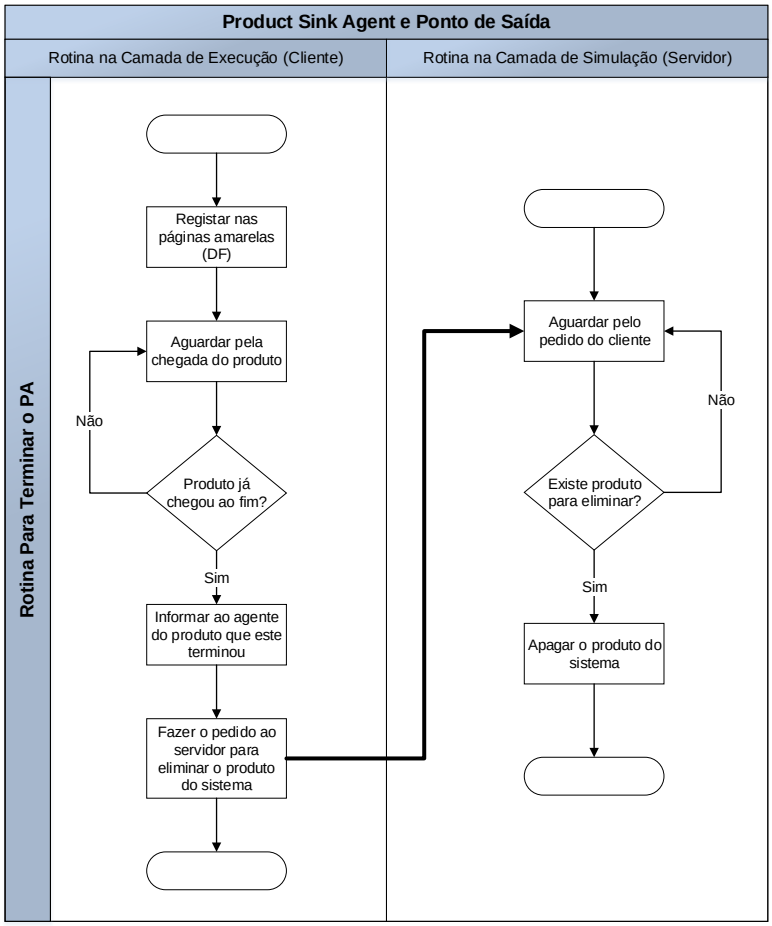


Figura 3.18 Diagrama de atividades de PSiA e ponto de saída

3.4 Camada de Simulação

A camada de simulação representada na arquitetura reproduz os modelos físicos de um sistema de manufatura. Esses modelos físicos são abstraídos pelos modelos virtuais que se encontram no simulador (neste caso o V-REP). O comportamento dos modelos virtuais deve ser

restringido aos funcionamentos básicos dos modelos que abstraem. Assim torna-se muito mais fácil implementar e/ou adaptar a arquitetura num modelo real. Esses modelos foram divididos em dois módulos, nomeadamente módulos de recursos e de transportes, que estão descritos nos seguintes subcapítulos.

3.4.1 Módulos de recursos

Os módulos de recursos representam as entidades físicas tais como os robôs, estações de serviços e operadores humanos. Esses módulos não possuem de nenhuma inteligência artificial além dos seus funcionamentos básicos como movimentação dos *joints* e ligar/desligar. O objetivo destes módulos é executar as suas funções básicas, a pedido dos agentes.

Na Figura 3.19 apresenta-se um exemplo da interação entre os agentes e os seus módulos. Como neste caso o agente do robô pede ao modelo de robô para mover à uma determinada posição, através do envio das posições pretendidos e por sua vez esta irá enviar o seu estado e que moveu com sucesso

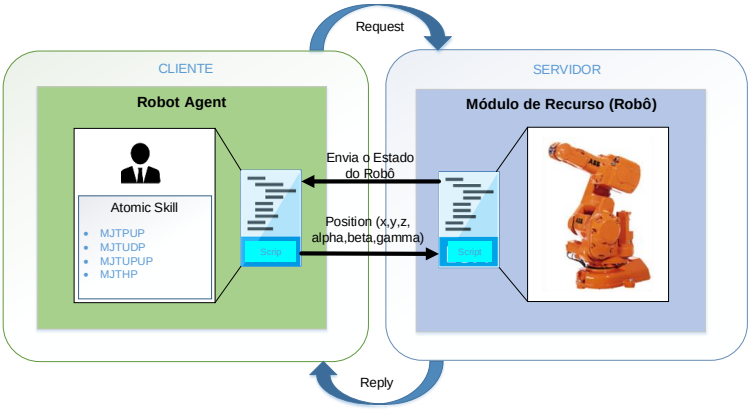


Figura 3.19 Interação entre RA's e os módulos de recursos

3.4.2 Módulos de Transportes

Os módulos de transportes simplesmente representam os mecanismos de transportes existentes numa linha de montagem, como por exemplo: passadeiras, encaminhadores, pontos de entrada e de saída. Os modelos também dispõem do comportamento básico como no caso das passadeiras e encaminhadores, permitem a leitura dos sensores, rodar para frente, parar, ligar/desligar.

4

IMPLEMENTAÇÃO

Neste capítulo descreve-se a implementação da arquitetura apresentada no capítulo 3. Nesta implementação recorreu-se ao ambiente de programação **NetBeans IDE** (suporta Java e JADE) e ao simulador **V-REP**. A camada de execução é implementada na linguagem Java em conjunto com a plataforma orientada ao comportamento, **JADE** (Bellifemine et al., 1999) e a camada de simulação é implementada no V-REP na linguagem Lua.

4.1 Tecnologia de Suporte

4.1.1 NetBeans IDE

O ambiente de programação utilizado para o desenvolvimento da camada de execução é o NetBeans IDE (*Integrated Devolopment Environment*) (NetBeans IDE, 2017) por ser uma plataforma flexível no desenvolvimento de software através de diferentes tipos de linguagens de programação como o Java, PHP, C/C++, HTML, etc. O NetBeans IDE pode ser executado em diversos tipos de sistema operativo como o Windows, MacOs, Linux, Solaris, etc. Esta plataforma é grátis, robusto, versátil, *open* source e ainda possui um fórum onde são partilhados diversos problemas e soluções que podem ser úteis durante o desenvolvimento de projeto.

4.1.2 JADE

O JADE (***J**ava **A**gent **DE**velopment framework*) é uma plataforma usada para o desenvolvimento de aplicações baseadas em sistemas multiagentes. Esta plataforma suporta a coordenação entre vários agentes conforme as especificações FIPA (***F**oundation for **I**ntelligent **P**hysical **A**gents*), para garantir a interoperabilidade entre sistemas multiagentes ou aplicativos baseados em agentes implementado em Java (Bellifemine et al., 1999). A linguagem de comunicação entre os agentes é definida em FIPA ACL (***A**gent **C**ommunication **L**anguage*). Para FIPA um agente é um programa ou uma máquina que comunica em ACL. Todas as mensagens

ACL têm a especificação do tipo de mensagem e um conjunto de parâmetros, descrita detalhadamente em (FIPA, 2002a).

Para interação entre agentes existem vários tipos de protocolos definidos, como por exemplo *FIPA-Request* que serve para pedir execução de uma ou mais ações, *FIPA-Contract-Net* que serve para pedir à um ou mais agentes as propostas de execução, e entre os protocolos que pode ser encontrado em (FIPA, 2002b) e (Bellifemine, Caire, & Greenwood, 2007). O JADE contém várias subclasses de *behaviour* que permite aos agentes adotar diferentes tipos de comportamentos durante a execução das tarefas, como *SimpleBehaviour, OneShotBehaviour, CyclicBehaviour, TickerBehaviour, WakerBehaviour, SequentialBehaviour, FSMBehaviour, ParallelBehaviour* etc. em (Bellifemine et al., 2007).

4.1.3 V-REP

O V-REP é um software aberto muito versátil e tridimendional desenvolvido por Coppelia Robotics. Nesta dissertação o V-REP é usado para a implementação da camada de simulação, uma vez que apresenta melhor característica para esta dissertação que pode ser encontrada na Tabela 2.4. Este software dispõe de uma licença gratuita para versão educacional com todas características de uma versão comercial. O V-REP é muito poderoso, oferece uma quantidade de funcionalidades que são muito adaptáveis. A sua arquitetura de controlo baseia-se na arquitetura distribuída que permite a simplificação dum sistema complexo em vários módulos simples que podem ser programados independentemente. Acresce ainda o facto de dás a possibilidade de controlar objetos ou módulos individualmente, este controlo pode ser conseguida de seis modos diferentes nomeadamente *embedded scripts, remote API clientes, custom solutions, Ros nodes, add-ons* e *plugins* (Coppelia Robotics, 2017b). Estes controlos podem ser programados em C/C++, Python, Java, Lua, Matlab, Octave ou Urbi.

Nesta dissertação o controlo é feito através do *remote API* e é programado em Java, por ser uma linguagem muito familiar e não só, mas também pelo facto de o JADE está completamente integrado em Java, sendo assim aproveita-se a experiencia adquirida durante o curso nesta linguagem.

4.1.3.1 Ambiente do V-REP

Neste subcapítulo serão apresentadas algumas funcionalidades mais pormenorizadas do simulador V-REP, que irá contribuir para a compreensão da implementação.

O V-REP é composto por três elementos:

- **Console window (janela de consola)** – exibe quais *plugins* que foram carregados e se o procedimento de inicialização foi bem-sucedida. Esta janela não é interativa é usada apenas para produzir informações.
- **Aplication window (janela de aplicação)** – é a janela principal do V-REP, é utilizada para exibir, editar, simular e interagir com uma *scene*.

- **Dialog (Diálogo)** – permite editar e interagir com uma sena ajustando a configuração do diálogo ou parâmetro.

Na Figura 4.1 apresenta-se as componentes do V-REP.

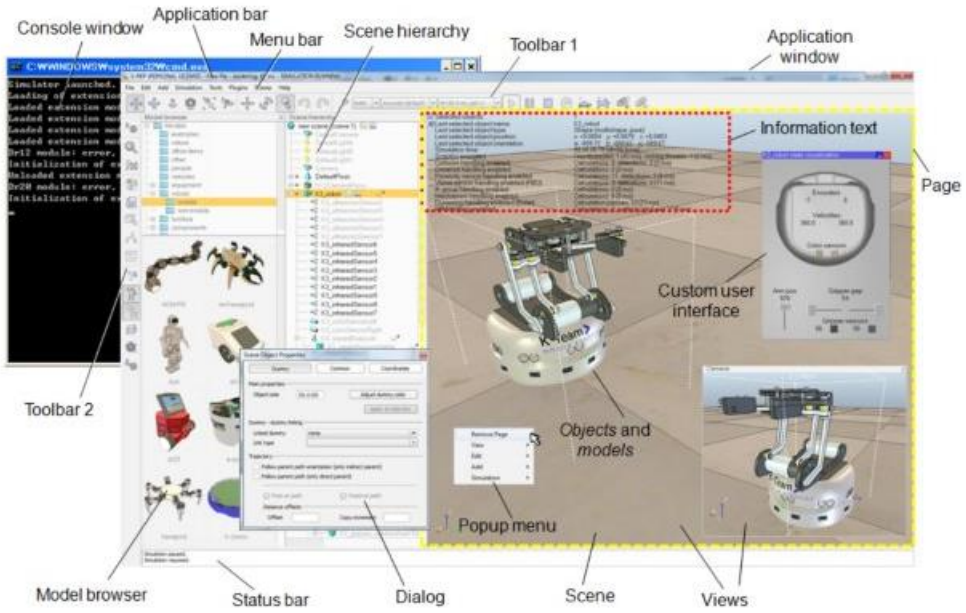


Figura 4.1Interface gráfica do V-REP (Coppelia Robotics, 2017c)

Dentro da janela de aplicação (*aplication Windows)* existem vários elementos, como por exemplo:

- **Aplication bar (barra de aplicação)** – mostra o tipo de licença do V-REP, o nome do arquivo que está a ser exibida, tempo usado para uma passagem de renderização e o estado atual do simulador.
- **Menu bar (barra de menu)** – permite acessar quase todas as funcionalidades do simulador V-REP.
- **Toolbars (barras de ferramentas)** – apresentam funções acessadas através de **toolbars 1** (Figura 4.2) e **toolbars 2** (Figura 4.3).

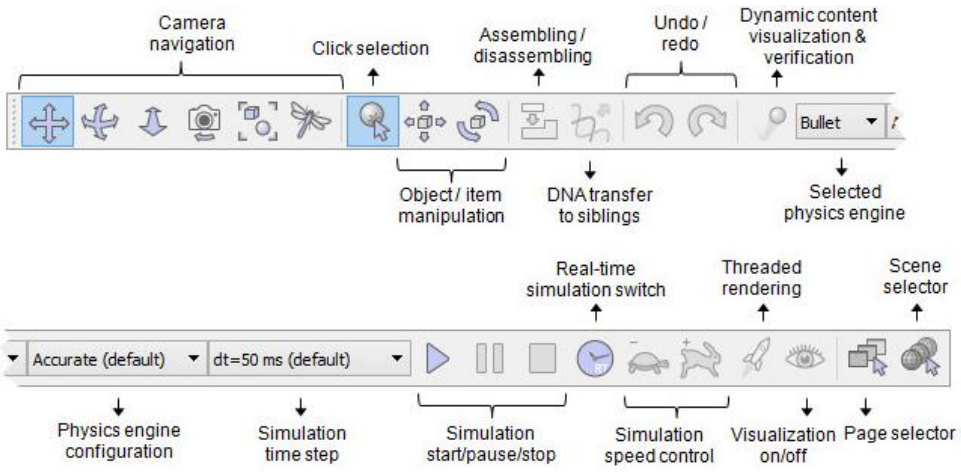


Figura 4.2 Toolbars 1 (Coppelia Robotics, 2017c)

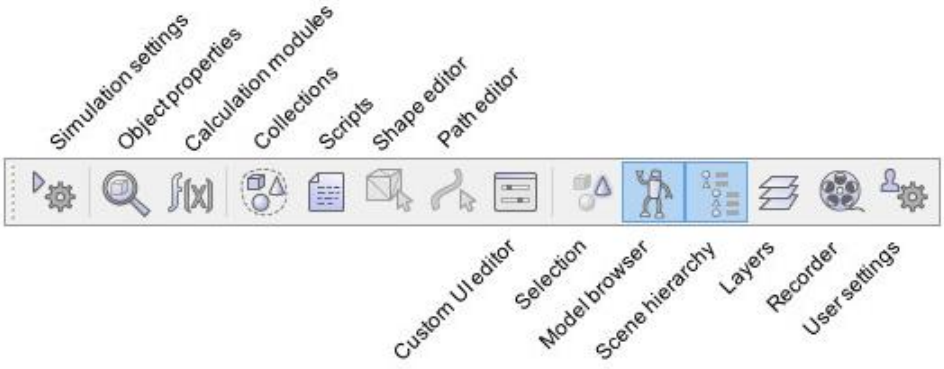


Figura 4.3 Toolbars 2 (Coppelia Robotics, 2017c)

- **Model browser (navegador do modelo)** – exibe na sua parte superior uma estrutura de pastas de modelos e na parte inferior são exibidas miniaturas de modelos contidas na pasta selecionada (Figura 4.4) adaptado de (Coppelia Robotics, 2017c).

- **Scene hierarchy (hierarquia da sena)** – exibe conteúdo de uma *scene* (ou seja todos os elementos de uma *scene* que compõem uma *scene*) Figura 4.5 adaptado de (Coppelia Robotics, 2017c).

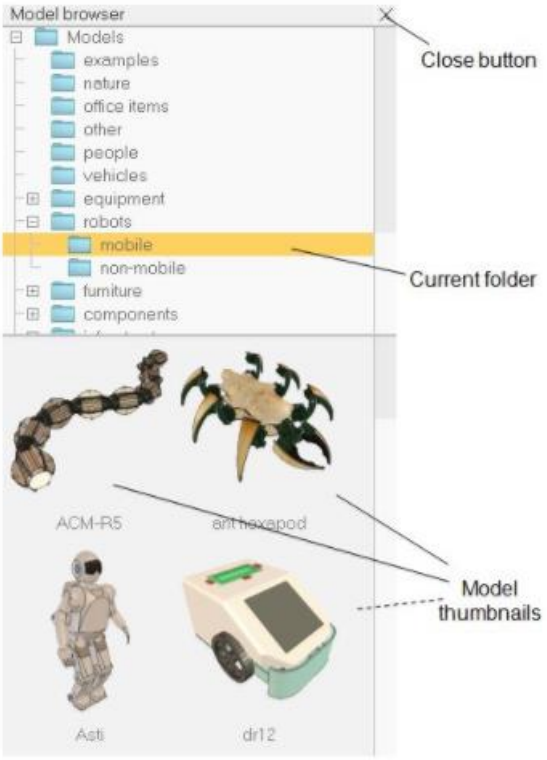


Figura 4.4 Navegador do modelo (Coppelia Robotics, 2017c)

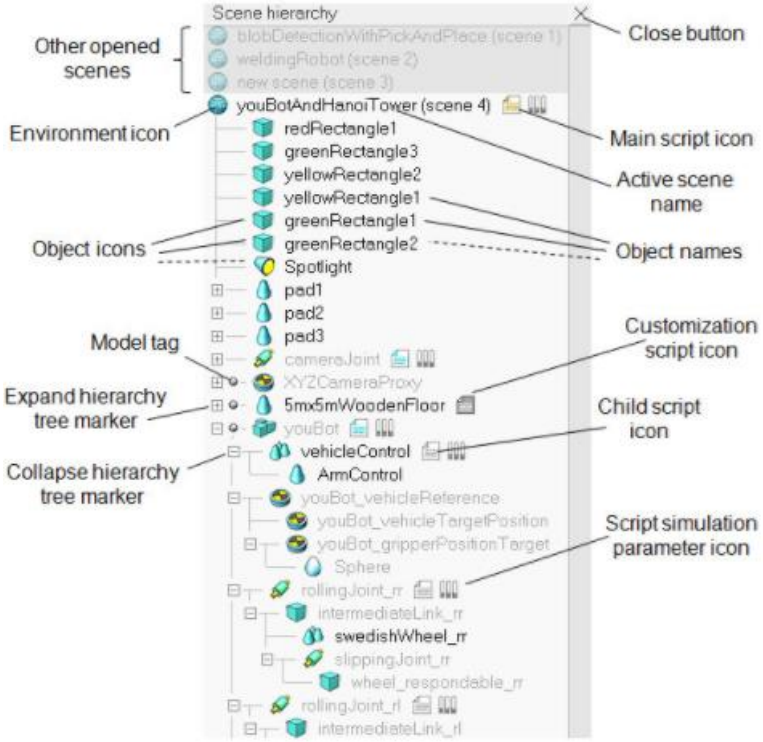


Figura 4.5 Hierarquia da sena (Coppelia Robotics, 2017c)

A arquitetura do V-REP é composto por três elementos centrais, como mostra na Figura 4.6.

