Jorge Emanuel Marçal Martins

Licenciado em Ciências da Engenharia Electrotécnica e de Computadores

Sistemas Multiagente para Componentes Mecatrónicos: Optimização e Análise

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

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

Júri

Presidente: Doutor Luís Augusto Bica Gomes de Oliveira
Arguente: Doutor João Paulo Branquinho Pimentão
Vogal: Doutor José António Barata de Oliveira

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2017

Sistemas Multiagente para Componentes Mecatrónicos: Optimização e Análise

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Para Andreia e Família

Agradecimentos

Nesta secção gostaria de expressar os meus agradecimentos a todos aqueles que contribuíram para a realização desta dissertação.

Quero agradecer em primeiro lugar ao meu orientador, Professor Doutor José Barata, por me ter dado a oportunidade de desenvolver esta dissertação no âmbito do projeto IDEAS.

Quero também agradecer ao André Rocha e ao Luís Ribeiro, por toda a disponibilidade e pela grande ajuda durante a realização deste trabalho.

Não posso deixar de demonstrar a minha gratidão para com Jorge Boavida e o João Santos, sempre disponíveis em todos os momentos, sem exceções. Com eles passei muitos dos meus dias, durante o meu percurso académico.

Um agradecimento para a Andreia, por toda a paciência e apoio ao longo desta etapa, principalmente na fase final.

Por fim, agradeço à minha família, em especial aos meus pais e irmão, por todo o apoio dado e pela confiança demonstrada. Permitindo que este percurso seja bem-sucedido.

Resumo

Os sistemas de manufatura, na sua origem, tinham como objetivo produzir em massa produtos padronizados. No entanto, a diminuição dos ciclos de vida de muitas gamas de produtos e o aumento da diversidade e exigência do mercado, levaram ao aparecimento de novas abordagens nos sistemas de manufatura.

Com estes novos sistemas de manufatura, têm surgido novos paradigmas que apresentam como principais características a robustez, reconfigurabilidade e a flexibilidade. Estes novos sistemas baseiam-se em sistemas de controlo distribuídos, ao contrário dos tradicionais que se baseiam em sistemas centralizados.

Muitos destes novos paradigmas usam tecnologia baseada em agentes. Estes novos paradigmas, quando bem implementados, apresentam um desempenho flexível, robusto, adaptativo e tolerante a falhas.

Este trabalho faz uma análise ao sistema multiagente IADE, desenvolvido no projeto FP7-IDEAS, com o objetivo de melhorar o seu desempenho, escalabilidade e usabilidade.

A arquitetura proposta apresenta uma solução com especial foco nas Skills, que são uma ontologia usada pelo IADE, e no seu processo de execução.

Esta arquitetura foi testada num ambiente virtual e o seu desempenho foi comparado com o desempenho da arquitetura em estudo.

Palavras-chave: Manufatura, Sistemas Distribuídos, Sistemas Multiagente, Reconfigurabilidade

Abstract

Manufacturing systems, at their origin, were intended to mass produce standardized products. However, decreasing product lifecycles and increasing diversity and market demand have led to new approaches in manufacturing systems.

With these new manufacturing systems, new paradigms have appeared that present as main characteristics the robustness, reconfigurability and flexibility. These new systems are based on distributed control systems, unlike traditional systems based on centralized systems.

Many of these new paradigms use agent-based technology. These new paradigms, when well implemented, present a flexible, robust, adaptive and fault tolerant performance.

This work analyzes the multi-agent system developed in the FP7-IDEAS project, aiming to improve its performance, scalability and usability.

The proposed architecture presents a solution with special focus on the Skills and in their execution process, which are one of the basic concepts composing this multiagent system.

This architecture was tested in a virtual environment and its performance was compared with the performance of the architecture being studied.

Keywords: Manufactuting, Distributed Systems, Multiagent Systems, Reconfigurability

Índice

Lista de Figuras

Lista de Tabelas

Siglas

ASk	Atomic Skill.
BBS	Black Board System.
BMS	Bionic Manufacturing Systems.
CA	Coordenating Agents.
CC	Coordinator Component.
CLA	Coalition Leader Agent.
CSk	Composite Skill.
DA	Deployment Agent.
DSk	Decision Skill.
EA	Equipment Agent.
EPS	Evolvable Production Systems.
FMS	Flexible Manufacturing Systems.
HCBA	Holonic Component-Based Approach.
HMS	Holonic Manufacturing Systems.
HUA	Handover Unit Agent.

IADE IDEAS Agent Development Environment.

MA Mechatronic Agent.

MAS Multi Agent System.

MB Message Broker.

MRA Machine Resource Agents.

OA Order Agent.

OH Order Holon.

PA Product Agent.

PH Product Holon.

PPA Product Plan Agent.

PSiA Product Sink Agent.

PSoA Product Source Agent.

RA Resource Agent.

RH Resource Holon.

RMS Reconfigurable Manufacturing Systems.

TEA Transport Entity Agent.

TSA Transport System Agent.

WA Workstation Agent.

WIP Work in Process.

YPA Yellow Pages Agent.

Capítulo 1

Introdução

Nos dias de hoje, os consumidores procuram uma nova gama de produtos mais variados e personalizados. Esta nova procura provoca uma diminuição do ciclo de vida de muitas gamas de produtos, sendo necessário encontrar novos paradigmas de produção capazes de satisfazer as necessidades dos consumidores e manter os níveis de competitividade do mercado global.

Com o passar do tempo e com as exigências dos consumidores a alterarem-se e com as mudanças introduzidas pelos mercados o número de modelos e a quantidade de produtos a produzir de cada modelo alterou-se drasticamente. Estas alterações devem-se à crescente procura de modelos altamente customizados e personalizados, como também ao elevado número de diferentes produtos à disposição do cliente.

Sendo assim, a constante alteração de produtos a serem produzidos como as constantes variações das quantidades a serem produzidas, levam a que os sistemas de produção tenham novas exigências, como dinamismo e flexibilidade.

Os sistemas baseados nas abordagens tradicionais tornaram-se demasiado rígidos para fazer frente a esta nova realidade, que requer sistemas que possam ser constantemente reconfigurados de modo sustentável, e flexíveis para produzir diversos produtos ao mesmo tempo.

Esforços têm sido realizados ao longo do tempo de modo a fazer frente a estas novas necessidades. Várias abordagens são propostas, desde o reajuste dos sistemas mais conservadores até a soluções mais evolutivas. Estas soluções mais recentes surgiram com a aplicação da informática à indústria de novas maneiras. O IDEAS Agent Development

Environment (IADE), é uma arquitetura desenvolvida no projeto **FP7 NMP IDEAS**. Este projeto foi criado neste âmbito, com o objetivo de desenvolver um sistema mecatrónico reconfigurável e robusto baseado em *Agentes*. O IADE gere a colaboração da sua rede de agentes utilizando o conceito de *Skill*. Cada agente da rede é responsável por executar uma determinada tarefa e divulga essa informação aos outros agentes na rede na forma de *Skill*.

1.1 Objetivos

O trabalho proposto tem o objetivo de analisar e otimizar o sistema mecatrónico *IADE* [28], desenvolvido no âmbito de um projeto *IDEAS*.

A analise do sistema vai focar principalmente a implementação do conceito de *Skill* na arquitetura, que é a ontologia usada pelos agentes do sistema, com o objetivo de:

- Melhorar o desempenho da execução de *Skills*.
- Reestruturar a *Skill* de modo a introduzir novos casos de uso.

1.2 Descrição do Documento

Esta dissertação é composta por seis capítulos: Introdução, Estado da arte, Arquitetura, Implementação, Testes e Validação e Conclusão e Trabalho Futuro.

O primeiro e presente capítulo, Introdução, dá uma breve introdução à evolução dos mercados no contexto produção e às contribuições desta tese.

O Estado da arte explora alguns dos novos paradigmas propostos no âmbito dos sistemas de manufatura e sistemas multiagente, e algumas arquiteturas que tentam implementar os teus conceitos.

No capítulo da Arquitetura, a arquitetura de referência IDEAS é apresentada, incluindo os seus conceitos básicos e as interações. O conceito de *Skill* e respetivo processo de execução, utilizados posteriormente, são estudados e detalhados de um ponto de vista concetual, tanto do ponto de vista do estado da arte como das novas contribuições.

O quarto capítulo, Implementação, apresenta a implementação da arquitetura detalhada no capítulo anterior. As entidades que compõem o sistema, as interações entre elas e o processo de execução de *Skills* são também descritos com um elevado grau de detalhe.

O capítulo Testes e Validação, apresenta um conjunto de casos de teste e os seus resultados obtidos através de simulação.

Por fim, o capítulo Conclusão e Trabalho Futuro é uma discussão das conclusões obtidas na realização deste trabalho e apontar os próximos passos.

Capítulo 2

Estado da Arte

Originalmente os sistemas de manufatura foram desenvolvidas tendo como objetivo uma produção em massa de produtos padronizados. Esta padronização visava a exploração da mão-de-obra altamente especializada e os conhecimentos associados a essa aprendizagem. No entanto a diminuição dos ciclos de vida dos produtos e o aumento da diversidade e exigência do mercado exigem uma nova abordagem no contexto da produção [7, 33].

Têm emergido novos paradigmas que introduzem no contexto da manufatura conceitos como robustez, reconfigurabilidade e flexibilidade. Muitos destes paradigmas, baseiam-se em sistemas de controlo distribuídos, dotados das características antes referidas, ao contrário dos sistemas tradicionais que se baseiam em sistemas de controlo centralizados. Tecnologia baseada em agentes têm sido utilizada para implementar este tipo de paradigmas, pois sistemas de controlo baseados nesta tecnologia, quando devidamente projetados e implementados, resultam num desempenho flexível, robusto, adaptativo e tolerante a falhas, o que são características chave em sistemas de manufatura. Ao longo deste capitulo serão apresentados vários paradigmas como os Sistemas Reconfiguráveis de Manufatura, Sistemas Holónicos de Manufatura, Sistemas Biónicos de Manufatura e Sistemas Evolutivos de Produção que têm o objetivo de trazer estes conceitos no contexto da manufatura.

2.1 Sistemas Reconfiguráveis de Manufatura

Os Sistemas Reconfiguráveis de Manufatura [17, 23] ou *Reconfigurable Manufacturing Systems (RMS)* na literatura, apareceram como alternativa aos Sistemas Flexíveis de Manufatura ou *Flexible Manufacturing Systems (FMS)* na literatura. Os principais problemas dos FMS são o seu elevado custo e baixo volume de produção, o que levou a fraca adoção

deste tipo de sistemas, por outro lado os RMS são uma solução mais económica que combina o elevado volume de produção com a flexibilidade e escalabilidade oferecida pelos FMS.

Os RMS são sistemas preparados para sofrer rápidas alterações na sua estrutura tanto a nível dos seus componentes de hardware como de software, para ajustar a sua capacidade de produção e funcionalidade em resposta a variações inesperadas no mercado.

As características de um RMS são modularidade, escalabilidade, integralidade, convertibilidade e diagnosticabilidade. Modularidade porque tanto a componente de hardware como de software são modularizadas para permitir a reorganização do sistema. Escalabilidade significa que o sistema é escalável em termos de volume de produção. Integralidade implica que o sistema e os seus componentes são desenvolvidos de modo a permitir uma rápida integração e futura introdução de novas tecnologias. Convertibilidade permite trocas rápidas entre produtos existentes e rápida adaptação a novos produtos. Diagnosticabilidade porque permite identificar rapidamente a fonte de problemas de qualidade que ocorram em sistemas de grandes dimensões [6]. Esta abordagem permite começar a produzir um produto ou uma nova mistura de produtos num curto espaço de tempo, nas quantidades pretendidas através de uma configuração rápida do sistema. Sendo assim num RMS os produtos são agrupados em famílias, cada uma das quais necessita de configuração diferente [15].

Alguns estudos e trabalhos têm sido desenvolvidos de modo a tornar a reconfiguração destes sistemas mais rápida e mais abrangente, através da utilização de Redes de Petri [21]. Neste caso apresenta-se uma solução em que são reconfigurados os modelos descritos pelas Redes de Petri de uma forma automática, sendo esta reformulação feita num nível mais baixo do habitual.

2.2 Sistemas Multi Agente

Um agente, ou *agent* na literatura, é um software autónomo que capaz de tomar decisões inteligentes, comunicar e cooperar com outros agentes [44], que normalmente está num ambiente composto por vários agentes que juntos formam um Sistema Multi Agente, referenciado na literatura como *Multi Agent System (MAS)*. As principais características de um agente são a sua autonomia, pois controlam tanto o seu estado interno como o seu comportamento no ambiente em que estão inseridos, a sua adaptabilidade e capacidade de interagir com o ambiente, o que permite ao agente adaptar-se a variações do ambiente e alterar o seu comportamento sem a intervenção do seu designer, e comunicar com os outros agentes permitindo atingir os objetivos do sistema através de colaboração [24].

Os MAS são compostos por agentes que colaboram dinamicamente entre eles de forma a satisfazer os objetivos do sistema, como cada agente tem apenas uma vista parcial do

sistema, quando um agente não tem conhecimentos ou capacidade para atingir o seu objetivo individualmente, ele procura outros agentes no sistema que possuam os conhecimentos necessários [19]. Os agentes organizam-se em estruturas hierárquicas nas quais cada agente tem um objetivo. Este objetivo especifica a função, direitos, obrigações e as regras das suas interação com os outros agentes e o ambiente. Existem dois tipos básicos de objetivos: o operador, que completa tarefas que normalmente estão associadas a um recurso físico. Por outro lado o gestor atribui tarefas, monitoriza a execução, recolhe e combina os resultados obtidos. Todas as interações entre agentes são definidas por uma sequência de mensagens que fazem parte do protocolo utilizado. O sistema não tem um controlo centralizado, apenas as regras impostas a cada agente mantêm a harmonia do mesmo [24].

A tecnologia de agentes tem sido reconhecida como um paradigma promissor para a próxima geração de sistemas de manufatura [24, 38]. Investigadores têm tentado aplicar esta tecnologia nas áreas de colaboração empresarial [9, 10], planeamento de processos [37] , controlo de linhas de produção [2, 27], entre outras.

Ao nível dos sistemas de controlo de linhas de produção existe a arquitetura RIDER (*Real-Time Decision Making in Manufacturing*) [27], onde após alguma falha num elemento da linha de produção, os agentes procuram alternativas para evitar parar a produção através de trocas de mensagens entre si. Os agentes que monitorizam o sistema podem também encontrar problemas ao nível da otimização do planeamento e propor alternativas aos agentes que gerem e esta componente do sistema.