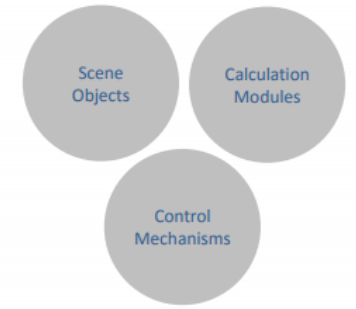


Figura 4.6 Elementos centrais da arquitetura do V-REP (Coppelia Robotics, 2017c)

4.1.3.2 Objetos de *scene*

Os objetos de *scene* é um dos principais elementos do V-REP que é utilizado para construir *scene* da simulação. Estes objetos podem ser adicionados a partir da barra de menu *add* ou com o botão direito do rato dentro da *scene* e seleciona opção *add.* Estes objetos são tridimensionais, podem ser combinados para construir objetos ou modelos mais complexos. Na Figura 4.7 apresenta-se alguns tipos de objetos contidos no simulador V-REP.

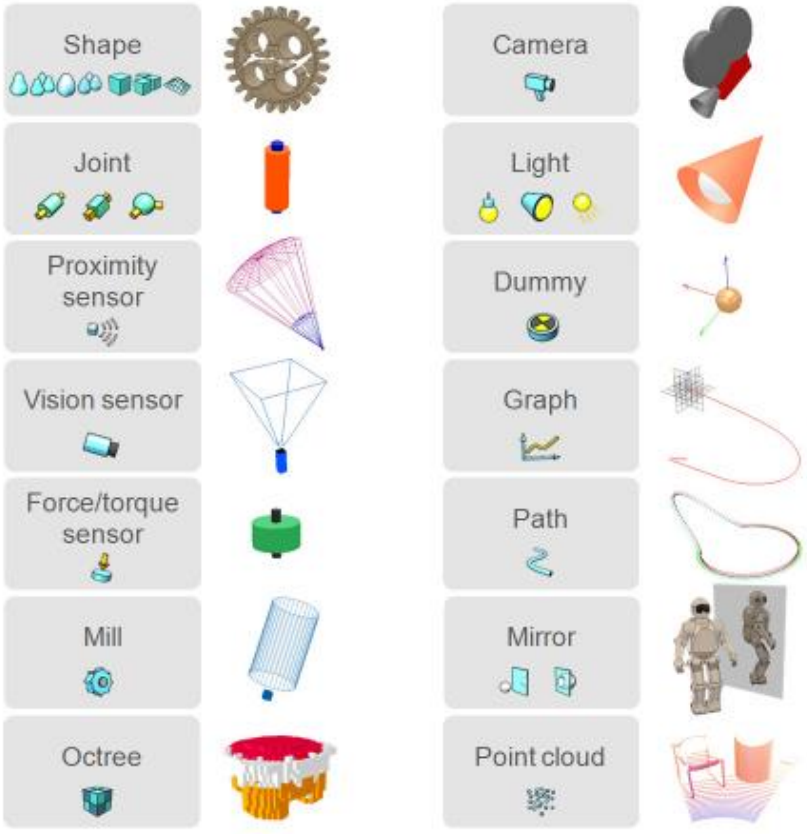


Figura 4.7 Objetos de scene (Coppelia Robotics, 2017d)

- **Shapes** – é uma malha rígida e é composta por faces triangulares.
- **Joints** – é um atuador que liga dois ou mais objetos, e existe quatro tipos de *joints: revolute joints*, *prismatic joints, screws joints e spherical joints.*
- **Graphs** – serve para gravar e visualizar os dados da simulação.
- **Dummies** – é um ponto com orientação, um objeto multifuncional que pode ser aplicado em diversas situações.
- **Proximity Sensors** – deteta objetos e determina a distância exata dentro do seu volume de deteção. Existem vários tipos de sensores de proximidade no V-REP: *Ray-type, Randomized ray-type, Pyramid-type, Cylinder-type, Disk-type e Cone-type.* Na Figura 4.8 apresenta-se o exemplo destes sensores sequencialmente.
- **Vision sensors** – é um sensor de tipo câmara que reage a luz, cores e imagem.
- **Force sensors** – é um objeto capaz de medir forças torques.

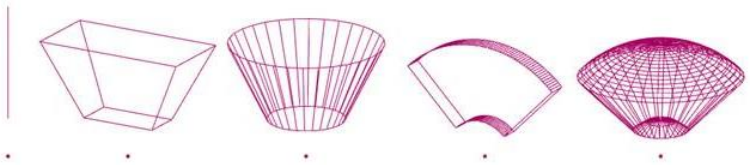


Figura 4.8 ray-type, pyramid-type, cylinder-type, disk-type e cone-type ou randomized ray-type (Coppelia Robotics, 2017e)

4.1.3.3 Módulo de cálculo

O módulo de cálculo oferece diversos tipos de cálculos que podem ser muito úteis dependendo das necessidades dos utilizadores. Os módulos de cálculo não são encapsulados diretamente num objeto, isto é, operam diretamente nos objetos encontrado na *scene.* Este módulo é adicionado a partir da barra de menu *tools* e selecionando *calculation modules* ou então a partir do *tollbar 2* apresentado na Figura 4.3.

Os módulos de cálculos podem formar sistemas complexos juntamente com objetos da sena e mecanismo de controlo. Os módulos de cálculo podem ser didcritos por:

- **Collision detection module** – permite rastrear, gravar e visualizar todas as colisões.
- **Minimum distance calculation module -** permite rastrear, gravar e visualizar distâncias mínimas entre qualquer entidade mensurável.
- **Forward/inverse kinematcs calculation** – permite resolver de forma muito eficiente qualquer tipo de cinemática inversa ou avançada.

- **Physics/Dynamics module** – Este permite a simulação dinâmica de objetos ou modelos para alcançar interações de objetos, como resposta de colisão, captação, etc.
- **Path/motion planning module** – permite a realização de cálculos de *path/motion planning* baseados na biblioteca OMPL (*Open Motion Planning Library*).

Na Figura 4.9 apresenta-se os esquemas dos módulos descritos anteriormente.

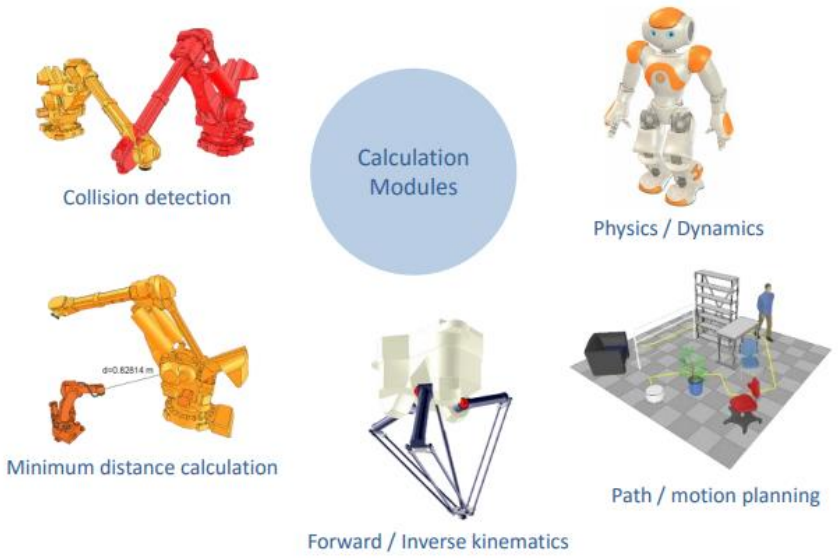


Figura 4.9 Módulos de cálculo (Coppelia Robotics, 2017b)

4.1.3.4 Mecanismos de controlo

Como anteriormente referido o V-REP é um simulador altamente personalizável, capaz de se adaptar a qualquer projeto. Esta flexibilidade de adaptação e personalização é obtida por seis diferentes mecanismo de programação que permite o controlo de objetos ou modelos. Estes mecanismos encontrem apresentados na Figura 4.10. A principal vantagem dos mecanismos de controlo é a utilização simultânea ou então *hand-to-hand.*

O mecanismo de controlo está dividido em dois interfaces distintos:

- **Interface Local** - *Embedded scripts, Plugins, Add-ons*.
- **Interface Remote** - *Remote API clients, Custom solutions, and ROS nodes.*

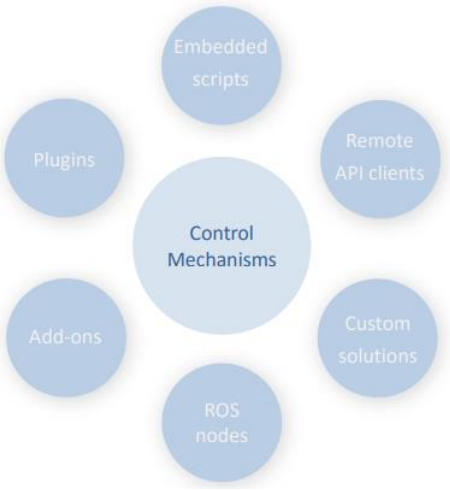


Figura 4.10 Mecanismo de controlo V-REP (Coppelia Robotics, 2017b)

De seguida é apresentado uma breve discrição dos mecanismos

- **Embedded scripts** – personalizar uma simulação (modelo ou *scene*) por meio de um script.
- **Plugins** – consiste em escrever um *plugin* para V-REP e é usado em conjunto com *embedded scripts.*
- **Add-ons** – consiste em escrever scripts na linguagem Lua, permite personalizar rapidamente o próprio simulador V-REP.
- **ROS nodes** - permite uma aplicação externa (robô, máquina) conectar ao V-REP via ROS (Robot Operating System).
- **Remote API clients** – este método permite uma aplicação externa (robô, máquina) conectar ao V-REP de uma forma fácil através do comando do *remote API.*

4.1.3.5 Ambiente de programação

O V-REP oferece possibilidade para programador em várias linguagens distintas, dependendo do mecanismo de controlo utilizado. A linguagem principal do simulador é Lua (LUA, 2017). Esta linguagem foi criada em 1993 no Tecgraf, o Grupo de Computação Gráfica da PUC-Rio no Brasil. A Lua é uma linguagem de *script* projetada para ser acoplada à programas mais complexos (Celes, Figueiredo, & Ierusalimschy, 2003). Assemelha-se ao *Python, Ruby, Icon* entre outras, é usada em diversos ramos de programação como no desenvolvimento de jogos, controlo de robôs, processamento de texto. Devido à sua eficiência, simplicidade e rapidez o método de programação em Lua pode ser encontrado em (Celes et al., 2003; Ierusalimschy, 2009).

As chamadas de Lua no V-REP são tratadas através da biblioteca *v_repLuaLibrary* que se encontra localizada na pasta de instalação do V-REP. Nesta pasta encontra-se centenas de

funções (Coppelia Robotics, 2017a). Essas funções são facilmente reconhecidas por seus prefixos *sim,* como por exemplo *simGetObjectPosition*, que podem ser chamados dentro *Child scripts.*

A execução de simulação no V-REP começa quando o aplicativo cliente chama um *main script,* que por sua vez este pode chamar o *child script.* Cada *scene* na simulação tem um único *main script.*

O *child script* representa um pequeno código escrito em Lua, que permite manipular e controlar uma determinada função do objeto associado. Na Figura 4.11 apresenta-se um objeto robô IRB140 que está associado ao *child scripts*. Este objeto pode ser facilmente reconhecido a partir do ícone do *script* encontrado na hierarquia de *scene* (Figura 4.5). Ao contrário do main script, numa *scene* pode ter inúmeros *child scripts* uma vez que cada um deles está associado à um objeto.

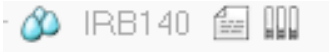


Figura 4.11 Child script associado ao robô IRB140

Para abrir o editor do *child script* é necessário apenas um duplo *click* no ícone, sendo assim é possível alterar ou editar o código de programação.

Figura 4.12 Icone child script (à esquerda non-threaded e à direita threaded child script)

O *child script* está devido em dois diferentes tipos:

- **Non-threaded child scripts -** são scripts *pass-through.* Isto quer dizer que sempre que são chamados, os scripts devem executar as suas tarefas e após a conclusão devem regressar ao script de controlo do chamador, e se não regressar ao script do chamador toda a simulação pára. Normalmente esse tipo de script é chamado em cada passo de simulação e pode ser chamado por um *threaded child scripts.* Neste tipo o ícone de *child script* é cinzento (Figura 4.12)
- **Threaded child scripts -** são scripts que serão lançados num *threaded.* O seu lançamento é controlado pelo *main script.* Ele pode chamar um *non-threaded child script.* Estes tipos de scripts têm vários pontos fracos em comparação com os *non-threaded child scripts* se não forem programados adequadamente. Podendo ser intensivos em recursos, desperdiçar muito tempo de processamento e podem ser um pouco mais sensíveis a um comando de parada de simulação. Neste tipo o ícone de *child script* é azul claro (Figura 4.12).

4.2 Implementação da Camada de Execução

Neste subcapítulo é apresentado as implementações dos agentes encontrados na arquitetura proposta.

4.2.1 Agentes do Suporte

Os agentes de suporte têm como principal objetivo garantir o bom funcionamento dos agentes na plataforma. O DA foi implementado simplesmente para lançar os agentes de execução e de transportes na plataforma. O serviço DF é disponibilizado pelo FIPA, no qual tem uma lista que contém todos agentes e os seus respetivos serviços inscritos na plataforma. Foram definidos vários métodos que permite pesquisar agentes através dos serviços inscritos. Estes métodos permitem também pesquisar agentes numa determinada área, permitem inscrever/desinscrever agentes e serviços entre outros métodos relevantes.

4.2.2 Agentes de Execução

4.2.2.1 Resource Agent

O RA é o agente responsável por abstrair estações de trabalhos como por exemplo robô, operadores humanos ou outros tipos de componentes inseridos na linha de produção. Ele é um agente que interage diretamente com o hardware ou controlador do sistema. Na classe *ResourceAgent* que no qual, foi implementada vários tipos funções de modo que as interações com outros agentes e com os seus recursos sejam adequadas. Na Figura 4.13 encontra-se a diagrama da classe ResourceAgent.

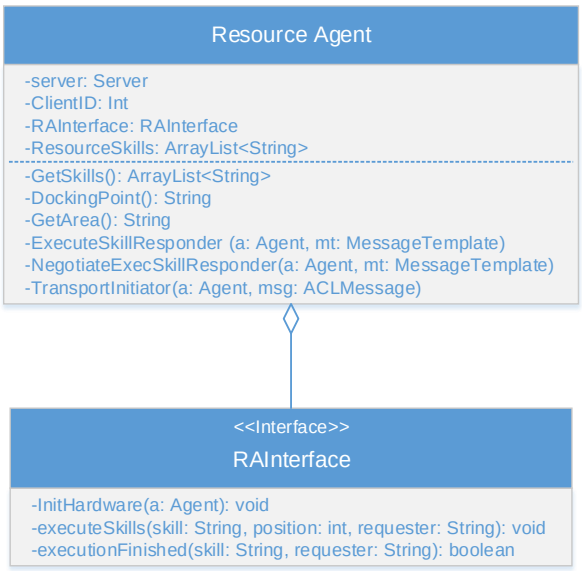


Figura 4.13 Diagrama de classe RA

Para garantir o bom funcionamento dos agentes, foram definidos alguns atributos na sua implementação. Na Figura 4.13 apresenta-se alguns atributos utilizados como por exemplo:

- **server -** é uma variável do tipo *Serve*r, no qual contém conjuntos funções API que permite interações com os simuladores, isto é, um mecanismo de controlo, *Remote API clientes.* Estas funções permitem acessar funções definidas no lado do servidor.
- **ClientID -** é uma variável do tipo inteiro que guarda um ID, através do qual o cliente se identifica para comunicar com o servidor através do porto.
- **ResorceSkills -** é uma lista de skills adicionada aquando do seu lançamento.

A classe do RA tem uma interface associado, **RAInterface***,* que dispõe de três métodos para a interação entre o agente e o seu recurso. O **InitHardware** tem a função de iniciar esta interface e é chamado no início do lançamento do seu agente. O **executeSkills** é chamado aquando da execução do skill. O **executionFinished** verifica se a execução do skill já chegou ao fim.

Na classe do RA são definidos conjuntos de métodos que permitem os agentes realizarem as suas funções adequadamente:

- **GetSkills() -** é um método que retorna a lista dos seus skills, que posteriormente serão inscritos nos serviços das páginas amarelas.
- **DockingPoint() -** é um método que retorna o ponto de acoplamento do agente, este ponto é do tipo string, que contém nome das passadeiras onde este se pretende acoplar. Este ponto é usado como possível destino do produto.
- **GetArea() -** retorna a área deste agente, que posteriormente será utilizada pelo CLA.

O **TransportInitiator** é um *behaviour* do tipo *AchieveREInitiator*, que inicia o pedido ao TEA, para informar o ponto de acoplamento na passadeira. Este ponto é o valor retornado do DockingPoint(). Na primeira coluna da Figura 4.14 encontra-se o diagrama de atividade que a **TransportInitiator** desempenha durante a execução do processo. Ainda na mesma coluna encontra-se definido um diagrama de sequência que ilustra como RA interage com TEA. O protocolo utilizado na troca de mensagens é *FIPA Request,* em que *Request* é para fazer pedido e *Inform* é para informar ao requerente. O DockingPoint() é enviado através do *setContent(DockingPoint())* disponibilizado pelo FIPA.