Outro exemplo de um MAS é a arquitetura CoBaSA [2] que é uma arquitetura que se baseia na criação de coligações entre os componentes do sistema. Cada componente é modular, pode ser facilmente reutilizado e é representado por um agente que é configurado com o mínimo esforço de programação. Uma coligação é um grupo de agentes que se juntam para atingir um objetivo comum, sendo assim a coligação consegue executar uma operação complexa através da execução de operações mais simples que são executadas pelos seus membros [4]. Os componentes básicos da arquitetura proposta são: *Machine Resource Agents (MRA)*, *Coordenating Agents (CA)*, *Clusters*, *Coalitions*, *Broker* e *Contracts*. OS MRA são os agentes de mais baixo nível, como tal, são a ponte entre o nível de controlo lógico e o nível de controlo físico, cada MRA representa um equipamento físico que consegue desempenhar determinadas funções (*Skills*) na linha de montagem. Uma *Coalition* é uma estrutura organizacional que junta um grupo de agentes com o intuito de satisfazer um objetivo comum. Este grupo de agentes é normalmente composto por MRA's, outras *Coalitions* e um CA que tem o objetivo de liderar, coordenar e representar a *Coalition*. Um *Cluster* é um diretório onde os agentes interessados em participar em *Coalitions* se podem registar. Este diretório é gerido pelo *Cluster Manager Agent* que é responsável por mander

o diretório e gerir as operações d registo e saída de agentes. O *Broker* é um agente responsável por criar *Coalitions* com a interação de um utilizador externo. O *Broker* recolhe informação dos *Clusters* e baseado no dados introduzidos pelo utilizador supervisiona o processo de criação da *Coalition*. Os *Contracts* são a ontologia proposta para comunicação entre os agentes da arquitetura. A *Skill* é um conceito importante presente na ontologia desta arquitetura. Por definição *Skill* é a habilidade de desempenhar ações necessárias ao processo de manufatura. As *Skills* podem ter vários níveis de abstração, por exemplo, uma *Skill* complexa é gerada através da agregação de outras *Skills*, básicas ou complexas. As *Skills* complexas presentes na ontologia identificam quais são as *Skills* necessárias à sua execução [14].

2.3 Sistemas Holónicos de Manufatura

De forma a fazer frente às novas necessidades de produção, a customização em massa ou o baixo volume de encomendas, surgiram os Sistemas Holónicos de Manufatura ou *Holonic Manufacturing Systems (HMS)*. A ideia principal neste tipo de sistemas é proporcionar um processo de fabrico altamente dinâmico e descentralizado.

O conceito de holon foi introduzido por Arthur Koestler com o objetivo de explicar a evolução de sistemas biológicos e sociais [16]. Um holon é uma entidade autónoma capaz de actuar em circunstancias imprevisíveis e capaz de cooperar com outros holons tornando-se num componente de um cenário mais complexo. Um holon pode representar um componente físico ou lógico num sistema, contem informação sobre si próprio e sobre o ambiente onde opera, contendo uma parte de processamento de informação e uma parte de processamento físico quando representa um componente físico como um robot industrial [19].

Um HMS [1, 8] é um sistema composto por holons, onde estes comunicam entre si e cooperam de como a atingir o objetivo do sistema. Um grupo de holons pode estabelecer uma hierarquia entre si, compondo uma pequena comunidade dentro do sistema com o intuito de atingir um objetivo comum, para o qual cada holon desempenha uma tarefa mais simples. As hierarquias criadas podem ser permanentes ou temporárias e um holon pode pertencer a várias hierarquias [24].

Tanto o paradigma de manufatura holónico como o de agentes, foram desenvolvidos tendo em conta s mesmos princípios fundamentais de autonomia e cooperação, explorando a distribuição e descentralização de entidades e funções. Do ponto de vista conceptual, um holon é um conceito e um agente é mutuamente um conceito e uma tecnologia, sendo possível implementar o conceito de holon e HMS utilizando tecnologia de agentes [19].

Existem na literatura várias arquiteturas baseadas em holons, uma das mais conhecidas é a arquitetura PROSA [42]. Nesta arquitetura existem três tipos básicos de holons: *Product Holon (PH)*, *Order Holon (OH)*, e *Resource Holon (RH)*. O PH contem o modelo de um produto, isto é, contem toda a informação acerca do processo de criação do produto, materiais necessários, etc. O RH é uma abstração de um recurso físico, ou seja, máquinas, transportadoras, ferramentas ou armazém de material. O objetivo desta camada de abstração é permitir ao recurso alocar a sua capacidade a outros holons. O OH representa a tarefa de construir o produto segundo as suas especificações, pode ser considerado uma instância de um produto. O processo de construção de um produto passa guardar informação acerca do estado do produto e negociar com holons recurso com o objetivo de encaminhar o produto tendo em conta as limitações logísticas do sistema no momento em questão [44].

ADACOR [18, 20] é outra arquitetura holónica onde o elemento básico é um holon autónomo e cooperativo que representa tanto os recursos físicos como as entidades lógicas do sistema. Esta arquitetura define um modelo semelhante à arquitetura PROSA, acrescentando apenas um *Supervisor Holon* the introduz coordenação, formação de grupos e otimização ao controlo descentralizado [24]. Exitem quatro tipos básicos de holons: *Product Holon*, *Task Holon*, *Operational Holon* e *Supervisor Holon*. Os primeiros três holons são idênticos aos PH, OH e RH, respetivamente, encontrados na arquitetura PROSA. O *Supervisor Holon* é um coordenador que processa planos de escalonamento otimizados para os *Task Holon* e *Operational Holon*. O *Operational Holon* consegue também enviar aos *Task Holon* um aviso caso um problema seja detetado, para que este consiga encontrar uma rota alternativa [44].

Holonic Component-Based Approach (HCBA) [12, 13] propõe uma abordagem diferente das arquiteturas vista anteriormente. O HCBA define apenas dois holons, o *Product Holon* e o *Resource Holon*. O *Resource Holon* é idêntico ao encontrado na arquitetura PROSA, mas ao contrário das arquiteturas anteriores, onde o *Product Holon* tinha apenas a função de guardar o modelo de um produto, nesta arquitetura este é composto também por um *Coordinator Component (CC)* que tem o objetivo de esperar de ordens de produção. Após receber uma ordem de produção o CC actualiza o seu plano de produção e cria *Work in Process (WIP)* agents de tempo a tempo com o objetivo de completarem os pedidos descritos na ordem de produção. Um WIP agent é responsável por transportar os materiais necessários ao produto ao longo da linha de produção segundo os objetivos que lhe foram dados pelo CC. O WIP agent é também responsável por negociar com os *Resource Holons*. Esta arquitetura define também dois mecanismos de comunicação, o *Black Board System (BBS)* e o *Message Broker (MB)*. O BBS é usado para comunicação interna dos holons, ou seja, para tornar possível a troca de informação entre a componente de software do holon e a componente física. Por sua vez, o MB gere a troca de mensagens entre holons, para ser

possível ao MB gerir toda a comunicação entre os holons, estes são registados no mesmo quando são introduzidos no sistema.