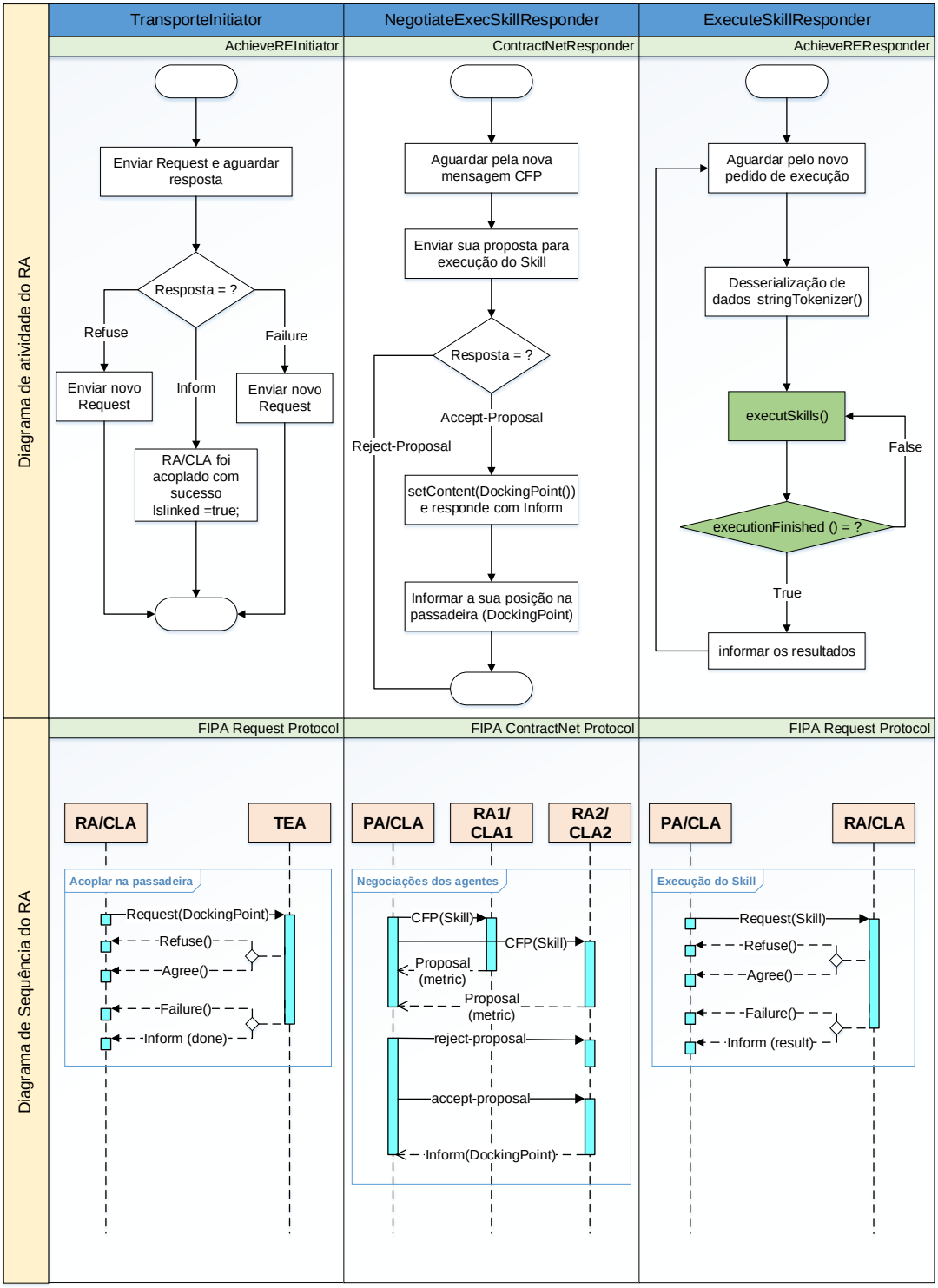


Figura 4.14 Diagrama de atividades e de sequências do RA

O **NegotiateExecSkillResponder** é um *behaviour* do tipo *ContractNetResponder*, implementado para oferecer as suas propostas para execução do skill. Estas propostas serão posteriormente utilizadas pelo requerente para decidir qual é a melhor proposta para a execução. A interação de mensagens entre os agentes é feita através do protocolo *FIPA ContractNet*. Este protocolo utiliza *CFP* para fazer pedido de proposta que é respondido através do *Propose,* no

qual envia sua proposta através do *setContent().* Na Figura 4.14, a segunda coluna corresponde a diagrama de atividade de sequência do NegotiateExecSkillResonder. A partir do momento que PA/CLA aceita proposta do RA, PA/CLA aguarda pela ordem de execução, executado através do **ExecuteSkillResponder.**

O **ExecuteSkillResponder** é um *behaviour* do tipo *AchieveREResponder* que responde aos pedidos efetuados pelo CLA ou PA para execução de um determinado skill. Nesta arquitetura assume-se que PA/CLA sempre aceita o pedido de execução do skill, uma vez que já tinha negociado para execução. Na terceira coluna da Figura 4.14 encontra-se o diagrama de atividade e de sequência. No diagrama de sequência a coluna está preenchido de verde devido à interação entre o agente e simulador V-REP.

4.2.2.2 Coalidition Leader Agent

O CLA é um agente especial na camada de execução, ele é único agente da camada de execução que pode interagir entre si. Por esta razão as suas interações são baseadas em regras. O CLA tem uma caraterística especial, só pode coordenar agentes da sua área. Para esta implementação foi criada uma classe chamada de *CoalitionLeaderAgent* onde é definida os atributos e métodos que permite alcançar o comportamento desejado dos agentes.

Figura 4.15 Diagrama da classe do CLA

Na Figura 4.15 apresenta-se o diagrama da classe do CLA com alguns atributos relevantes, como por exemplo:

- **CLASkills** - é uma lista que guarda os skills (CSk), que são adicionados no seu lançamento.
- **SubSkill** - contém uma lista de skills (CSk e ASk) dos agentes para negociação ou execução.
- **AllmyAgent** - é uma *HAshMap* que contém os agentes de cada área. A chave do *HashMap* é a área (string) e os valores são agentes das respetivas áreas. Os

agentes são pesquisados pela área, assim permite o CLA coordenar agentes inseridos na mesma área.

- **Rules (skill: String, área: String) –** é responsável por decompor os skills complexos (CSk) de mais alto nível até ao nível inferior. Estes skills do nível inferior serão retornados para a variável **SubSkill**. No fim das decomposições sobram só os skills atómicos (Ask), uma vez que CSk é composto por conjuntos de ASk. É de salientar que este método retorna uma lista de skills sequenciais, que serão executadas de acordo com as ordens que se encontram na lista.

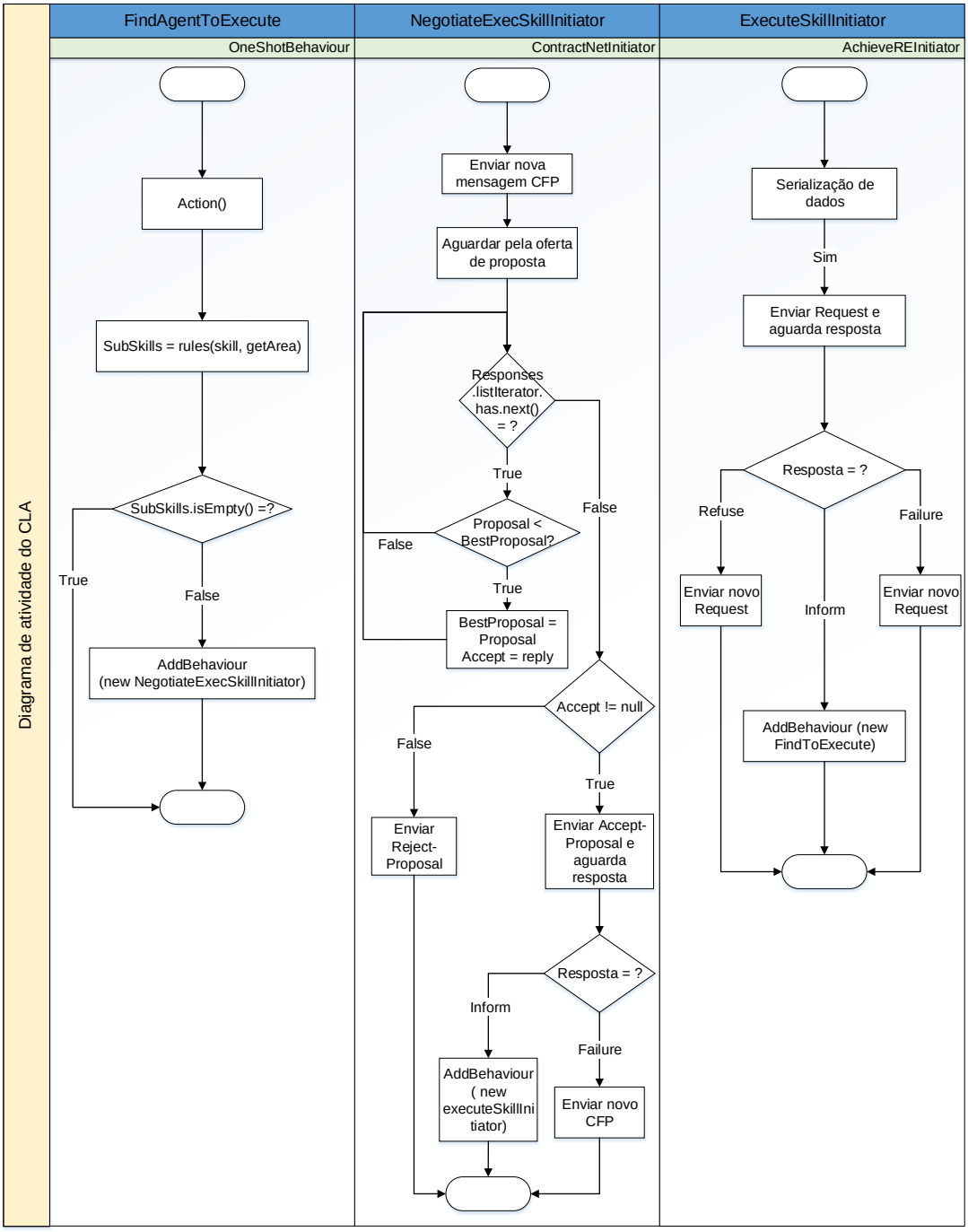


Figura 4.16 Diagrama de atividades do CLA

Os métodos como **getSkill(), DockingPoint() e getArea()** têm as mesma função que os RA's. Assim como os *behaviours* **TransportInitiator, ExecuteSkillResponder e NegotiateExecSkillResonder** desempenham os mesmos comportamentos, visto que o CLA as vezes comporta como um RA, uma vez que são agentes mecatrónicos. A diferença é que o CLA não executa o seu skill diretamente com o hardware, esta são executadas pelos RA's associados.

O **FindAgentToExecute** é um *behaviour* do tipo *OneShotBehaviour*, implementado para procurar agentes associados ao CLA. Estes agentes são procurados através dos **Subskill**

retornado pelo método **Rules**. Esses skills retornados são negociados com os seus agentes, (primeira coluna da Figura 4.16), através do *behaviour* **NegociateExecSkillInitiator.** Este behaviour é do tipo *ContractNetInitiator,* responsável por pedir proposta para execução de cada skill e decidir qual é a melhor proposta. Aceita apenas uma e as restantes são rejeitadas, só acontece quando na mesma área existir mais de que um agente que pode executar o mesmo skill.

Na Figura 4.16, na segunda coluna, encontra-se o diagrama de atividade e diagrama de sequência é a mesma que **NegotiateExecSkillResponder** do RA. Uma vez que os dois interagem-se através do protocolo *FIPA ContractNet,* isto é, um inicia o pedido e o outro responde ao pedido. Após a negociação, o *behaviour* do **ExecuteSkillInititor** é iniciado para execução do skill negociado. Este *behaviour* é do tipo *AchieveREInitiator* e a sua interação com **ExecuteSkillResponder** é através do protocolo *FIPA Request* (terceira coluna ilustrada na Figura 4.14). O principal objetivo deste *behaviour* é iniciar o pedido para o RA executar as habilidades.

4.2.2.3 Product Agent

O PA é um dos agentes mais importante da arquitetura, visto que todos os outros agentes comportam de forma a satisfazer as suas tarefas. Este agente não interage diretamente com o seu produto (abstraído por ele) no simulador V-REP mas controla apenas o plano de produção, isto é, todas as comunicações com outros agentes e as decisões são da sua responsabilidade.

Para implementar o comportamento pretendido pelo PA foi criada a classe *ProductAgent*, (Figura 4.17). Nesta classe foi implementada um conjunto de atributos e métodos que permitissem a reprodução do seu comportamento.

Figura 4.17 Diagrama de Classe PA

Para PA funcionar de acordo com os objetivos pretendidos, foi definida uma lista que contém os Skills. Essas skills serão executadas na mesma ordem em que se encontra na lista no qual o PA utiliza para procurar no DF os agentes responsáveis pela sua execução

- **Skills** - é uma lista de skills, que é executada para completar os objetivos da produção. Através deste skill o PA procura no DF os agentes responsáveis pela

execução, e posteriormente inicia-se a negociação ou execução. Esta lista varia de acordo com o tipo de produto (*TypeProduct*).

- **TypeProduct** - contém um inteiro que indica o tipo de produto.
- **Destiny** - contém o destino do produto onde será executado.
- **Skill** – contém o nome do skill negociado para execução.
- **IDAgentToExec** - contém o ID do agente para execução da **skill**, que será executada posteriormente.

O **StartProcess** é um *behaviour* do tipo *OneShotBehaviour* que foi implementado para iniciar o processo de execução. Este behaviour procura através do DF os agentes disponíveis para negociação de um determinado skill. Após encontrar os agentes disponíveis, inicia-se a negociação através do *behaviour* **NegotiateExecSkillInitiator**, que possui exatamente o mesmo comportamento que o CLA (Figura 4.16).

Após a negociação do agente, o PA faz um *Request* ao TEA para transportar o produto até ao destino. Este destino foi obtido durante a fase de negociação, implementado no *behaviour* **TEAIniciator** que é do tipo *AchieveREInitiator* que envia o pedido ao TEA. O TEA por sua vez responde com a performativa *agree* ou *refuse*.

Quando o produto chega o destino pretendido, o TEA irá informar ao PA que o produto chegou ao destino, e de imediato o PA irá fazer um *Request* ao agente negociado para começar a execução do skill. Esta decisão foi feita através do behaviour **ExecuteSkillInititor** que foi implementado como o CLA. Na Figura 4.16 encontra-se o diagrama de atividade e de sequência feito neste *behaviour*.

Para garantir que o PA deve ser eliminado do sistema, foi implementado o *behaviour* **FinishedProduct**, que é do tipo *TickerBehaviour*. A cada intervalo do tempo verifica-se se não existe nenhum aviso que o produto encontra-se no ponto de saída, Caso tiver o PA será apagado da plataforma.

Na Figura 4.18 é possível observar o diagrama de sequência completo que o PA deve cumprir para alcançar os objetivos de produção. Este diagrama encontra-se devido em seis fases sequenciais. Na primeira fase, o PA faz a negociação com os agentes responsáveis pelas execuções das suas tarefas, onde decidir á qual é melhor agente para a execução. Na segunda fase, o PA pede ao TEA para transportar o produto até ao destino, se o destino não estiver na respetiva passadeira será iniciado a terceira fase. Nesta fase o TEA pede ao HUA para encaminhar o produto. É de salientar que enquanto o produto não chegar ao TEA de destino as fases dois e três serão repetidos. Na quarta fase o PA é informado que chegou ao destino e de imediato irá começar a quinta fase. Nesta fase o PA faz um pedido ao agente negociado para iniciar a execução. Na sexta fase, após terminar a execução, o PA é informado e de seguida faz um novo pedido ao TEA para transportar o produto para o ponto de saída.

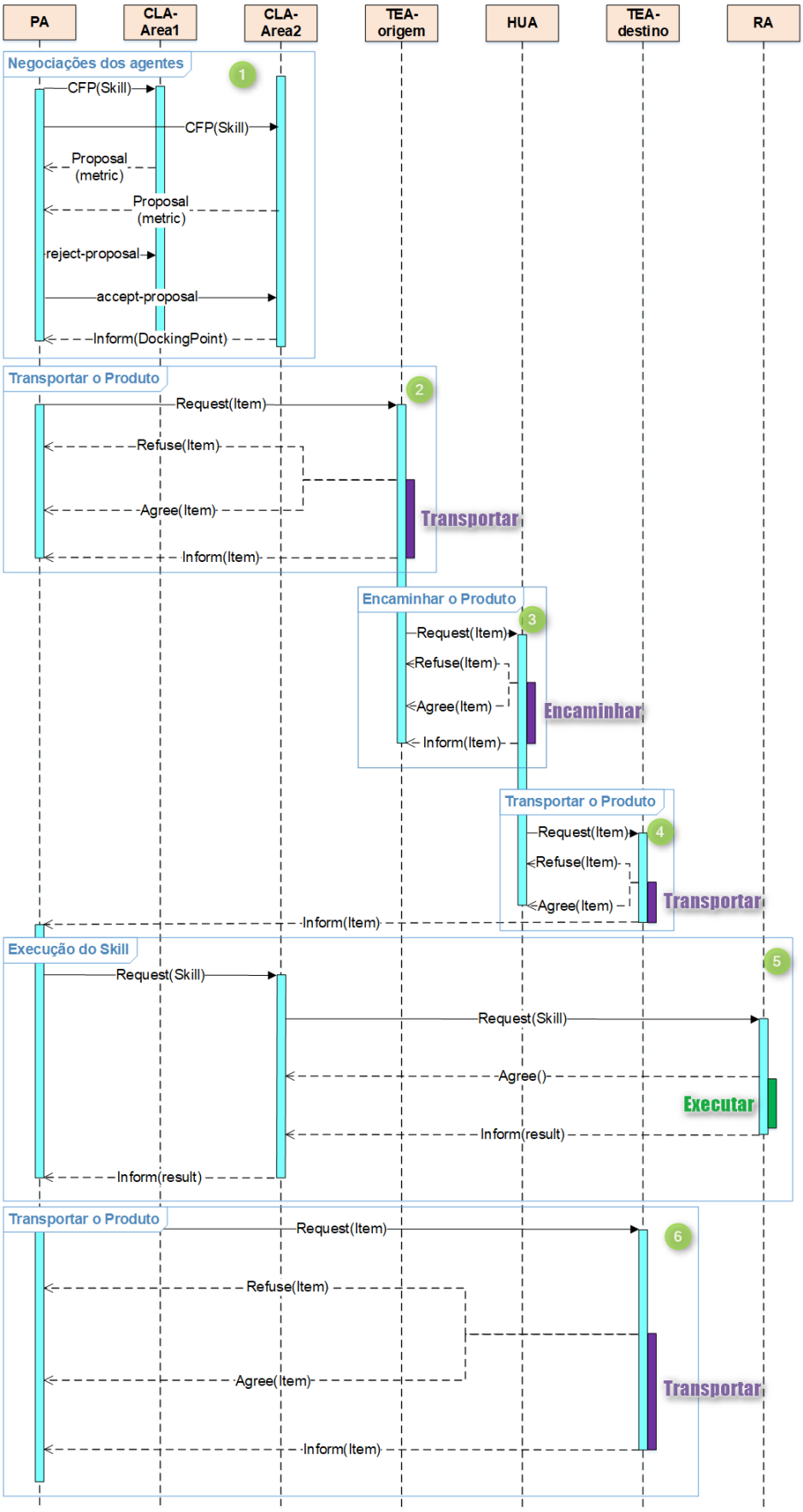


Figura 4.18 Diagrama de sequência completo do PA

4.2.3 Agentes de Transporte

4.2.3.1 Product Source Agent

Para a implementação so PSoA foi criada uma classe chamada de *ProductSourceAgent*. Nesta classe foi implementada um conjunto de atributos e métodos (Figura 4.19).

Os atributos **server** e **ClientID** tem a mesma função que os RA's. Estes atributos são usados para estabelecer comunicação entre agentes e servidor (simulador V-REP). Esta classe está associada à interface **PSoAInterface**, que permite a implementação dos métodos responsáveis nas interações entre os agentes e servidores. Os métodos usados são *initHardware* e *CreateProduct.* O *initHardware* é responsável por iniciar a interface. O *CreateProduct* tem a responsabilidade de criar o produto no simulador V-REP. É de salientar que este produto dispõe o mesmo nome que o do seu agente (PA), sendo assim facilita a sua identificação no simulador V-REP.