A *Rockwell Automation* [22, 43] propõe uma arquitetura holólonica implementada com agentes, onde agentes/holons básicos são: o *Product Agent (PA)*, o *Product Plan Agent (PPA)*, o *Order Agent (OA)*, o *Equipment Agent (EA)* e o *Workstation Agent (WA)*. Comparando com os holons definidos na arquitetura PROSA, o PPA representa o PH, o PA representa o OH, pois do ponto de vista desta arquitetura o PA mantém informação sobre o processo de produção contrariamente ao que acontece com o PH da arquitetura PROSA que é apenas um modelo para a criação de um tipo de produto, e os agente equipamento e WA representam, respetivamente, a componente física e lógica do RH. O WA negoceia a alocações de recursos dos EA's a ele associados, com os agentes que controlam o processo de produção, e supervisiona e controla o processo de execução. Normalmente uma operação fornecida pelo WA é a combinação de operações mais simples executadas pelos equipamentos a ele associados. O OA é o elo de ligação entre os novos pedidos realizados e resto do sistema. Quando um pedido de produção entra no sistema, é criado um OA, que por sua vez, cria uma nova instância do PA para cada produto pedido. Para obter os instruções de criação do produto o PA contacta o PPA, que é responsável por facultar aos PA os planos para os diferentes tipos de produtos implementados. Esta arquitetura utiliza também agentes do tipo *Middle-Agent*, que funcionam como uma rede de agentes que têm o objetivo de difundir as tarefas disponibilizadas pelos vários WA. Este tipo de implementação permite, ao contrário do que acontece nas abordagens tradicionais, aumentar a tolerância a falhas, pois tradicionalmente existe apenas um agente encarregue de gerir a lista de tarefas dos WA [44].

Na ontologia proposta nesta arquitetura uma *Order* pode ser composta por várias *ProductOrders* que são especificados por um *ProductSpecification*. O *ProductSpecification* contem os parâmetros específicos do produto como cor, forma, material utilizado, etc... Cada *Product* tem o seu próprio *ProductionPlan* e que é composto por uma sequência de *ProductionSteps* que poder ser ordenados utilizando a relação *precedes*. Existe também a possibilidade de especificar determinados passos necessários através da relação *requires*, o que é bastante importante na customização dos planos. Cada *ProductionStep* tem uma *Operation* que representa a tarefa a realizar neste passo, assim como parâmetros que especificam detalhes da *Operation*. Neste contexto a *Operation* é uma tarefa que é desempenhada por um WA.

2.4 Sistemas Biónicos de Manufatura

Os Sistemas Biónicos de Manufatura referenciados na literatura como *Bionic Manufacturing Systems (BMS)* [35, 40] são sistemas, autónomos e espontâneos capazes de comunicar e cooperar dentro de uma hierarquia. Este tipo de sistemas têm como entidade básica o modelon que pode ser visto como uma célula de um organismo biológico. Todas as partes de um BMS são compostas por modelon que apenas diferem entre si nas funções que conseguem desempenhar. Num BMS cada elemento da linha de montagem é um modelon e tal como uma célula num organismo biológico, este recebe os parâmetros necessários por parte do sistema e executa uma operação. Estes modelons podem ser utilizados como blocos básicos na construção de sistemas de controlo hierárquicos, tal como, linhas de montagem. Nestas hierarquias quando é dada uma ordem ao nível mais alto a informação é passada nível a nível até ao nível mais baixo onde as operações básicas são executadas. Os modelons têm dois tipos de informação: Informação genética que é herdada (DNA-type), e informação aprendida individualmente (BN-type). Ao nível do processo de criação, os produtos são desenvolvidos a partir de materiais que têm a sua informação DNS-type. Os equipamentos físicos da linha de produção constroem o produto utilizando a sua informação BN-type [41]. A entidade responsável por garantir que a passagem informação baseada no ADN de cada modelon é corretamente passada entre camadas é a entidade supervisora (enzimas, que na biologia funcionam como catalisadores de reações), permitindo assim a formação dos modelons adequados para cumprir tarefas mais complexas.

Apesar das similaridades entre os modelons nos BMS e os holons nos HMS, os sistemas compostos por estas unidades são bastante diferentes ao nível da criação das hierarquias, pois nos HMS os holons são mutuamente uma parte e um todo no sistema enquanto que nos BMS os modelons estão organizados em camadas de complexidade [39].

2.5 Sistemas Evolutivos de Produção

Os Sistemas Evolutivos de Produção [25], referenciados na literatura como *Evolvable Production Systems (EPS)*, são um paradigma derivado de uma abordagem dinâmica e mutável que tem em atenção a alterações feitas ao sistema e em como elas podem ser geridas eficientemente. Neste conceito, adaptabilidade representa a capacidade de um sistema propor uma configuração alternativa para superar uma perturbação existente. Evolução, por outro lado, é mediada por uma decisão estratégica de alto nível que irá resultar no desenvolvimento de um ou mais módulos ou reestruturação de produtos com o intuito de explorar alternativas na reconfiguração do sistema [11]. Os EPS segue três princípios fundamentais [3, 26, 30, 36]:

- 1: O design do produto mais inovador só pode ser alcançado se não existirem restrições no processo de montagem. Apenas nestas circunstancias um design otimizado.
- 2: Sistemas em condições dinâmicas têm de ser evolutivos. Ou seja, precisam de ter a capacidade de evoluir para abordar o novo conjunto de requisitos.
- 3: Os sistemas EPS são baseados em módulos inteligentes, orientados a processos, autónomos, auto-organizados que se podem juntar com o intuito de fornecer diferentes funcionalidades quando necessário.

Para atingir estes princípios um sistema EPS tem de ser desenvolvido tendo em consideração as seguintes características [5, 29]:

- Modularidade: Deve ser composto por módulos independentes.
- Granularidade: Um sistema EPS deve permitir vários níveis de granularidade, ou seja, um módulo pode ser uma simples garra ou um robot inteiro.
- Conectividade: Deve ser capaz de lidar com a introdução de novos módulos enquanto o sistema está a funcionar. O sistema deve reestruturar-se internamente para se manter eficiente.
- Reconfigurabilidade: O sistema deve ser a capacidade de lidar com a reestruturação do seu layout sem comprometer qualquer funcionalidade.

Uma das arquitetura EPS num contexto multi agente é a arquitetura *IADE* [32]. Esta arquitetura tem como referência a arquitetura CoBaSa descrita anteriormente, como tal muitos dos agentes e conceitos encontrados aqui são comuns às duas arquiteturas. Os agentes que constituem esta arquitetura podem ser colocados em três grupos distintos: execução, suporte e transporte. O grupo de execução é composto pelo *Resource Agent (RA)*, que é a entidade que interage direobjetotamente com os controladores da linha e tem um lista de tarefas que conseguem executar. O *Coalition Leader Agent (CLA)* que é utilizado para compor tarefas complexas utilizando as tarefas disponibilizadas por RA's e outros CLA's, este agente não tem qualquer contacto com os controladores mas conseguem coordenar a execução de outros CLA's e RA's. O *Product Agent PA* é o nível de abstração mais alto do sistema, cada PA representa um produto e tem a capacidade de controlar o seu próprio plano de execução e de tomar decisões sobre onde e como essas execuções se processam. O grupo de suporte é constituído pelo *Deployment Agent (DA)* que permite ao utilizador lançar agentes de dos grupos de execução e transporte num determinado controlador e pelo *Yellow Pages Agent (YPA)* que regista e disponibiliza no sistema as tarefas que cada agente consegue executar. O grupo de Transporte tem agentes responsáveis por introduzir, remover e deslocar os produtos no sistema [31].

Na ontologia proposta nesta arquitetura as tarefas que os Agentes conseguem executar são denominadas por *Skills* [14]. Uma *Skill* desempenha para um agente o mesmo papel que um método para um objeto. Sendo assim uma *Skill* inclui uma interface que contém toda a informação necessária à sua execução. Uma *Skill* representa todas as funcionalidades que um agente partilha no MAS, como tal, pode representar uma tarefa disponibilizada por um RA, ou seja, uma tarefa que representa uma ação do controlador gerido pelo RA em questão, ou pode representar uma tarefa complexa fornecida por um CLA (*Composite Skill*), este tipo de tarefa não é nada mais que uma lista de *Skills* mais simples (*Atomic Skills* e/ou *Composite Skill*) que pode ser executada sequencialmente, paralelamente, condicionalmente dependendo de uma expressão booleana ou em qualquer combinação dos métodos anteriores.