O atributo **Products** é uma lista que guarda os produtos que estão prestes a ser lançados. Estes produtos ficam na fila de espera e são lançados um de cada vez. Os produtos são objetos do tipo *Item,* que dispõe de toda informação do PA, como o *nome*, *origem*, *próximo destino*, *destino final* e *msg* para informar a chegada. O *item* contém dois métodos Set() e Get() que permite alterar e consultar os atributos.

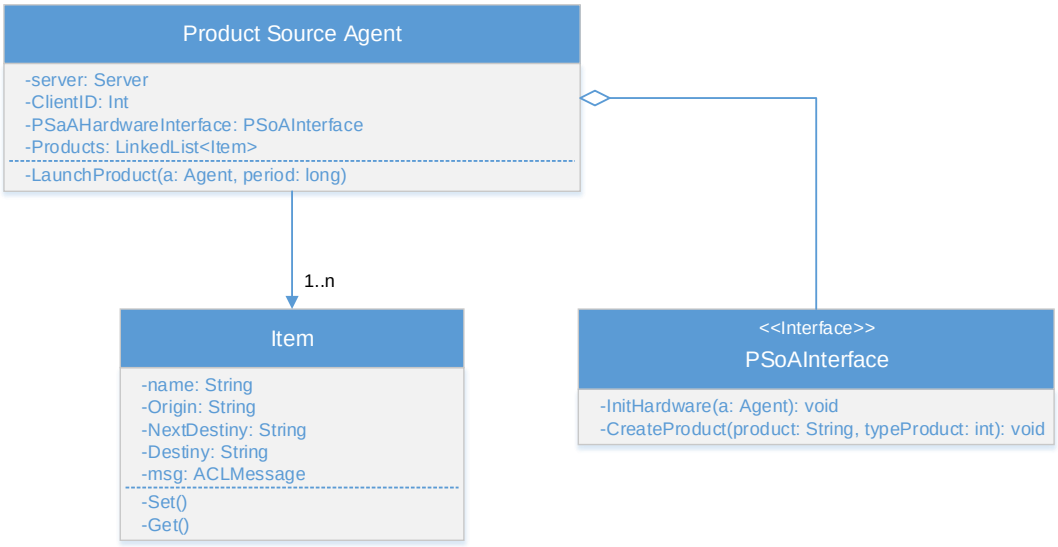


Figura 4.19 Diagrama da classe PSoA

O **LaunchProduct** é um *behaviour* do tipo *TickerBehaviour* que foi implementado para lançar o PA na plataforma e criar produto no simulador V-REP ao ordem do utilizador, que interage com o agente DA (Deployment Agent) através de uma interface implementada (Figura 4.21). O DA possibilita ao utilizador escolher o tipo de produto e ordenar o seu lançamento. Após a ordem de lançamento o DA interage com o PSoA. Nesta interação, o DA informa que tem um

novo produto para ser lançado de acordo com a disponibilidade do sistema. O protocolo de comunicação usado na interação foi *FIPA Request* (Figura 4.20).

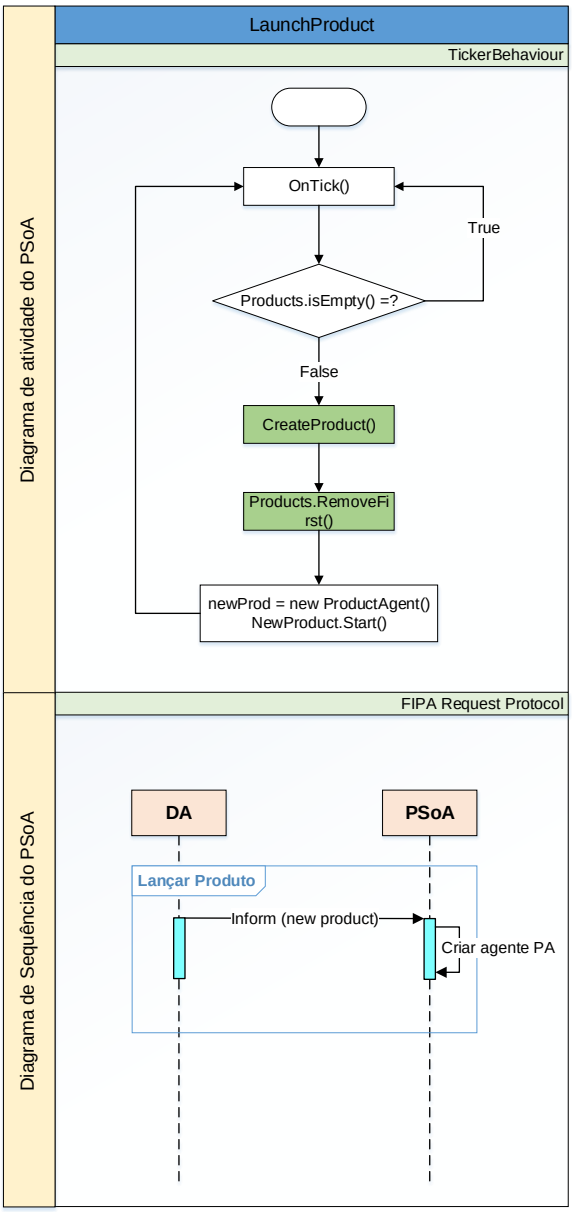


Figura 4.20 Diagrama doBehaviours do PSoA

A interface da Figura 4.21 foi baseada na interface desenvolvida no âmbito de outro trabalho de mestrado.

Esta interface é composta por vários campos que estão identificados pelos números:

1. Este campo mostra a hora que foi iniciada a execução do skill (**Time**), o nome da skill que está a ser executada (**Skill**), o nome do agente que executou (**Executor**) e o nome do agente que pediu a execução da mesma (**Requester**).

2. Este campo mostra o nome do produto que está a ser executado (**ID product**) e a hora que deu entrada na linha de produção (**Time**).
3. Este é semelhante ao primeiro campo, a única diferença é a hora que aquela mesma execução terminou.
4. Este é semelhante ao segundo campo, a diferença é a hora que este saiu do sistema.
5. Este campo permite a interação com o utilizador, dando possibilidade de escolher quais dos tipos de produtos (**Product1** e **Product2**) que podem ser lançados no sistema através do botão **Launch Product.**

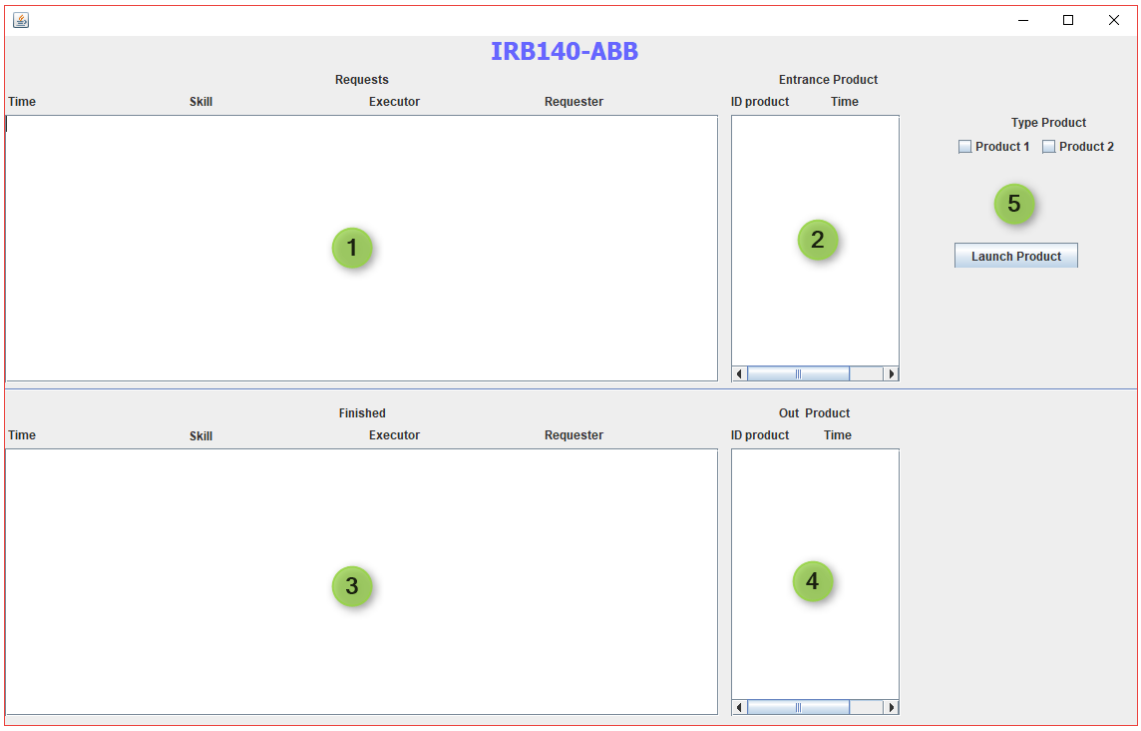


Figura 4.21 Interface de lançamento de produtos

4.2.3.2 Product Sink Agent

O PSiA é responsável pela remoção do produto do sistema. Para atingir este foi implementado a classe *ProductSinkAgent* (Figura 4.22), que dispõe os atributos e métodos necessários para reproduzirem os comportamentos pretendidos ao este agente.

O atributo **Product** contém o produto a eliminar, que será eliminado através do método **DeleteProduct()**. Este método interage de imediato com o simulador V-REP informando para eliminar o produto que se encontra no ponto de saída. Este método está definido na interface associado a classe **PSiAInterface**. Os restantes atributos que se encontram na classe, *ProductSinkAgent*, tem a mesma funcionalidade que os PSoA.

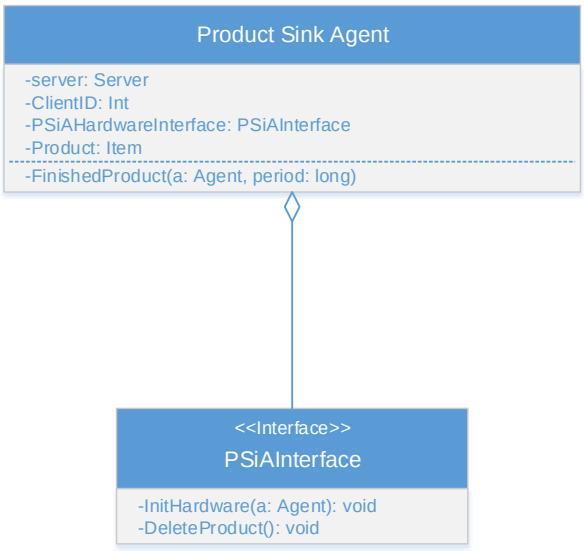


Figura 4.22 Diagrama da classe PSiA

Para implementar o PSiA foi definido a *behaviour* **FinishedProduct** do tipo *TickerBehaviour* implementado para lidar com a remoção dos produtos. Este *behaviour* aguarda pela *inform* do TEA aquando da chegada do produto, assim que recebe o produto remove do simulador V-REP através da função apresentada a verde na Figura 4.23. Posteriormente informa ao PA que a sua execução chegou ao fim e serão removidos da plataforma. Estes agentes interagem através do protocolo *FIPA Request*, no diagrama de sequências (Figura 4.23).

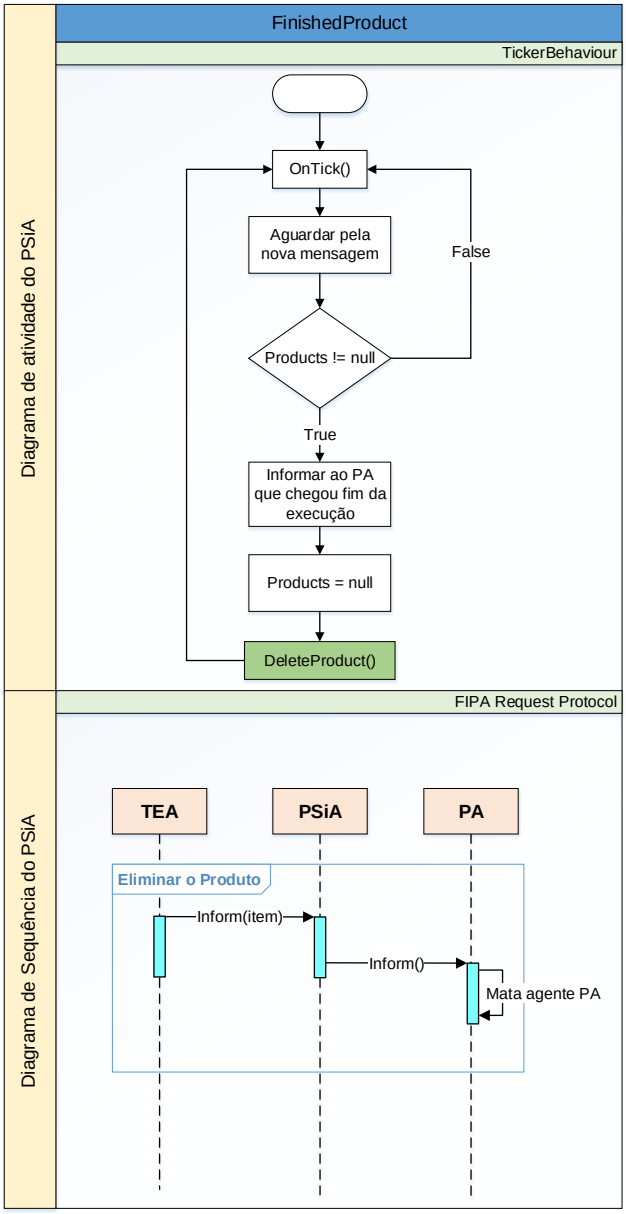


Figura 4.23 Diagrama do Behaviours do PSiA

4.2.3.3 Handover Unity Agent

Para implementar o comportamento pretendido para o HUA foi criado a classe *HandoverUnityAgent* (Figura 4.24). Nesta classe foi implementada um conjunto de atributos e métodos relevantes do comportamento do HUA.

Os atributos como **Server, ClientID** e **Product** são exatamente o mesmo que os dos PSiA. Os dois primeiros são para lidar com a interação entre os clientes e servidor e o último contém o produto (do tipo *item*) que está no encaminhado.

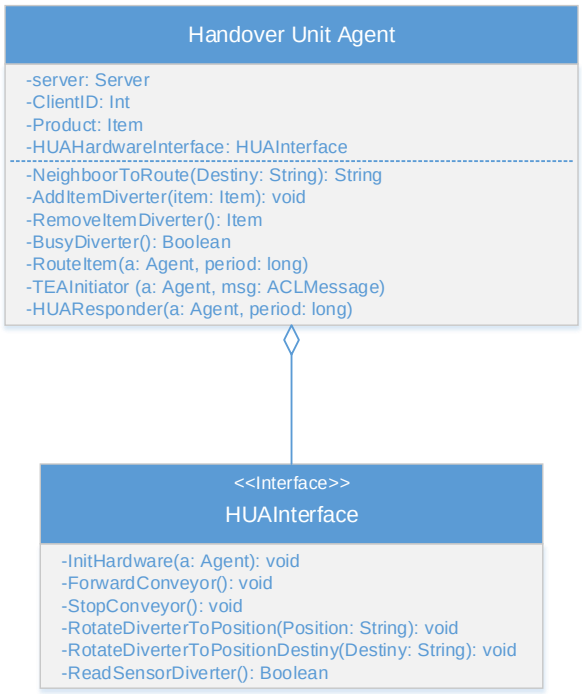


Figura 4.24 Diagrama de classe HUA

Esta classe contém uma interface **HUAInterface**, onde foi implementado um conjunto de métodos:

- **InitHardware** –inicia a interface e é chamado logo no início do lançamento do agente.
- **ForwardConveyor** – mover o encaminhador para frente, isto é, envia a velocidade positiva para o encaminhador.
- **StopConveyor** – parar o encaminhador através do envio da velocidade zero.
- **RotateDiverterToPosition** – girar o encaminhador para posição de entrada de produto.
- **RotateDiverterToPositionDestiny** – girar o encaminhador para posição de saída do produto.
- **ReadSensorDiverter** – ler o estado do sensor do encaminhador.

Como mencionado anteriormente foram definidos um conjunto de métodos que permitem o HUA desempenhar o seu comportamento, tais como:

- **NeighboorToRoute** – retornar o próximo destino que o produto deve seguir até ao destino final. Isto é, consulta-se na tabela de encaminhamento e verifica qual é o caminho mais curto que este deve seguir.
- **AddItemDiverter** - adicionar o produto no encaminhador.

- **RemoveItemDiverter** – remover o produto do encaminhador.
- **BusyDiverter** – verificar se o encaminhador está ocupado.

Para encaminhar o produto com sucesso foi implementado um conjunto de *behaviours* que permitem a transação do produto. O **RouteItem** é um *behaviour* do tipo *TickerBehaviour* foi implementado para atualizar todos estados dos encaminhadores e interagir diretamente com o simulador V-REP para encaminhar o produto. Cada decisão é tomada de acordo com os estados recebidos através dos atributos e métodos. Na Figura 4.25, a terceira coluna, ilustra um diagrama de atividade deste *behaviour*, onde as figuras geométricas a verde são os métodos que interagem sobre hardware (simulador).

O **TEAInitiator** é um *behaviour* do tipo *AchieveREInitiator,* foi implementado para iniciar pedido ao TEA para transportar o produto para o destino. As interações são feitas através do protocolo *FIPA Request*. Este *behaviour* envia um *request* ao TEA para alocar um produto à um transporte e por sua vez este vai analisando o pedido e responda, com *agree* ou *refuse*. De seguida informa-se o HUA através do *inform.* Na Figura 4.25 observa-se na segunda coluna um diagrama de atividade que ilustra o comportamento. Na mesma figura está ilustrado um diagrama de sequência que representa o modo de comunicação.

O **HUAResponder** é um *behaviour* do tipo *AchieveREResponder*, implementado para responder o pedido para o encaminhamento do produto. O pedido é feito através do protocolo de comunicação *FIPA Request*, e neste caso este *behaviour* responda com *agree* ou refuse. De seguida o TEA é informado através do *inform.* Na Figura 4.25, na primeira coluna, encontra-se diagrama de atividade do **HUAResponder** e o diagrama de sequência está representado na Figura 4.27, que responde o **HUAInitiator** implementado no TEA.