Esta arquitetura suporta o presente trabalho e será discutida com mais detalhe no capítulo da arquitetura.

Capítulo 3

Arquitetura

O IADE é uma biblioteca desenvolvida no âmbito do projeto **FP7 NMP IDEAS** e baseada na plataforma JADE.

O resultado desta tese é suportado pelo IADE e simultaneamente contribuí para a sua implementação. As contribuições principais consistem na otimização do conceito de *Skill* e interações entre os principais agentes.

Este capitulo começa por descrever os agentes que compõe a arquitetura e as suas principais interações, relevantes ao funcionamento das *Skills*, que são o foco principal deste trabalho. O conceito de *Skill* é explorado e finalmente a nova abordagem proposta e otimizações são apresentadas.

3.1 IADE: Agentes da Arquitetura

A arquitetura do IADE, é composta por um total de nove agentes (Figura 3.1) [34]. Estes agentes comunicam e organizam-se entre si tornando possível a gestão e execução de planos de produção introduzidos no sistema.

Os agentes que compõem esta arquitetura são:

- Deployment Agent (DA): Medeia a plataforma de agentes e os controladores da linha. Permite ao utilizador lançar agentes responsáveis pela execução ou transporte num determinado controlador.

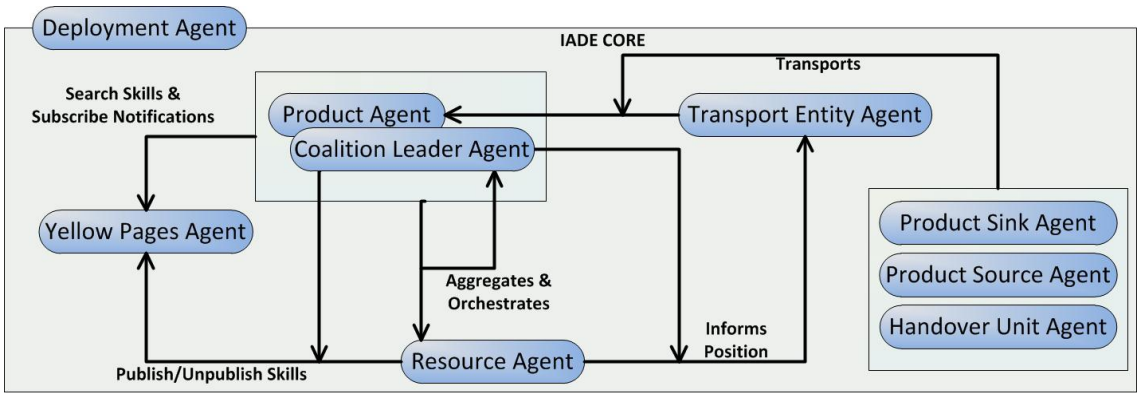

Figura 3.1: Agentes da arquitetura IADE [34]

- Yellow Pages Agent (YPA): Mantém o controlo das *Skills* que cada um dos agentes disponibiliza no Sistema. Os agentes podem consultar o YPA para localizar outros agentes e as suas habilidades. Cada agente pode subscrever um serviço de notificação que o informa, se um agente específico deixou a plataforma.

- Resource Agent (RA) ou Machine Resource Agent (MRA): Interagem diretamente com os controladores da linhae representam a entidade de menor abstração possível da pilha do IADE. O seu principal papel é o de traduzir habilidades para código nativo e assegurar a sua execução e sincronização com a plataforma de agentes. Os RA's informam o sistema de transporte sobre a localização física onde as suas habilidades podem ser executadas.

- Coalition Leader Agent (CLA): Os CLA's são utilizados para compor funcionalidades recorrendo a agentes já existentes. Não interferem com qualquer controlador da linha, mas são capazes de coordenar a execução de outro RA ou CLA. Estes implementam subprocessos que serão posteriormente consumidos por agentes do tipo produto, durante a execução do seu plano de processo. À semelhança do RA, o CLA informa o sistema de transporte sobre a localização física onde as habilidades estão disponíveis.

- Product Agent (PA): O PA representa o nível mais alto da abstração do sistema e que implica um mapeamento direto entre o agente e um item específico a ser produzido. Cada PA tem a capacidade de controlar o seu próprio plano de execução e de tomar decisões sobre o local onde cada passo dessa execução é executado. A ação coletiva do PA e os agentes do sistema de transporte permite a este basear as suas decisões;

- Product Source Agent (PSoA): O PSoA gere a entrada de matéria prima no sistema e posteriormente, quando esta já se encontra no sistema, associa-lhe um PA específico. Esta associação é fundamental, já que pode haver mais que um PA disponível para entrar, e neste contexto, o PSoA gere a entrada de cada PA em espera.

- Product Sink Agent (PSiA): O PSiA tem a função oposta ao PSoA. Neste contexto, o PSiA liberta o produto do PA, sempre que um PA especifica termina o seu plano de execução.

- Transport Entity Agent (TEA): Abstrai um transportador específico do sistema. É responsável pelo deslocamento de paletes entre encaminhadores. Cada TEA disponibiliza vários pontos de acoplamento, onde as estações, constituídas por RA's e CLA's, podem ser acopladas e desacopladas durante a execução. O TEA é sensível a essas ações e gere o tráfego das paletes no mesmo, informando o PA sobre o custo do encaminhamento e a posição atual aquando da chegada ao destino.

- Handover Unit Agent (HUA): Cada HUA controla um encaminhador do sistema e calculam a árvore de custos mínimos para chegar a cada um dos destinos possíveis na linha. Esta informação é posteriormente partilhada como TEA associado, para quando este necessita negociar com o PA.

Os agentes da arquitetura estão divididos de acordo com o seu papel no sistema. Os agentes DA e YPA pertencem ao grupo de agentes de **suporte**, enquanto que os agentes MRA, CLA e PA desempenham tarefas ao nível da **execução**, e por fim os agentes PSoA, PSiA, TEA e HUA estão encarregues do **transporte**. Os agentes relevantes para o trabalho em estudo são os agentes diretamente relacionados com a execução.

A Figura 3.2 [34] representa as interações entre os agentes quando o sistema é iniciado. O sistema é iniciado com a iniciação dos HUA's. Após todos os HUA's estarem iniciados, são introduzidos no sistema os TEA's. Existe um HUA entre dois ou mais TEA's e estes juntos formam a rede de transporte. Após a formação da rede de transporte são introduzidos os PSoA's e PSiA's, estando assim o sistema de transporte totalmente inicializado.

O próximo passo é a introdução dos MRA's no sistema e a indexação das suas *Skills*. A configuração dos CLA's é o passo seguinte, pois estes permitem criar processos mais complexos através da composição de novas *Skills* tirando partido daquelas que já se encontram no sistema. Após a sua configuração estes agentes são iniciados no sistema e indexam as suas próprias *Skills* no YPA que está associado ao TEA da zona onde o CLA em questão executa as suas *Skills*.

Por fim após definidos os processos adequados e respetivas localizações de execução conhecidas pelo sistema, os PA's são lançados e é iniciada a produção. A sua entrada no sistema é sempre assistida por um PSoA e a sua saida por um PSiA.