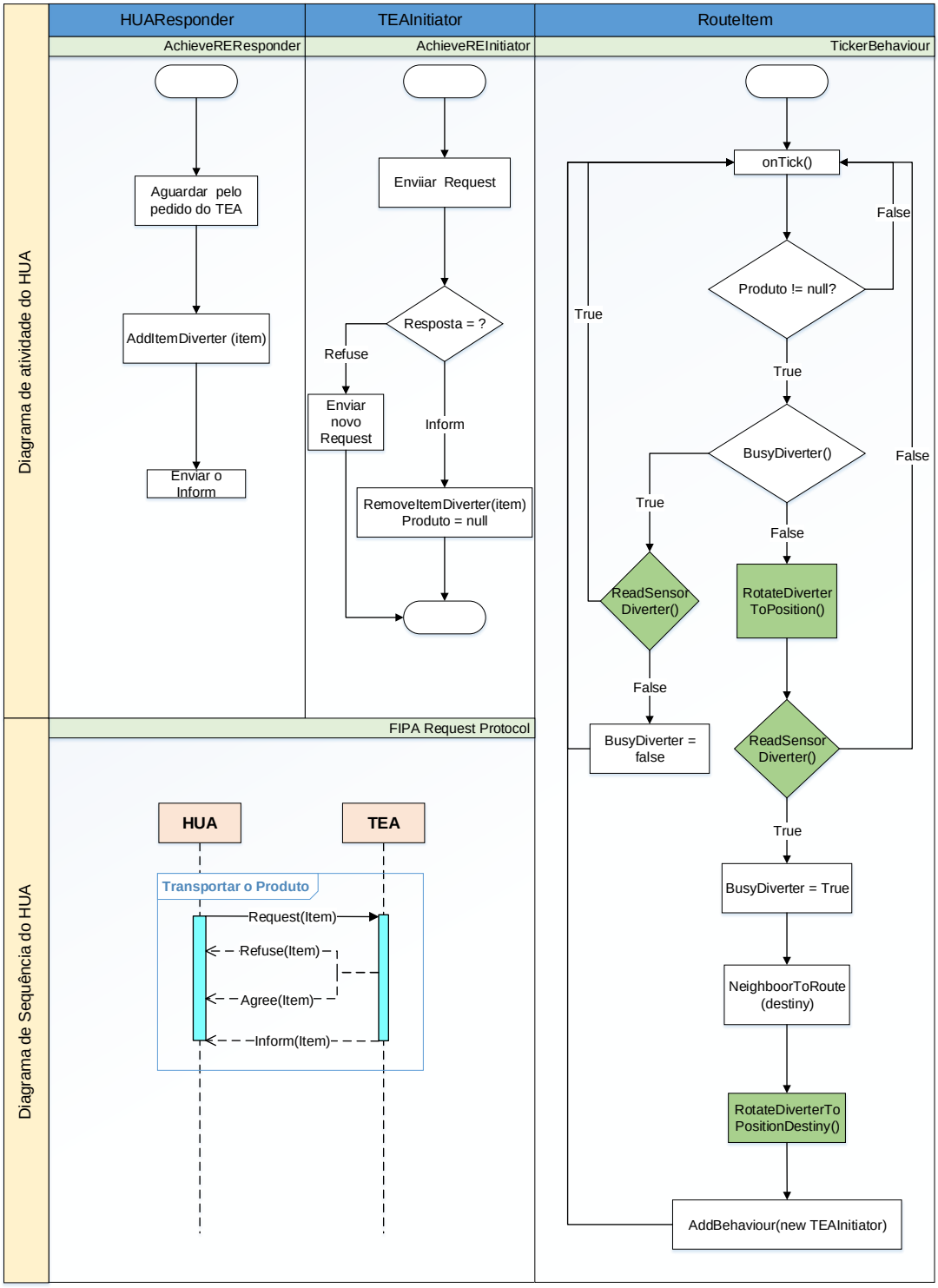


Figura 4.25 Diagrama do Behaviours de HUA

4.2.3.4 Transport Entity Agent

O TEA tem como objetivo controlar a passadeira, gerenciar os produtos de forma organizável e ainda lidar com vários produtos simultaneamente. Para implementar este comportamento foi criado a classe *TransportEntityAgent* (Figura 4.26).

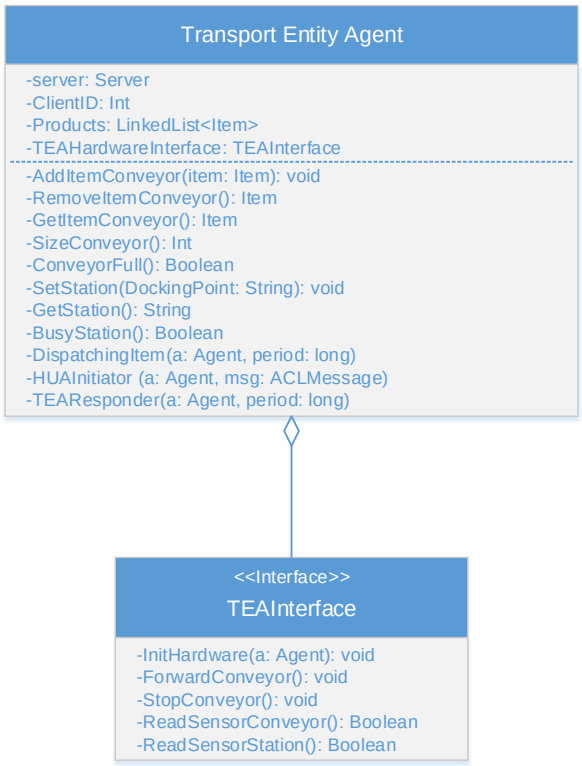


Figura 4.26 Diagrama da classe TEA

Esta classe o seu atributos não diferem muito dos HUA, praticamente as diferenças estão no atributo **Products** que neste caso é uma lista que contém todos os produtos encontrado numa passadeira, e a outra diferença está na interface associado, **TEAInterface**, que contém alguns métodos tais como:

- **ForwardConveyor** – mover a passadeira para frente (velocidade positiva).
- **StopConveyor** – parar a passadeira (velocidade zero).
- **ReadSensorConveyor** – ler o estado do sensor da passadeira se estiver presença de algum objeto
- **ReadSensorStation** – ler o estado do sensor da estação.

Os outros atributos como **Server** e **ClientID** são exatamente como os outros agentes mencionados anteriormente. Qualquer agente neste sistema interage com os módulos residentes no simulador V-REP.

A classe do TEA contém os seguintes métodos implementados:

- **AddItemConveyor –** adicionar o produto na lista *Products*.
- **RemoveItemConveyor** - remover o produto da lista *Products*.
- **GetItemConveyor –** retornar o produto contido na lista *Products*.

- **SizeConveyor** – retornar o número do produto contido na lista *Products.*
- **ConveyorFull** – verificar se a passadeira encontra-se cheio.
- **SetStation** – cria a estação na passadeira através do dockingPoint enviado pelos agentes mecatrónicos.
- **GetStation** – retornar a estação existente numa passadeira.
- **BusyStation** – verifica se a estação encontra-se ocupada.

O **TEAResponder** é um behaviour do tipo TickerBehaviour, implementado para responder os pedidos de RA's, CLA's, HUA's e PA's. Este behaviour verifica sempre se existe pedido. Na Figura 4.27, na primeira coluna apresenta-se o diagrama de atividade. Para responder os pedidos foi usado um protocolo de comunicação *FIPA Request*. Na Figura 4.27, na segunda coluna, observa-se o diagrama de sequências que mostra a forma de comunicação entre TEA e os agentes.

O **HUAInitiator** é um *behaviour* do tipo *AchieveREInitiator,* implementado para pedir ao HUA o encaminhamento do produto. Foi usado também o protocolo *FIPA Request*, como mostra a Figura 4.27.

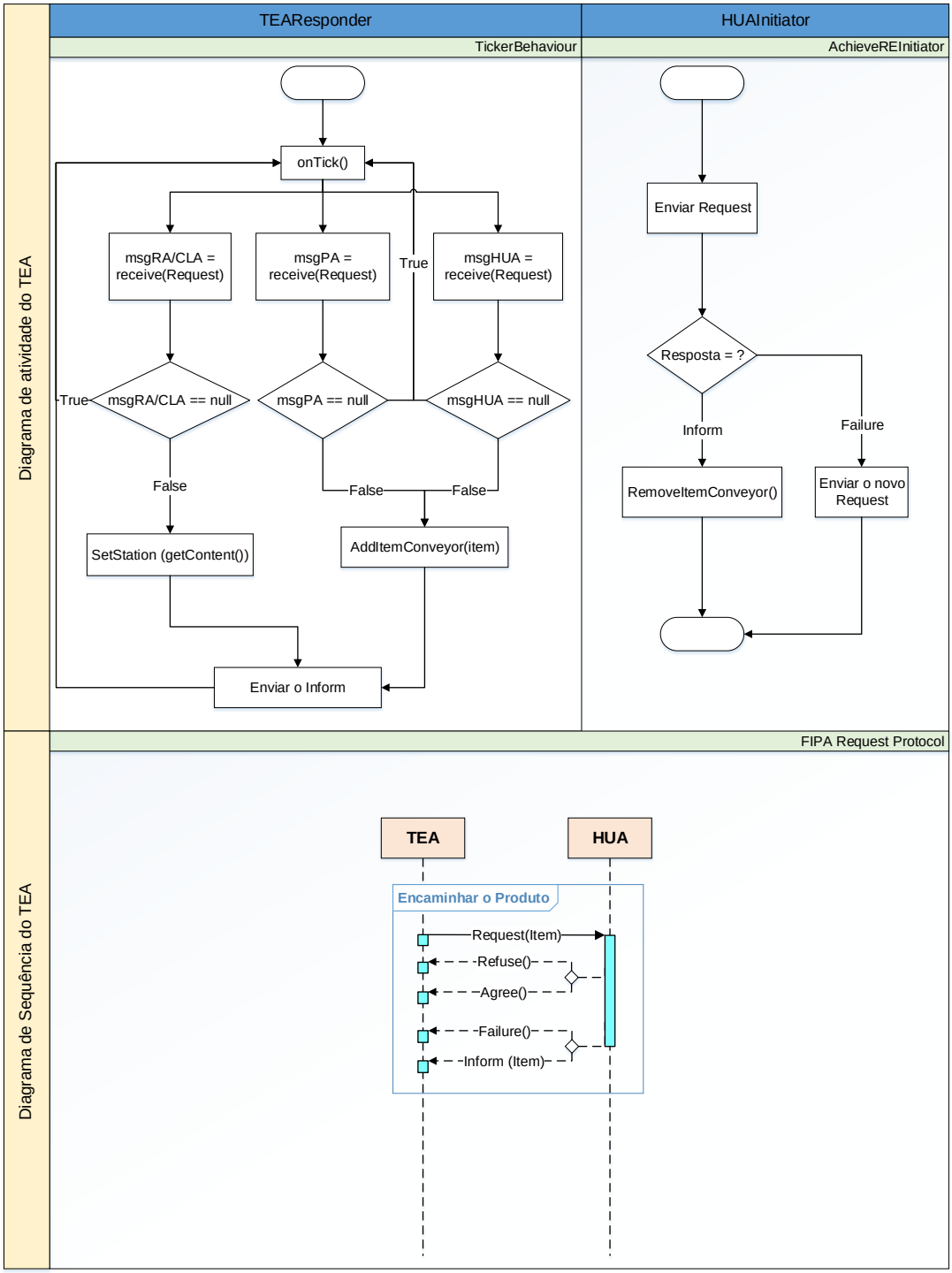


Figura 4.27 Diagrama do Behaviours de TEA

O **DispatchingItem** é um *behaviuor* do tipo *TickerBehaviour*, implementado para reproduzir os principais comportamentos do TEA. Como por exemplo a verificação do tamanho da passadeira, quantidade de produtos que a passadeira tem, verificação da existência da estação, despachar os produtos. Portanto, qualquer decisão do TEA é decidido neste *behaviour*,

onde foi implementada todas as inteligências mínimas para um bom desempenho do mesmo. Na Figura 4.28 apresenta-se o diagrama de atividade e sequências.

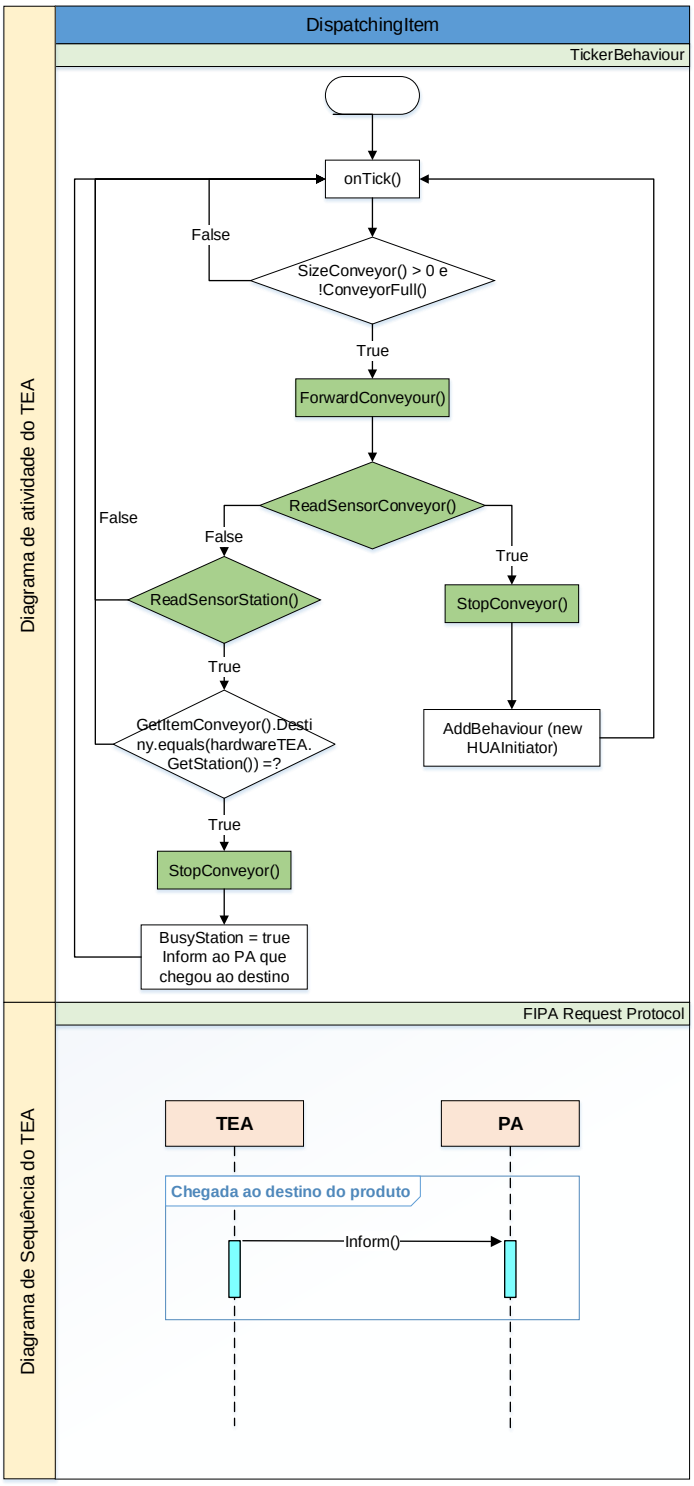


Figura 4.28 Diagrama do Behaviours de TEA dispatchingItem

4.3 Implementação de Camada de Integração

A camada de integração permite a interação entre os agentes e os seus recursos em qualquer momento da execução. Para garantir esta condição foi implementada várias funções tanto na camada de execução como na camada de simulação, que permite trocas de informações em qualquer instante do tempo.

A camada de execução solícita o pedido à camada de simulação através do comando *RemoteAPI*, dado pelo simulador V-REP. Este comando (*simxCallScriptFunction)* permite chamar a função definida no simulador V-REP. Para isso foi criado um objeto *non-threaded child* dentro do simulador V-REP, *RemoteApiCommandServer,* no qual contém um conjunto de funções definidas para simplificar as implementações. Esta decisão é tomada por causa de utilizações de alguns comandos que não existem no *RemoteAPI*. Isto é, não podem ser chamados diretamente no Java. Acresce também o facto de utilizações de comandos que só podem ser utilizados num objeto do tipo *non-threaded child,* uma vez que existe utilização do objeto do tipo *threaded child* neste projeto, que não permite certos comandos API. Para garantir o funcionamento este comando é utilizado num objeto do tipo *non-threaded child* e é enviado um sinal para *threaded child*, através do *RemoteApiCommandServer.*

A Figura 4.29 encontra-se dividida em duas colunas principais. A primeira representa as funções definidas no Java, que permite interagir com o servidor, como por exemplo o *simxCallScriptFunction.* Posteriormente, é chamado dentro da classe *Server*, no método *CallFunctionVrep* como indicada na primeira coluna da Figura 4.29. Este método tem como parâmetros de entrada ClientID, nome de objeto definido no simulador V-REP (RemoteApiCommandServer), nome da função definido dentro do objeto no simulador V-REP como por exemplo o *GotoPosition* e os restantes parâmetros são tipos de dados enviados para servidor.

Na segunda coluna da Figura 4.29 estão os objetos *child script*, como por exemplo o *RemoteApiCommandServer* que contém as funções que permite interação com *IRB140 ProductSink*, *ProductSource* e *Conveyors*. Cada um desses objetos contém um *child script* que lhe permite programar os seus comportamentos.

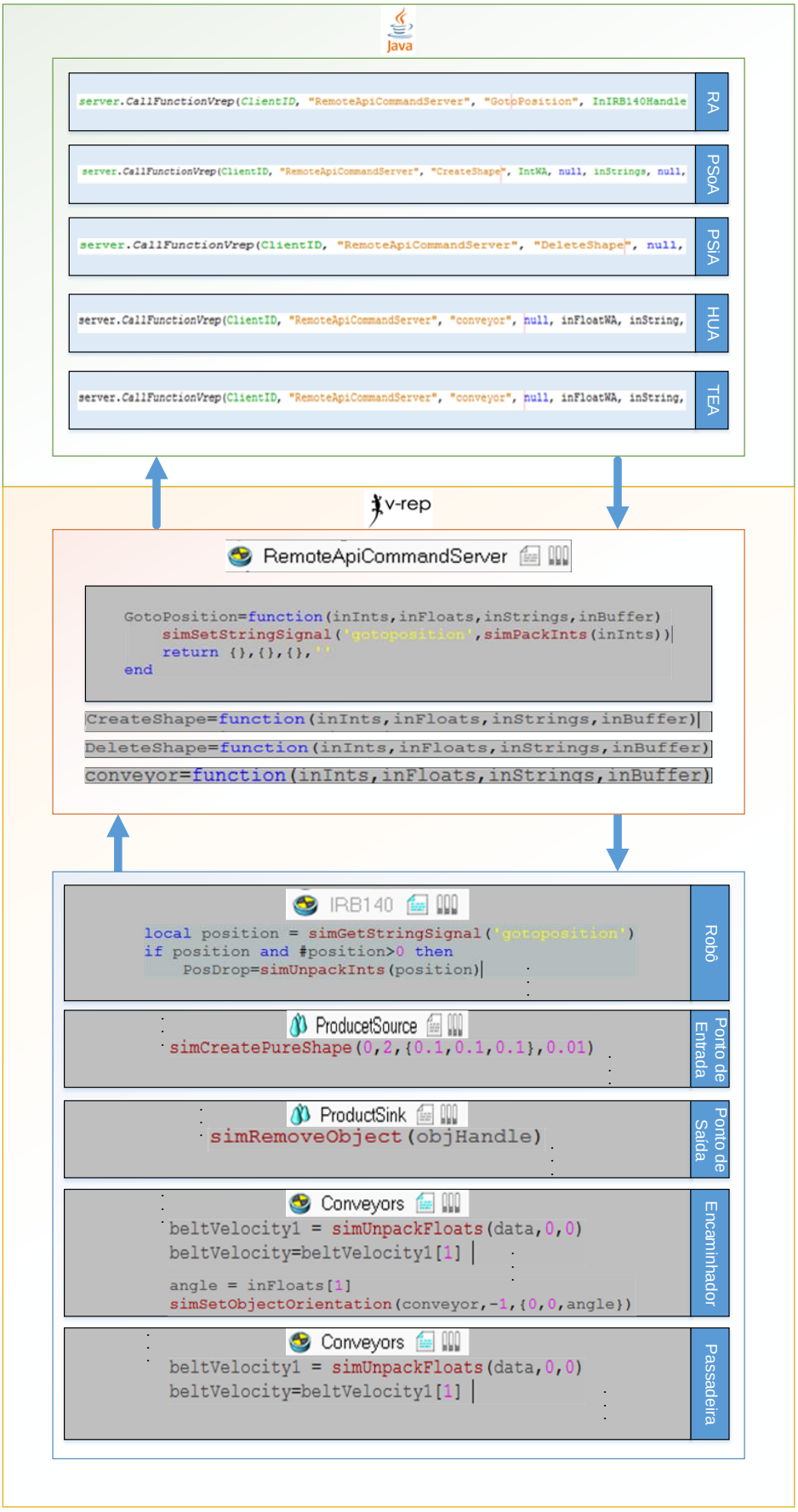


Figura 4.29 Integrações entre camadas

4.3.1 Comunicação entre camadas

A comunicação entre as camadas de execução e de simulação é feita através do sokect. Este tipo de comunicação é feito uma vez que as camadas habitem-se em software diferente, isto é, em Netbeans e no V-REP, respetivamente.