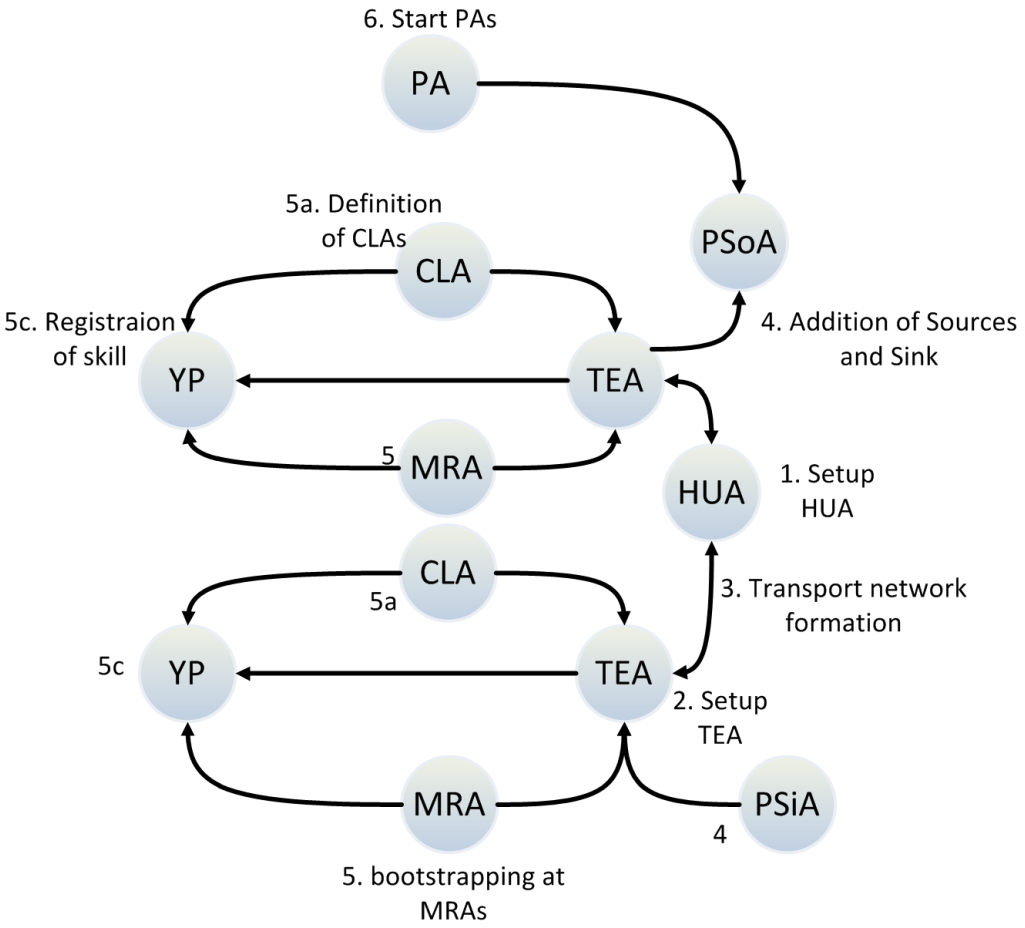

Figura 3.2: Arquitetura IADE - Interações na configuração [34]

A Figura 3.3 [34] mostra que de um ponto de vista funcional, tanto as principais interações como o processo de decisão são centrados no PA.

Isto verifica-se porque o PA é a entidade do sistema responsável pela tomada de decisões de alto nível e pela execução do plano de produção. Aquilo que do ponto de vista do PPAA podem parecem interações simples, acabam por representar um número considerável de ações entre os vários agentes.

Em particular, antes de decidir onde executar a próxima *Skill*, o PA contacta o TEA, que está associado à sua zona, que por sua vez vai recolher informação sobre custo de transporte, custo de execução e localização dos agentes capazes de executar a *Skill* em questão. O local de execução é depois escolhido com base no custo combinado dos valores anteriormente obtidos. Por fim é usada a mesma lógica para encontrar um transporte para o PA.

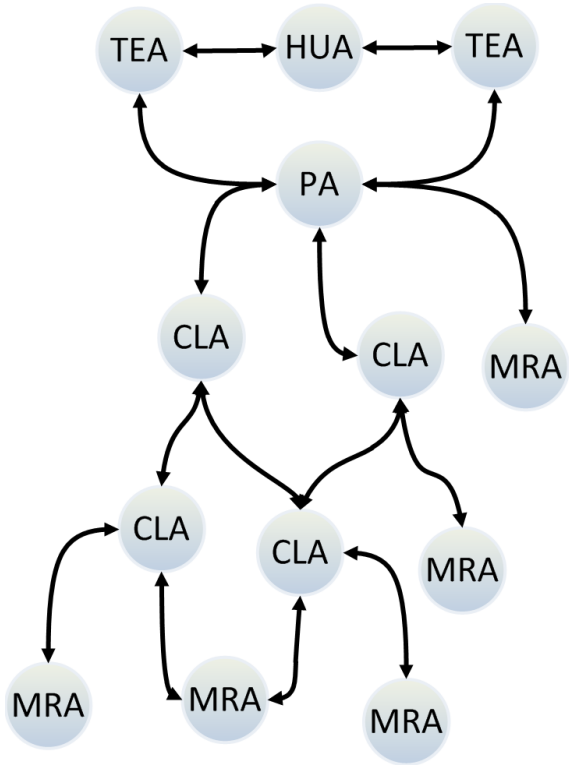

Figura 3.3: Arquitetura IADE - Interações na execução [34]

No local de execução o PA pede a execução ao agente correspondente. Se o agente for um CLA, ele vai pedir execuções a outros agentes até completar a sua tarefa.

3.2 Arquitetura Simplificada

Com o intuito de estudar e compreender o conceito de *Skill*, e para o isolar da influência do sistema de transporte, cuja dinâmica está fora do âmbito do presente trabalho e têm sido estudada em [33], uma versão simplificada da arquitetura IADE foi considerada Figura 3.4 [34].

Como se pode verificar na figura, o número de agentes que compõem o sistema de transporte foi reduzido. O sistema de transporte é totalmente controlado por uma instância do *Transport System Agent (TSA)*.

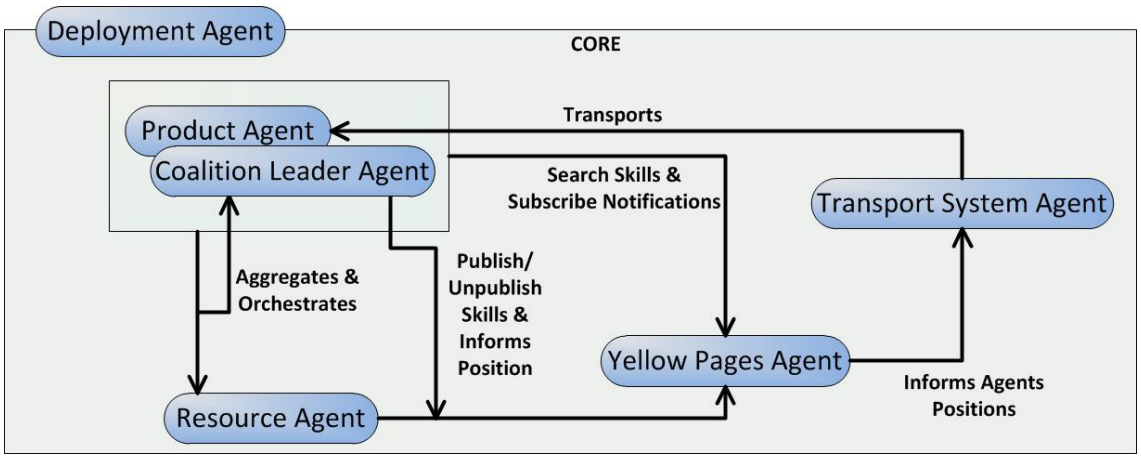

Figura 3.4: Agentes da arquitetura IADE simplificada [34]

Como se pode verificar na Figura 3.5 [34], o sistema é iniciado de maneira idêntica. Primeiro é iniciado o TSA, depois os YPA's e finalmente os MRA's e CLA's são adicionados ao sistema e registados no YPA correspondente.