O cliente e servidor comunicam através de uma rede local, eles podem residir em computadores distintos, ou no mesmo computador. Nesta dissertação ambos estão localizados no mesmo. Na Figura 4.30 ilustra-se a comunicação feita para conexão ou desconexão entre clientes e servidor.

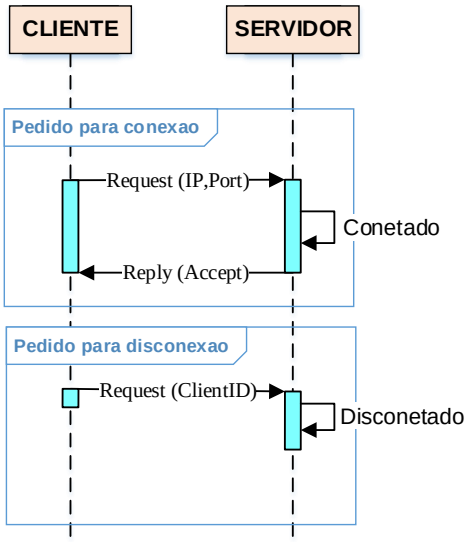


Figura 4.30 Comunicação entre clientes e Servidor

Após o lançamento dos agentes mecatrónicos, cada um deles faz pedido de conexão ao servidor com uma mensagem assíncrona, referenciado com o IP da rede onde encontram-se conectados ao servidor, e também com o número de porto das componentes (passadeiras, robôs, encaminhadores, pontos de entradas e de saídas). De seguida o servidor responde cada um dos clientes o ID exclusivo que será usado numa futura comunicação. Quando um cliente quer desconectar ao servidor, basta enviar um pedido referenciado com o ID do cliente atribuído na fase de conexão.

O comando usado para conectar o cliente e o servidor é *simxStart*. Este comando é usado para iniciar a comunicação. O *simxStart* retorna o valor -1 se a conexão falhar e caso contrário retorna o número do ID do cliente. Para desconetar os clientes do servidor o comando utilizado é *simxFinish* que termina toda a comunicação.

4.3.2 Integração dos agentes dos recursos com os seus módulos

Para interagir os RA's com os recursos (robôs), foi definido uma função na classe do agente, o método *CallFunctionVrep (ClientID," RemoteApiCommandServer"," GotoPosition",dados)* definido na interface do RA que chama a função definido no V-REP, de acordo com os parâmetros de entrada.

A função *GotoPosition* encontra-se definida dentro do *RemoteApiCommandServer* para interagir diretamente com script. Neste caso do robô *IRB140*, fica a espera do sinal enviado pelo *GotoPosition.* Este sinal é criado através do *simSetStringSignal (string signName, string signalValue).* Os parâmetros de entrada são o nome de sinal e o valor do sinal, que neste caso serão dados enviados a partir do agente, como a posição, skill, velocidade do joint e entre outros dados relevantes.

O robô *IRB140* encontra-se definido como *threaded child* que através do seu *child script* permite programar o comportamento. O robô recebe dados através da função *simGetStringSignal (string signName)*. Foram utilizados várias funções API's na implementação do *child script* do robô como por exemplo:

- *simGetObjectHandle*- obter o identificador dos objetos.
- *simGetJointPosition*- obter a posição intrínseca de um *joint*.
- *simRMLMoveToPosition*- mover vários *joints* ao mesmo tempo.

Para verificar se a execução do skill chegou ao fim, dentro do objeto *RemoteApiCommandServer* foi criada uma função *EndExecution* (Figura 4.31). Esta função recebe o sinal do fim de cada execução do skill. O valor 1 é retornado informando o fim da execução e 0 informa que ainda não chegou ao fim. Desta forma o agente sabe se o recurso encontra-se ocupado.

RemoteApiCommandServer

```
EndExecution=function(inInts,inFloats,inStrings,inBuffer)
    if(simGetIntegerSignal('endexecution')==1)then
        simClearIntegerSignal('endexecution')
        return {1},{},{},''
    end
    return {0},{},{},''
end
```

Figura 4.31 Rotina do EndExecution no V-REP

4.3.3 Integração dos agentes de transportes com os seus módulos

A integração entre os módulos de transportes e os seus agentes foi feita usado os mesmos procedimentos utilizados na integração com os módulos de recursos.

A interação do PSoA e do ponto de entrada é feita usando o método *CallFunctionVrep (ClientID," RemoteApiCommandServer"," CreatShape",dados),* definido na classe interface do PSoA. O *CreatShape* é uma função definida dentro do *RemoteApiCommandServer* para interagir com o objeto *ProductSource*. Os dados enviados do PSoA através dos parâmetros são o nome e o tipo de produto. Para a criação dos produtos são utilizados algumas funções API, como por exemplo:

- *simCreatePureShape* – criar uma forma primitiva (cubo, esfera, cilindro etc.).
- *simSetObjectPosition* – definir as coordenadas x,y e z do objeto.
- *simSetObjectName* – definir o nome do objeto.
- *simSetShapeColor* – definir a cor do objeto.

A interação do PSiA e do ponto de saída, é feita usado o método *CallFunctionVrep (ClientID," RemoteApiCommandServer"," DeleteShape",dados),* definido na classe interface do PSiA. O procedimento de implementação são praticamente igual ao *ProductSource* a diferença está na seguinte função API:

- *simRemoveObject* –remover o objeto da *scene*.

A integração entre HUA e o encaminhador e entre o TEA e a passadeira foram implementadas usando praticamente os mesmos comportamentos, a única diferença é que o encaminhador é capaz de girar em torno do seu eixo "imaginário" para que transação do produto seja realizada com sucesso. Portanto para implementar o encaminhador foi utilizado uma passadeira, dimensionado para ter apenas um produto de cada vez, ao contrário da passadeira do TEA que pode ter vários produtos. Quer HUA ou TEA chama a função *conveyors* inserido no *RemoteApiCommandServer* (Figura 4.29). Portanto, a diferença centra-se nos dados enviados pelos agentes. Estes dados contém o nome de cada objeto *conveyors*, são enviados as velocidades para passadeira assim como para o encaminhador e são enviados também os ângulos de rotação dos encaminhadores. Cada objeto *conveyors* é acessado a partir do nome através das funções API's como *simTubeOpen, simTubeWrite e simTubeRead*.

- A função *simTubeOpen* permite um tubo para comunicação dentro do V-REP. Um tubo é semelhante a um tubo de comunicação bidirecional. As mensagens escritas de um lado podem ser lidas do outro lado na mesma ordem em que foram escritas.
- A função *simTubeWrite* envia um pacote de dados para um tubo de comunicação previamente aberto pelo *simTubeOpen*.
- A função *simTubeRead* recebe um pacote de dados de um tubo de comunicação previamente aberto pelo *simTubeOpen*.
- A função *simSetObjectOrientation* permite definir a orientação (ângulos de Euler) dos objetos.

4.4 Implementação da camada de simulação

4.4.1 Modelos

Na implementação do projeto foram usados alguns modelos existentes no simulador V-REP, para simular a passadeira, encaminhador, robô, garra, ponto de entrada e de saída.

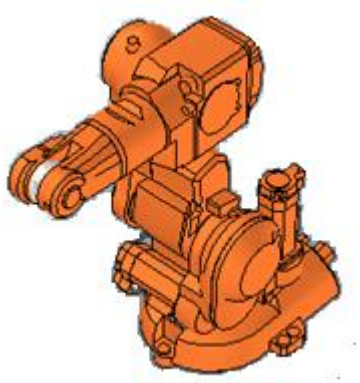


Figura 4.33 Modelo do robô ABB 140

Para simular o robô foi utilizado um modelo de robô *ABB IRB 140* (Figura 4.33). Este modelo representa um robô industrial com uma ampla gama de aplicações, com seis graus de liberdade. Este robô é compacto, polivalente e dispõe de uma tecnologia de detenção de colisão que torna-o muito seguro.

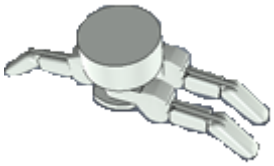


Figura 4.32 Modelo da garra Barrett Hand

Para simular a garra do robô foi utilizado um modelo *Barrett Hand* (Figura 4.32). Esta garra é muito flexível e seguro, possui várias caraterísticas como a sua firmeza no agarramento mesmo em superfície delicadas e de precisão, adapta-se aos vários tipos de objetos. Tem bom desempenho na aplicação do tipo pick and place e os seus dedos reajustam de acordo com o objeto e a integração com o braço do robô é fácil.

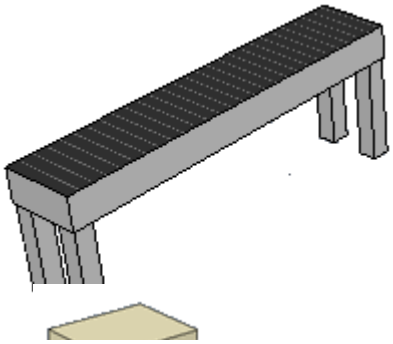


Figura 4.34 Modelo da passadeira

Para simular a passadeira foi utilizado o modelo *customizable Conveyor Belt* (Figura 4.34). Este modelo permite dimensionar o comprimento, largura, altura da passadeira. Esta passadeira é bastante dinâmica e dispõe velocidade variável.

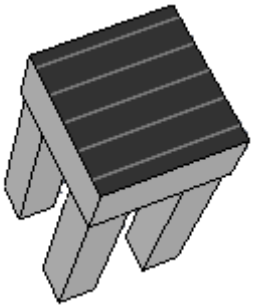


Figura 4.35 Modelo do encaminhador

Para implementar o encaminhador foi utilizado também o modelo *customizable Conveyor Belt.* Visto que o encaminhador tem comportamento idêntico a passadeira. A única diferença é que o encaminhador pode rodar e é dimensionado de uma forma que permite acomodar um produto de cada vez (Figura 4.35). A rotação foi implementada para tornar o encaminhador multidirecional, isto é, fazer a passadeira mover em todas as direções.

Para a implementação dos pontos de entrada e de saída foi utilizado um modelo encontrado no V-REP para simular a criação e a eliminação da peça no ambiente (Figura 4.36).

Figura 4.36 Modelo de ponto de entrada e de saída

5

VALIDAÇÃO E TESTES

Neste capítulo apresenta-se, inicialmente, as características dos sistemas utilizados na validação e testes da arquitetura e na parte final apresenta-se os principais resultados obtidos.

5.1 Célula NOVAFLEX

Para validar e testar arquitetura apresentada no capítulo 3, foi virtualizada a célula NOVAFLEX (Figura 5.1). Esta célula encontra-se instalada na FCT/UNL. A escolha desta célula deve-se ao facto da sua disponibilidade para os testes e também ao seu alto grau de complexidade, que possibilita vários estudos relativamente à linha de produção. A NOVAFLEX é composta por nove passadeiras interligadas, por seis encaminhadores, dois robôs, um ponto de entrada e um ponto de saída.

Figura 5.1 Célula NOVAFLEX instalada na FCT/UNL

5.2 Caracterização do sistema utilizado no teste

A implementação da célula NOVAFLEX foi feita recorrendo-se ao V-REP. Na Figura 5.2 apresenta-se a topologia da célula NOVAFLEX implementada no V-REP.

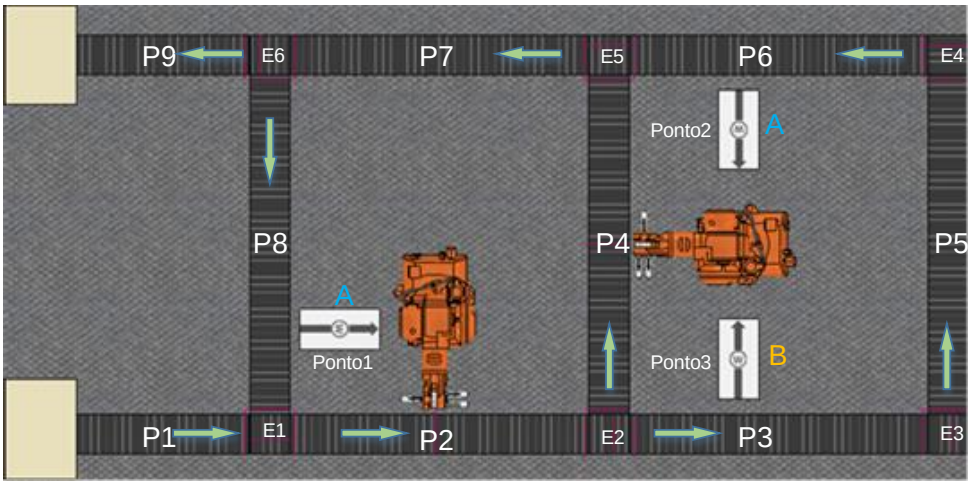


Figura 5.2 Sistema implementado no V-REP utlizado no teste

A implementação da NOVAFLEX no V-REP foi feita utilizando os modelos apresentados no capítulo 4. Estes modelos permitem simular as passadeiras, encaminhadores, robôs, pontos de entrada e saída. Estes modelos foram implementados de forma que os seus comportamentos sejam reproduzidos o mais realista possível.

O sistema apresentado na Figura 5.2 é composto por várias estações. Cada estação contém os seus skills e os pontos onde serão executadas as tarefas. As estações que estão associados aos agentes RA's ou CLA's, têm a capacidade para executarem as tarefas exigidas pelo PA. A estação acoplada ao P2 executa um único skill (A) no Ponto 1 e a estação acoplada ao P4 executa dois skills (A e B) no Ponto 2 e Ponto 3, respetivamente. No sistema também existe duas estações responsáveis pela introdução e remoção dos produtos.

Na Tabela 5.1 apresenta-se as características das estações implementadas no V-REP, como os referidos skills, passadeiras e os pontos onde são executadas os skills. É de realçar que uma estação pode estar habilitado a executar mais de que um skill.

Tabela 5.1 Características das estações

Estação (ID)	Skill	Passadeira	Ponto de Execução
1	Entrada	1	Ponto de entrada
2	Saída	9	Ponto de saída
3	A	2	Ponto 1
4	A	4	Ponto 2
	B	4	Ponto 3

Para simular a NOVAFLEX foram usados pelo menos trintas agentes apresentadas na arquitetura IADE. Estes agentes encontram-se descritos na Tabela 5.2. É de salientar que o PA não tem um número fixo de agente no sistema, pois este é lançado no tempo real, o número varia de acordo com o teste que se pretende. No entanto, para testar o sistema é necessário pelo menos um PA lançado na plataforma.

Tabela 5.2 Números de agentes utilizados na simulação da NOVAFLEX

Agentes	Número
Deployment Agent (DA)	1
Directory Facilitator (DF)	1
Resource Agent (RA)	4
Coalidition Leader Agent (CLA)	6
Product Agent (PA)	>0
Product Source Agent (PSoA)	1
Product Sink Agent (PSiA)	1
Handover Unit Agent (HUA)	6
Transport Entity Agent (TEA)	9

Para o teste da arquitetura apresentada foi utilizado dois tipos de produtos: Tipo1 e Tipo2. Na Tabela 5.3 apresenta-se os produtos utilizados no teste da arquitetura e os respetivos planos de produção. Neste teste considerou-se que cada skill do plano de produção é executado sequencial.

Tabela 5.3 Produtos utilizados nos testes

Tipo de produto (ID)	Plano de produção (skills)
Tipo 1	A
Tipo 2	BA

As passadeiras neste sistema transportam os produtos nas direções das setas apresentadas na Figura 5.2. Os encaminhadores localizados entre passadeiras desvia o produto no sentido das setas das passadeiras ligadas. As estações e os encaminhadores só podem ter um produto de cada vez, enquanto que as passadeiras podem acomodar e lidar vários produtos em simultâneos. Portanto, o tamanho das passadeiras são variáveis, e têm a dimensão do número de produtos que estão acomodados. Na Tabela 5.4 encontra-se o tamanho de cada passadeira e sua respetiva velocidade.

Tabela 5.4 Identificação e características das passadeiras utilizadas no teste

Passadeira (ID)	Velocidade (m/s)	Tamanho (produto)
1	0.08	5
2	0.08	7
3	0.08	7
4	0.08	7
5	0.08	7
6	0.08	7
7	0.08	7
8	0.08	7
9	0.08	5

Para testar a arquitetura apresentada no simulador V-REP foram definidos alguns parâmetros auxiliares tais como: o número de portos onde os clientes podem conectar e o endereço IP onde o servidor (V-REP) está localizado. Na Tabela 5.5 encontra-se um conjunto de parâmetros utilizados nos testes para estabelecer as comunicações entre os clientes (agentes) e o servidor. Por exemplo, verifica-se na Tabela 5.5 que o RA, do Robo1 está a usar o endereço **IP 127.0.0.1**, **porto 19991** para comunicar com **Robo1** e tem ID do **cliente 0**. Este ID do cliente é gerado pelo simulador durante a fase de conexão entre os clientes e o servidor.

É de realçar que os agentes de Robo1 e Robo2 foram definidos em portos diferentes. Esta decisão foi feita para evitar uma possível colisão que pode ocorrer entre os dois agentes quando acedem à um porto ao mesmo tempo. Embora o V-REP consegue gerir acessos de

portos em simultâneos, esta colisão pode ocorrer por acesso de um curto intervalo de tempo. No caso dos robôs esta colisão não ocorre uma vez que a execução de um skill não tem um tempo definido, e durante esse tempo o porto estará ocupado, ao contrário do robô.