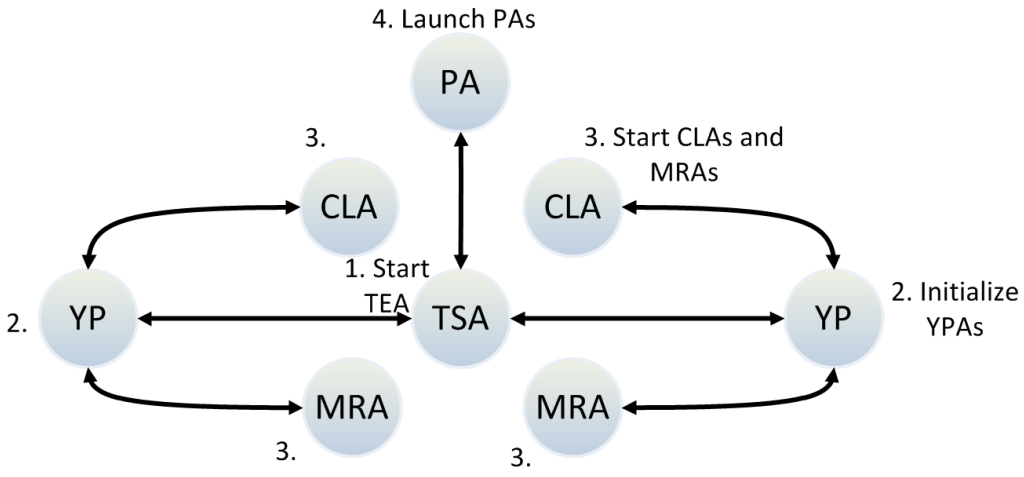

Figura 3.5: Arquitetura IADE simplificada - Interações na configuração [34]

Após o sistema estar completamente iniciado, o PA pode ser adicionado. Como se pode verificar, as interações entre PA's, CLA's e MRA's são as mesmas encontradas anteriormente.

As interações durante a execução sofreram algumas alterações. Como está representado na Figura 3.6 [34], o PA pede ao a uma única instância do TSA pela lista de localizações onde a *Skill* desejada pode ser executada. O TSA responde com a localização de todos os YPA's, em vez dos custos como acontecia anteriormente.

De seguida o PA contacta os YPA's para pedir a execução nas diferentes localizações.

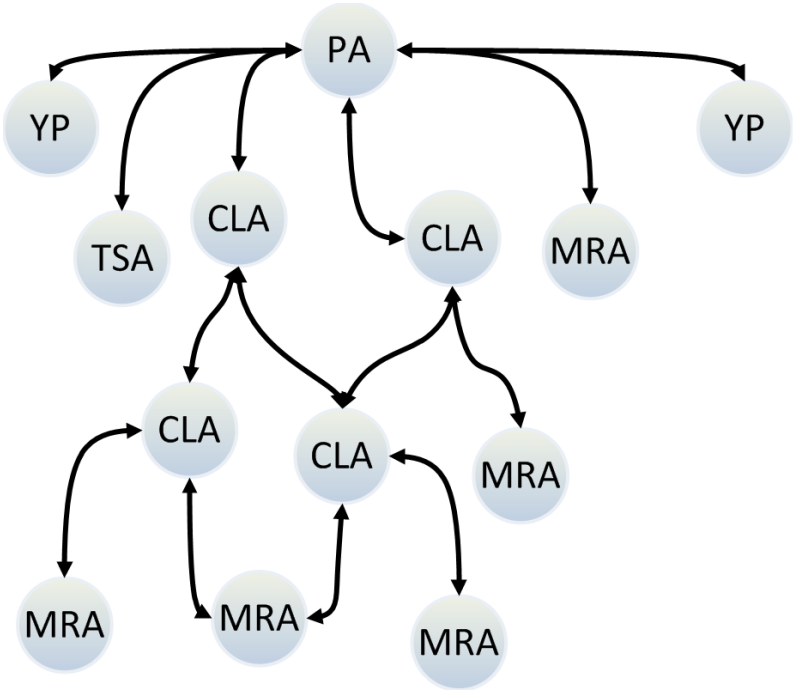

Figura 3.6: Arquitetura IADE simplificada - Interações na execução [34]

A consequência desta abordagem é o facto do custo de transporte não ser considerado no processo de decisão.

3.3 Definição de Skill

Os agentes presentes no ambiente em estudo expressam as suas funcionalidades na forma de *Skills*. As *Skills* desempenham para o agente o mesmo papel que um método para um objeto. Sendo assim, uma habilidade inclui uma interface que contém toda a informação necessária à sua execução, requerida por um produto. Nessa informação encontram-se:

- Nome e ID;
- Parâmetros e os seus tipos;
- Referência ao agente que a irá executar;
- Variáveis de estado que permitem o controlo de execução;

As *Skills* estão divididas em três subtipos:

- *Atomic Skill (ASk)*: Implementam mecanismos genéricos que associam a informação da *Skill* à execução do mesmo num controlador específico (procedimentos de baixo nível, programas e outros sistemas de integração);

- *Composite Skill (CSk)*: Implementam processos utilizando outras *Skills*, que neste contexto são chamadas de *SubSkills*. Os processos são criados com recurso a um work-flow como descritor. Ao contrario das ASk's, as CSk's nunca interagem diretamente com o hardware, elas apenas mediam a execução ao nível dos agentes. Também é definido se as *SubSkills* são executadas em paralelo ou sequencialmente;

- *Decision Skill (DSk)*: Implementam a capacidade de tomar decisões durante a execução de processos, através da avaliação de uma expressão booleana. De acordo com o resultado da decisão, o processo pode tomar diferentes caminhos;

A Figura 3.7, apresenta uma CSk que está configurada para executar as suas *SubSkills* sequencialmente.