Tabela 5.5 Exemplo das conexões entre os módulos e os agentes

Módulo	Cliente ID	Porto	IP	Agente
Robo1	0	19991	127.0.0.1	RA
Robo2	1	19992	127.0.0.1	RA
Ponto de entrada	2	19993	127.0.0.1	PSoA
Ponto de saída	3	19994	127.0.0.1	PSiA
Encaminhador	4	19995	127.0.0.1	HUA
Passadeira	5	19996	127.0.0.1	TEA

Para garantir que os agentes controlam os módulos, tanto de recursos como de transportes, foram definidos alguns skills atómicos (ASk) e complexos (CSk) utilizados pelos agentes RA's e CLA's, respetivamente. Neste teste foi usado quatros agentes instanciados do RA. Estes agentes são de dois robôs e de duas garras (grippers), em que os skills dos robôs são as movimentações dos joints e abertura e fecho das garras. Na Tabela 5.6 encontra-se apresentado todos os skills usados pelos agentes para execuções das tarefas exigidas pelo PA.

Tabela 5.6 ASk's dos RA's

Resource Agent	Skills (ASk)
Robo1	MoveJointToDropPos (MJTDP) MoveJointToPickUpPos (MJTPUP) MoveJointToUpperDropPos (MJTUDP) MoveJointToUpperPickUpPos (MJTUPUP) MoveJointToHomePos (MJTHP)
Gripper1	GripperOpen (GO) GripperClose (GC)
Robo2	MoveJointToDropPos (MJTDP) MoveJointToPickUpPos (MJTPUP) MoveJointToUpperDropPos (MJTUDP) MoveJointToUpperPickUpPos (MJTUPUP) MoveJointToDropPos1 (MJTDP1) MoveJointToPickUpPos1 (MJTPUP1) MoveJointToUpperDropPos1 (MJTUDP1) MoveJointToUpperPickUpPos1 (MJTUPUP1) MoveJointToHomePos (MJTHP)
Gripper2	GripperOpen (GO) GripperClose (GC)

Para manter o sistema organizável e menos complexo foram implementados seis agentes CLA's. Estes agentes foram divididos em três para cada área, no total duas áreas. Na Tabela 5.7 encontra-se apresentado todos os seus agentes e os seus respetivos skills. Na primeira coluna encontra-se os nomes dos agentes CLA's de cada área, na segunda coluna os seus CSk e na terceira coluna os conjuntos de ASk e CSk, denominados de subskills (skills complexos de mais alto nível). É de realçar que estes subskills estão listados por ordem da sua execução, de cima para baixo. Isto é, para o agente executar um skill complexo, como por exemplo *PickAndPlace*, o seu agente terá que pedir todos agentes responsáveis pelas execuções dos seus subskills (Pick, Place, Pick, MJTUDP, GO e MJTHP) e estes serão executados sequencialmente e individualmente.

O skill *A* existente no plano de produção do produto é o mesmo que skill *PickAndPlace* e o skill *B* é o mesmo que o skill *PickAndPlace1,* portanto pode verificar que o *PickAndPlace* pode ser executado em ambas as áreas (CLA_A_Area1 e CLA_A_Area2) e skill *PickAndPlace1* só pode ser executado na área 2 (CLA_A_Area2).

Tabela 5.7 CSk's dos CLA's

Coalidition Leader Agent	Skills (CSk)	SubSkills
CLA_A_Area1	PickAndPlace	Pick Place Pick MJTUDP GO MJTHP
CLA_B1_Area1	Pick	GO MJTUPUP MJTPUP GC MJTUPUP
CLA_B2_Area1	Place	MJTUDP MJTDP GO MJTUDP
CLA_A_Area2	PickAndPlace	Igual ao da área 1
	PickAndPlace1	Pick1 Place1 Pick1 MJTUDP1 GO MJTHP1
CLA_B1_Area2	Pick	Igual ao da área 1
	Pick1	GO MJTUPUP1 MJTPUP1 GC MJTUPUP1
CLA_B2_Area2	Place	Igual ao da área 1
	Place1	MJTUDP1 MJTDP1 GO MJTUDP1

5.3 Discussão dos Resultados

Após estabelecidas todas as especificações, inicia-se o teste no simulador V-REP para validar a arquitetura apresentada. Portanto neste subcapítulo são apresentados e discutidos os resultados dos testes efetuados.

Na Figura 5.3 está representada uma sequência de imagens que ilustra as execuções dos produtos. Na Figura 5.3a observa-se as entradas dos dois tipos de produtos, em que primeiro foi lançado o produto do Tipo 1 (cubo cinzento) e de seguida o produto do tipo 2 (cubo azul). Neste teste foram lançados num total de dez produtos (PA's) de forma alternada e progressiva.

Na Figura 5.3b observa-se que um dos produtos do Tipo 1 (que se encontra na P3) já terminou a sua tarefa (Skill A). Esta tarefa foi executada na estação 1 como era expectável, visto que poderia ser executada tanto na estação 1 como na estação 2. De seguida este produto foi enviado para o ponto de saída. Ainda pode-se observar a chegada dos produtos, do Tipo2 na estação 2, uma vez que a sua sequência de execução como antes referido é skill BA, em que o B será sempre executado na estação 2 e o skill A pode ser executado numa das estações.

Na Figura 5.3c observa-se os dez produtos no sistema. Na Figura 5.3d verifica-se a execução do produto na estação 1 (skill A). Na Figura 5.3e nota-se que um produto ainda não terminou a sua tarefa e volta para estação 1 para completar-se. Por fim, na Figura 5.3f é possível ver a conclusão de todos os produtos lançados, e o último produto está a encaminhar na direção do ponto de saída para ser eliminado do sistema.

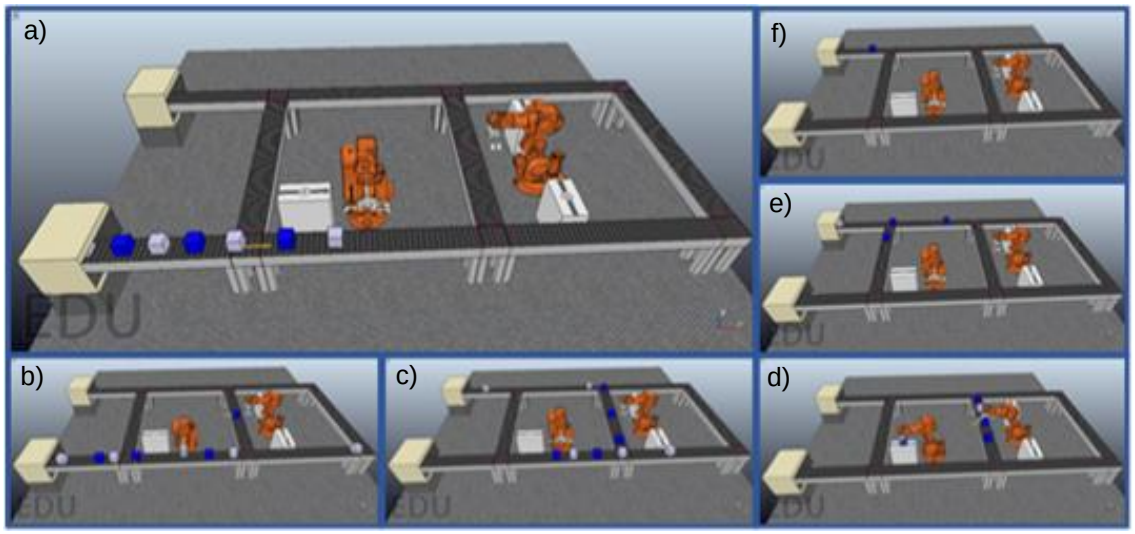


Figura 5.3 Sequência de produção de dez produtos

Na Figura 5.4 confirma-se que os dez produtos foram executados com sucessos. Todos os produtos entraram no sistema sequencialmente num curto intervalo de tempo. Esta confirmação pode ser feita através da análise dos campos I*D product* e *time* no *entrance product*. A saída dos produtos do sistema não será obrigatoriamente sequencial, uma vez que cada

produto tem o seu próprio plano de execução de skill, este facto é também confirmado na Figura 5.4 através dos campos *Out product* em que mostra o tempo de saída de cada produto.

O tempo necessário para executar os dez produtos foi de aproximadamente 15 minutos como podem confirmar a hora da entrada do primeiro produto, *product_0*, e saída do último produto, *product_7*.

Na Figura 5.4 observa-se que as ASk's foram executadas sequencialmente como definido na Tabela 5.7.

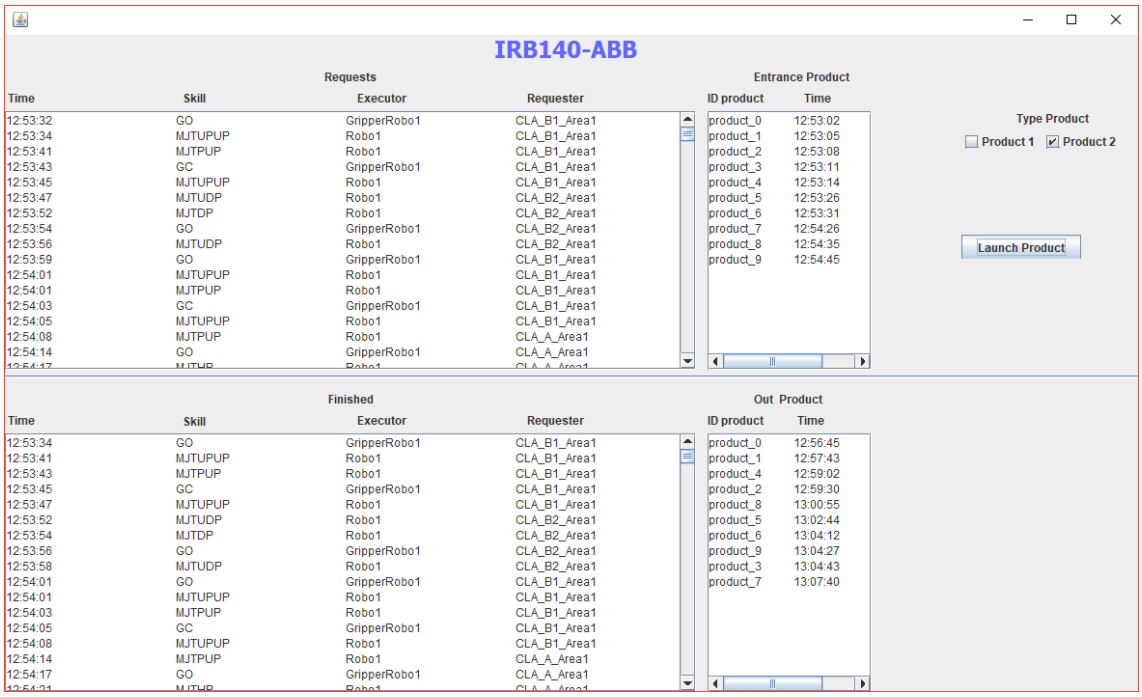
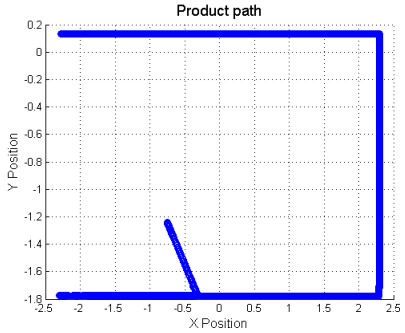


Figura 5.4 Interface do lançamento de controlo de entrada e saída dos produtos

Na Figura 5.5 apresenta-se os percursos dos produtos do Tipo 1 (executa skill A) realizadas para completar as suas tarefas. Na Figura 5.5a pode-se observar que um produto foi executado no ponto 1 da estação 1 e na Figura 5.5b é possível observar que o outro produto foi executado no ponto 2 da estação 2, como era espectável. Portanto cada produto do tipo 1 será executado na estação alternado de forma a garantir melhor distribuição das tarefas entre agentes com o mesmo skill.

a)



b)

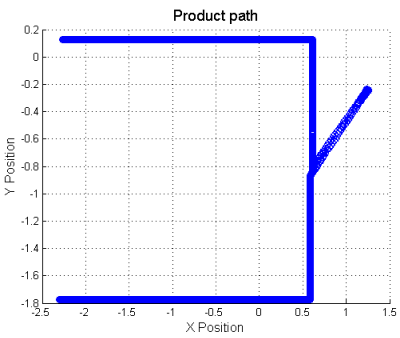


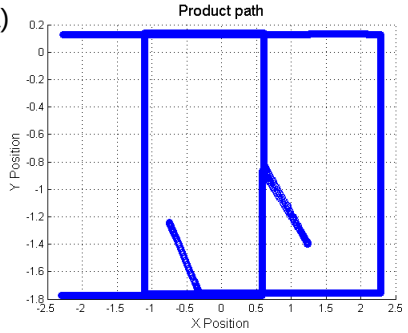
Figura 5.5 Exemplo do percurso do produto do Tipo 1

Na Figura 5.6 apresenta-se os percursos dos produtos do tipo 2 (Skill BA) para completar as suas tarefas. Como antes referido o skill B só será executado no ponto 3 da estação 2. Qualquer produto que tenha esse skill passa pela estação 2. Esta condição pode ser confirmada na Figura 5.6. Verifica-se que sempre foi executado skill B na estação 2 no ponto 3. Na Figura 5.6a verifica-se que inicialmente foi executada o skill B na estação 2 e posteriormente o Skill A na estação 1. Portanto como o produto encontra-se na estação 2 e deslocou-se até à estação 1 para completar-se a sua tarefa e só depois seguir para à saída.

Na Figura 5.6b à direita é possível observar que numa única estação (estação 2) foi executado dois skills (B e A) de forma sequencial. O skill B foi executado na estação 2 no ponto 3 e o skill A desta vez foi executado no ponto 2 da estação 2.

É de realçar que quando um produto se encontra numa estação que é capaz de executar mais de que um skill, e se esses skills foram decididos para serem executados naquela estação, o produto simplesmente efetua a sua execução de forma sequencial antes de sair para outro destino. Esta decisão é feita devido à implementação do transporte, em que o agente PA ao pedir o agente TEA para transportar o produto até ao destino este por sua vez vai informar ao agente PA que o produto encontra-se no mesmo destino. Sendo assim, o agente PA simplesmente pede apenas aos agentes responsáveis para procederam as execuções das tarefas.

a)



b)

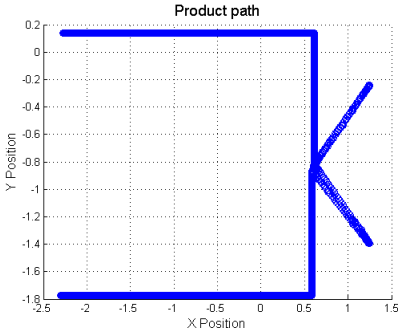


Figura 5.6 Exemplo do percurso do produto Tipo 2

As linhas oblíquas Figura 5.6 são os percursos que os produtos fazem da estação até ao ponto de execução através do robô de cada estação.

Para verificar que este sistema é capaz de se adaptar a qualquer tipo de produto, foi criado um novo plano de execução que requer mais skills para ser completadas todas as tarefas. Neste caso foi usado o seguinte plano com a sequência de skills, ABA. Para comprovar a flexibilidade e adaptabilidade do sistema foi lançado apenas um produto. Nas Figura 5.7 a e b mostram-se a entrada e a chegada do produto na estação 1, respetivamente. Na Figura 5.7c observa-se o produto a executar o skill A no ponto 1. Na Figura 5.7d na imagem 4 é possível observar o produto a ser executado o skill B no ponto 3 na estação 2. Na Figura 5.7e mostra-se a execução do skill A (a última para completar a tarefa) que está a ser executado na mesma estação do skill anterior, mas no ponto 2. Após a execução, o produto segue para o destino final, ponto de saída, como pode ser observado nas Figura 5.7f, g e h. Na Figura 5.7 f mostra-se a saída do produto da estação, na Figura 5.7g observa-se o produto no encaminhador a ser enviado para P7 e na Figura 5.7h apresenta-se o produto a seguir para ponto de saída.

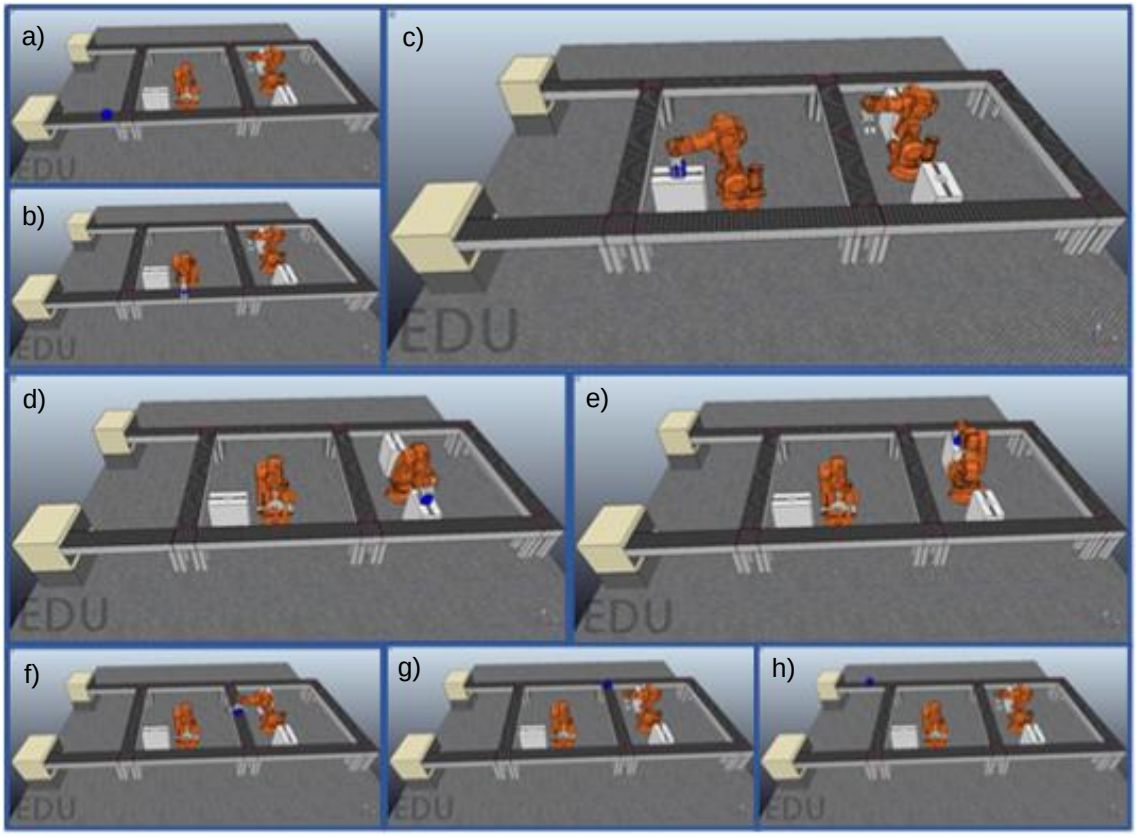


Figura 5.7 Produção de um novo tipo de produto (ABA)

Na Figura 5.8 é possível confirmar em planta e isométrica, o percurso do produto para completar as tarefas. Na Figura 5.8a pode ver que o produto passou pelo ponto 1 na estação 1 onde foi executada o skill A e de seguida para ponto 3 na estação 2 onde foi executada o skill B

e por fim na mesma estação o ponto 2 no qual foi executada o skill A e seguiu-se para a saída., Na Figura 5.8b observa-se o percurso do produto feito através dos robôs.

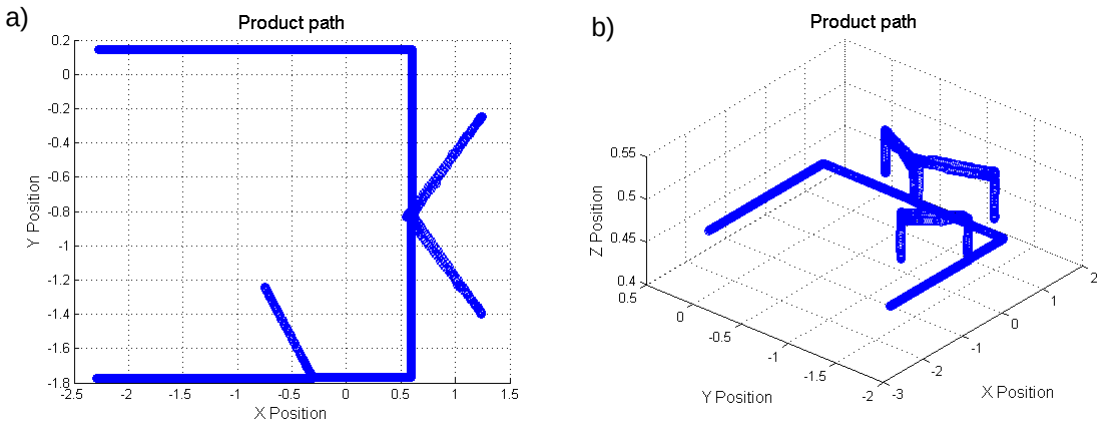


Figura 5.8 Percurso de um produto para executar a sequência ABA

De forma global conclui-se que o sistema é capaz de se adaptar ao qualquer tipo de produtos. Portanto o sistema garante maior flexibilidade e personalização na produção dos produtos.

6

CONCLUSÕES E TRABALHO FUTURO

6.1 Conclusão

A utilização de ferramentas de simulação num sistema de manufatura tem-se tornando cada vez mais frequente no teste, simulação, prevenção e desenvolvimento de novas sistemas. No entanto nem sempre é fácil a implementação de uma nova arquitetura nos sistemas de manufatura sem esta passar por uma validação prévia, que comprove da sua funcionalidade e da melhoria antes de ser implementado no mundo real. Neste contexto, o objetivo principal deste trabalho foi desenvolver uma solução de simulação para sistemas controlados por sistemas multiagente. Esta solução foi integrada com um simulador de excelência, V-REP. Os modelos utilizados para simular uma linha de produção foram abstraídos por agentes no MAS. Este método garante maior flexibilidade, adaptabilidade, autonomia e inteligência no controlo das linhas de produções.

O V-REP demostrou ser um simulador flexível, versátil, escalável e compatível graça a sua arquitetura distribuída que permite de forma eficaz a integração com sistema MAS.

Os resultados comprovam que uma arquitetura distribuída baseado no MAS pode ser integrada de forma eficaz com o V-REP. Os resultados dos testes apresentados no capítulo 5 sugerem que a arquitetura proposta funciona de uma forma consistente, independentemente da complexidade dos planos de produção. Da análise do percurso dos produtos mostrou-se que o sistema desenvolvido é auto-organizável e adaptável aos produtos. Portanto, estes resultados comprovam que é possível integrar duas tecnologias distintas para funcionar como um só.

6.2 Trabalho Futuro

Em termo de sugestões para trabalhos futuros têm-se algumas perspetivas que possam ser estudadas. A primeira perspetiva baseia-se na adaptação da arquitetura proposta num outro simulador de sistema de manufatura existente no mercado, uma vez que neste trabalho esta arquitetura foi implementada de forma genérica. Adaptar outra arquitetura baseado no MAS para o V-REP e utilizando os mesmos componentes como foram programados.

A segunda perspetiva seria abstrair cada *joint* do robô à um agente diferente, em vez de ter cada robô ao um agente. Portanto, um robô passava a ser controlado por vários agentes e teriam maior controlo e autonomia. Uma das tarefas que podia ser atribuído ao agente seria calcular os seus valores das coordenadas através da cinemática inversa. Com essas distribuições dos agentes o sistema poderia ficar com granularidade mais fina e teriam maior controlo.

A terceira perspetiva seria implementar a arquitetura apresentada num sistema real, NOVAFLEX, e comparar os resultados obtidos com resultados da simulação.

A última perspetiva seria integrar o MAS e o ROS para controlar o sistema implementado no V-REP, utilizando o mecanismo de comunicação ROS interface disponibilizado pelo V-REP, e posteriormente comparar as eficiências deste em relação ao sistema que foi implementado nesta dissertação.

7

REFERÊNCIAS BIBLIOGRÁFICAS

Abdelkader, A., Bakhta, N., & Abdelkader, O.-M. (2012). Multi-Agents Model for Web-based Collaborative Decision Support Systems, *867*, 294–299. Retrieved from http://jcheminf.springeropen.com/articles/10.1186/s13321-016-0132-8

Abdi, M. R. (2005). Selection of a layout configuration for reconfigurable manufacturing system using the AHP. *International Journal of Manufacturing Technology and Management*, *17*(Number 1-2/2009), 149–165.

Banks, J. (1999). Introduction to simulation. In *WSC'99. 1999 Winter Simulation Conference Proceedings. "Simulation - A Bridge to the Future" (Cat. No.99CH37038)* (Vol. 1, pp. 7–13). IEEE. http://doi.org/10.1109/WSC.1999.823046

Bellifemine, F., Caire, G., & Greenwood, D. (2007). *developing multi-agent systems with jade*.

Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE–A FIPA-compliant agent framework. *Proceedings of PAAM*, 97–108. http://doi.org/10.1145/375735.376120

Borshchev, A., & Filippov, A. (2004). From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools. *22nd International Conference of the System Dynamics Society, 25-29 July 2004*, 45.

Celes, W., Figueiredo, L. H. De, & Ierusalimschy, R. (2003). A Linguagem Lua e suas Aplicacações em Jogos, 1–19.

Christensen, J. H. (1994). Holonic Manufacturing ystem: Initial architecture and standards directions - Presented at First European Conference on Holonic Manufacturing Systems, Hannover, Germany, 1 December 1994., (March), 1–20.

Coppelia Robotics. (2017a). Regular API function list. Retrieved June 6, 2017, from http://www.coppeliarobotics.com/helpFiles/en/apiFunctionListAlphabetical.htm

Coppelia Robotics. (2017b). V-Rep. Retrieved July 1, 2017, from http://coppeliarobotics.com/assets/v-repoverviewpresentation.pdf

Coppelia Robotics. (2017c). V-REP. Retrieved July 1, 2017, from http://www.coppeliarobotics.com/helpFiles/en/userInterface.htm#SceneHierarchy

Coppelia Robotics. (2017d). V-REP. Retrieved July 1, 2017, from http://www.coppeliarobotics.com/helpFiles/en/objects.htm

Coppelia Robotics. (2017e). V-rep. Retrieved July 1, 2017, from http://www.coppeliarobotics.com/helpFiles/en/proximitySensorDescription.htm

Coppelia-Robotics. (2016). V-rep virtual robot experimentation platform. Retrieved November 15, 2016, from http://www.coppeliarobotics.com/

El Maraghy, H. A. (2006). Flexible and reconfigurable manufacturing systems paradigms. *Flexible Services and Manufacturing Journal*, *17*(4 SPECIAL ISSUE), 261–276. http://doi.org/10.1007/s10696-006-9028-7

Eldardiry, O. M. (2000). Usability of reconfigurable manufacturing systems. *Proceedings of the 41st International Conference on Computers & Industrial Engineering*, 205–210.

FIPA. (2002a). FIPA ACL Message Structure Specification. Retrieved June 25, 2017, from http://www.fipa.org/specs/fipa00061/XC00061F.html

FIPA. (2002b). FIPA ACL Message Structure Specification. Retrieved June 25, 2017, from http://www.fipa.org/specs/

Flexsim. (2017). Flexim. Retrieved May 27, 2017, from https://www.flexsim.com

Forouzan A., B. (2010). *TCP/IP Protocol Suite*.

Forrester, J. W. (1958). Industrial Dynamics: A Major Breakthrough for Decision Makers. *Harvard Business Review*, *36*, 37–66.

Gebus, S., Martin, O., Soulas, A., & No, R. A. (2004). *Control Engineering Laboratory Production Optimization on Pcb Assembly Lines Using Discrete-Event Simulation*.

Gordon, G. (1961). A general purpose systems simulation program. In *Proceedings of the December 12-14, 1961, eastern joint computer conference: computers - key to total systems control on - AFIPS '61 (Eastern)* (pp. 87–104). New York, New York, USA: ACM Press. http://doi.org/10.1145/1460764.1460768

Gunal, M. M. (2012). A guide for building hospital simulation models. *Health Systems*, *1*(1), 17–25. http://doi.org/10.1057/hs.2012.8

Ierusalimschy, R. (2009). Uma Introdução à Programação em Lua, 1–39.

Jahromi, M. H. M. A., & Tavakkoli-Moghaddam, R. (2012). A novel 0-1 linear integer programming model for dynamic machine-tool selection and operation allocation in a flexible manufacturing system. *Journal of Manufacturing Systems*, *31*(2), 224–231. http://doi.org/10.1016/j.jmsy.2011.07.008

Kanaga, E. G. M., & Valarmathi, M. L. (2012). Multi-agent based patient scheduling using particle swarm optimization. *Procedia Engineering*, *30*(2011), 386–393. http://doi.org/10.1016/j.proeng.2012.01.876

Kapil Kumar, M., & Pramod Kumar, J. (2014). Responsiveness Measurement of Reconfigurable Manufacturing System (pp. 373–382). http://doi.org/10.2507/daaam.scibook.2014.30

Khan, M. W., & Wang, J. (2017). The research on multi-agent system for microgrid control and optimization. *Renewable and Sustainable Energy Reviews*, *80*(February), 1399–1411. http://doi.org/10.1016/j.rser.2017.05.279

Koestler, A. (1967). *The Ghost in the Machine*. New York: random House.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., & Van Brussel, H. (1999). Reconfigurable Manufacturing Systems. *CIRP Annals - Manufacturing Technology*, *48*(2), 527–540. http://doi.org/10.1016/S0007-8506(07)63232-6

Koren, Y., & Shpitalni, M. (2010). Design of reconfigurable manufacturing systems. *Journal of Manufacturing Systems*, *29*(4), 130–141. http://doi.org/10.1016/j.jmsy.2011.01.001

Leitão, P., Colombo, A. W., & Restivo, F. (2003). An Approach to the Formal Specification of Holonic Control Systems. *Holonic and Multi-Agent Systems for Manufacturing SE - 6*, *2744*,

59–70. http://doi.org/10.1007/978-3-540-45185-3_6

Li, X., Zhang, C., Gao, L., Li, W., & Shao, X. (2010). An agent-based approach for integrated process planning and scheduling. *Expert Systems with Applications*, *37*(2), 1256–1264. http://doi.org/10.1016/j.eswa.2009.06.014

LUA. (2017). A Linguagem de Programação LUA. Retrieved July 2, 2017, from https://www.lua.org/portugues.html

Maria, A. (1997). Introduction to modeling and simulation. In *Proceedings of the 29th conference on Winter simulation - WSC '97* (pp. 7–13). New York, New York, USA: ACM Press. http://doi.org/10.1145/268437.268440

Marshall, D. A., Burgos-Liz, L., Ijzerman, M. J., Osgood, N. D., Padula, W. V., Higashi, M. K., … Crown, W. (2015). Applying dynamic simulation modeling methods in health care delivery research - The SIMULATE checklist: Report of the ISPOR simulation modeling emerging good practices task force. *Value in Health*, *18*(1), 5–16. http://doi.org/10.1016/j.jval.2014.12.001

Mitchell, M., Oldham, J., & Samuel, A. (2001). *Advanced Linux Programming*.

Mittal, K. K., & Jain, P. K. (2014). An overview of performance measures in reconfigurable manufacturing system. *Procedia Engineering*, *69*, 1125–1129. http://doi.org/10.1016/j.proeng.2014.03.100

Monostori, L., Kumara, S. R. T., & Váncza, J. (2006). Agent-Based Systems for Manufacturing. *CIRP Annals - Manufacturing Technology*, *55*(2), 697–720. http://doi.org/10.1016/j.cirp.2006.10.004

Morgan, J. S., Howick, S., & Belton, V. (2017). A toolkit of designs for mixing Discrete Event Simulation and System Dynamics. *European Journal of Operational Research*, *257*(3), 907–918. http://doi.org/10.1016/j.ejor.2016.08.016

Nazarian, E., Ko, J., & Wang, H. (2010). Design of multi-product manufacturing lines with the consideration of product change dependent inter-task times, reduced changeover and machine flexibility. *Journal of Manufacturing Systems*, *29*(1), 35–46. http://doi.org/10.1016/j.jmsy.2010.08.001

NetBeans IDE. (2017). Welcome to the NetBeans Community. Retrieved June 20, 2017, from https://netbeans.org/

Neves, P., & Barata, J. (2009). Evolvable production systems: Approach towards economical and ecological production systems. *2009 IEEE International Symposium on Assembly and Manufacturing, ISAM 2009*, (November), 189–195. http://doi.org/10.1109/ISAM.2009.5376907

Onori, M., & Barata, J. (2009). EVOLVABLE PRODUCTION SYSTEMS : MECHATRONIC PRODUCTION EQUIPMENT WITH PROCESS-BASED DISTRIBUTED CONTROL. *IFAC Proceedings Volumes*, *42*(16), 80–85. http://doi.org/10.3182/20090909-4-JP-2010.00016

Pach, C., Berger, T., Sallez, Y., & Trentesaux, D. (2015). Reactive control of overall power consumption in flexible manufacturing systems scheduling: A Potential Fields model. *Control Engineering Practice*, *44*, 193–208. http://doi.org/10.1016/j.conengprac.2015.08.003

Pegden, C. D. (2007). Simio: A new simulation system based on intelligent objects. In *2007 Winter Simulation Conference* (pp. 2293–2300). IEEE. http://doi.org/10.1109/WSC.2007.4419867

Peralta, E., Fabregas, E., Farias, G., Vargas, H., & Dormido, S. (2016). Development of a Khepera IV Library for the V-REP Simulator. *IFAC-PapersOnLine*, *49*(6), 81–86. http://doi.org/10.1016/j.ifacol.2016.07.157

Ribeiro, L., & Barata, J. (2011). 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.

http://doi.org/10.1016/j.compind.2011.03.001

Ribeiro, L., Barata, J., Onori, M., & Hoos, J. (2015). Industrial Agents for the Fast Deployment of Evolvable Assembly Systems. In *Industrial Agents* (pp. 301–322). Elsevier. http://doi.org/10.1016/B978-0-12-800341-1.00017-6

Ribeiro, L., Rocha, A., & Barata, J. (2012). A product handling technical architecture for multiagent-based mechatronic systems. In *IECON Proceedings (Industrial Electronics Conference)* (pp. 4342–4347). http://doi.org/10.1109/IECON.2012.6389190

Rockwell Software. (2017). Arena. Retrieved May 26, 2017, from http://www.erlang.com.br/arena.asp

Saba, D., Laallam, F. Z., Hadidi, A. E., & Berbaoui, B. (2015). Contribution to the Management of Energy in the Systems Multi Renewable Sources with Energy by the Application of the Multi Agents Systems "mAS." *Energy Procedia*, *74*, 616–623. http://doi.org/10.1016/j.egypro.2015.07.792

Shannon, R. E. (1998). Introduction to the art and science of simulation. In Intergovernmental Panel on Climate Change (Ed.), *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)* (Vol. 1, pp. 7–14). Cambridge: IEEE. http://doi.org/10.1109/WSC.1998.744892

Shivanand, h k, Benal, M. ., & Koti, V. (2006). Flexible manufacturing system. In *New Age International* (pp. 1–17). New Age International. Retrieved from www.newagepublishers.com

Simio. (2017). SIMIO LLC. Retrieved May 25, 2017, from https://www.simio.com

Sturrock, D. T., & Pegden, C. D. (2010). Recent innovations in Simio. In *Proceedings of the 2010 Winter Simulation Conference* (pp. 21–31). IEEE. http://doi.org/10.1109/WSC.2010.5679177

Suda. (1989). Future Factory System Formulated in Japan. *Techno Japan*, *22*(10), 15–25.

Sumari, S., & Ibrahim, R. (2013). Comparing Three Simulation Model Using Taxonomy: System Dynamic Simulation, Discrete Event Simulation and Agent Based Simulation. *International Journal of …*, *1*(3), 4–9. Retrieved from http://ijmeonline.com/index.php/ijme/article/view/1300000009

Tharumarajah, A., Wells, A. J., & Nemes, L. (1998). Comparison of emerging manufacturing concepts. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)* (Vol. 1, pp. 325–331). IEEE. http://doi.org/10.1109/ICSMC.1998.725430

Ueda, K. (1992). A concept for bionic manufacturing systems based on DNA-type information. In *Human Aspects in Computer Integrated Manufacturing* (pp. 853–863). Elsevier. http://doi.org/10.1016/B978-0-444-89465-6.50078-8

Ueda, K. (2007). Emergent synthesis approaches to Biological Manufacturing Systems. *Digital Enterprise Technology: Perspectives and Future Challenges*, 25–34. http://doi.org/Doi 10.1007/978-0-387-49864-5_3

Ueda, K., Hatono, I., Fujii, N., & Vaario, J. (2000). Reinforcement Learning Approaches to Biological Manufacturing Systems. *CIRP Annals - Manufacturing Technology*, *49*, 343–346. http://doi.org/10.1016/S0007-8506(07)62960-6

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., & Peeters, P. (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, *37*(3), 255–274. http://doi.org/10.1016/S0166-3615(98)00102-X

van Leeuwen, E. H., & Norrie, D. (1997). Holons and Holarchies. *Intelligent Manufacturing*, 86–88. http://doi.org/10.1049/me:19970203

Vik, P., Dias, L., Pereira, G., Oliveira, J., & Abreu, R. (2010). Using SIMIO for the specification of

an integrated automated weighing solution in a cement plant. *Proceedings - Winter Simulation Conference*, 1534–1546. http://doi.org/10.1109/WSC.2010.5678915