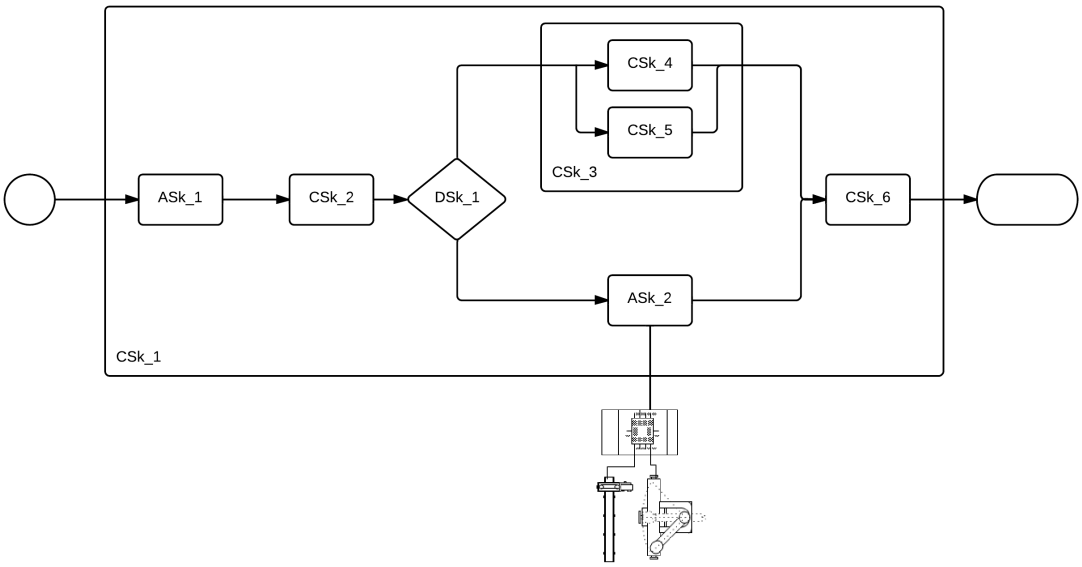

Figura 3.7: Exemplo de uma Composite Skill com Decision Skill

Quando é iniciada a execução da CSk_1, o agente responsável pela sua execução cria um *Sequencial Behaviour* que vai executar todas as *SubSkills* sequencialmente. As *Skills* ASk_1 e CSk_2 são executadas através da máquina de estados descrita anteriormente, a DSk_1 é uma DSk como tal, apenas um dos seus ramos vai ser executado. O ramo escolhido depende da avaliação de uma expressão definida pelo utilizador, caso a expressão seja avaliada como verdadeira, o ramo superior é executado. Caso a avaliação seja falsa então é executado o outro ramo. Para poder executar as *Skills* CSk_4 e CSk_5 em paralelo, foi criada a CSk_3 configurada para executar as suas *SubSkills* em paralelo. Ao contrario das restantes, as *Skills* DSk_1 e CSk_3 só existe no contexto da CSk_1, o que significa que estas são processadas pelo agente responsável pela CSk_1, sendo assim as suas *SubSkills*

são incluídas na lista de execuções do agente.

Um dos pontos comuns mais importantes do funcionamento das CSk's e das DSk's, é a associação entre os seus parâmetros e os parâmetros das suas *SubSkills*. Este processo é realizado durante a configuração da *Skill* pelo utilizador, que deve associar a cada parâmetro de cada *SubSkill* um parâmetro da *Skill*.

Como é possível verificar pelas *Skills* CSk_2 e CSk_4 da Figura 3.8, é possível criar novas CSk's e respetivas *SubSkills*.

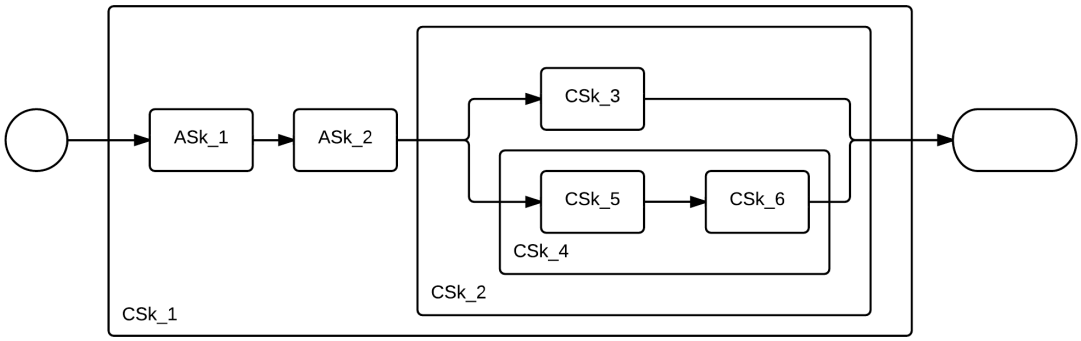

Figura 3.8: Exemplo de uma Composite Skill

É importante notar que ao contrário das outras *Skills* da Figura 3.8, que podem ser controladas por diferentes agentes, as CSk_2 e CSk_4 são controladas pelo menos agente da CSk_1. Isto implica também que na fase de associação de parâmetros, os parâmetros das *Skills* CSk_3, CSk_5 e CSk_6 sejam considerados. As *Skills* CSk_2 e CSk_4, não têm parâmetros porque existem apenas com o objectivo de definir o modo de execução das suas *SubSkills*. O facto das *Skills* CSk_2 e CSk_4 só existirem dentro da CSk_1 e não serem autocontidas, torna a criação de CSks mais complexas bastante difícil de configurar.

O objetivo desta abordagem ao nível da configuração das *Skills* e dos seus parâmetros, é ter uma utilização de memória baixa e manter um processo de configuração simplificado para o utilizador responsável pela configuração do sistema.

Alguns dos pontos negativos desta abordagem são:

- Com o aumento do número de *SubSkills* a associação de parâmetros fica cada vez mais difícil e complexa devido ao elevado número de associações que é necessário estabelecer;

- Como a associação de parâmetros é apenas entre os parâmetros da *Skill* principal e

as suas *SubSkills*, não é possível tirar partido da utilização de parâmetros de saída entre as *SubSkills*;

- Não é possível tirar partido das DSk's para criar ciclos na execução;

3.3.1 Skill: Nova Abordagem

Com o intuito de eliminar os pontos negativos da abordagem descrita na secção anterior, foi desenvolvida no contexto deste trabalho uma nova abordagem para o conceito de *Skill*.

Os pontos mais importantes desta abordagem são:

- A criação de CSk's e DSk's autocontidas que possibilitam a existência de vários níveis de abstração nos work-flows;
- A possibilidade de associar parâmetros de *SubSkills* entre si, permitindo utilizar outputs de uma *Skill* A como input de uma *Skill* B;

A criação de *Skills* autocontidas permite dividir o processo de configuração da *Skill* principal descrito na secção anterior, na configuração de várias *Skills* mais simples. Utilizando esta nova abordagem, a configuração da CSk_1 da Figura 3.8 necessita apenas de contemplar as *Skills* ASk_1, ASk_2 e CSk_2 tal como pode ser visto na Figura 3.9, pois a *Skill* CSk_2 é vista como uma caixa negra da qual são apenas conhecidos os parâmetros e a tarefa que desempenha.

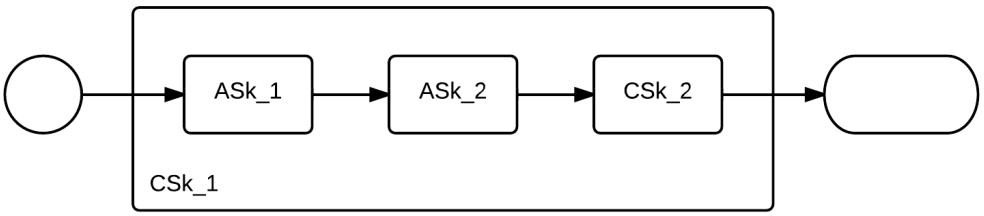

Figura 3.9: Skill principal da CompositeSkill da Figura 3.8

As *SubSkills* CSk_2 e CSk_4 são configuradas independentemente da *Skill* principal e, ao contrário da abordagem anterior, têm todos os parâmetros necessários à execução das suas *SubSkills* tal como uma *Skill* principal, como se pode verificar na Figura 3.10.

Nesta abordagem, para cada CSk ou DSk o utilizador associa os seus parâmetros aos parâmetros das suas *SubSkills*, tornando cada CSk ou DSk presente no work-flow num módulo autocontidas que representa um nível de abstração no design no mesmo. Sendo

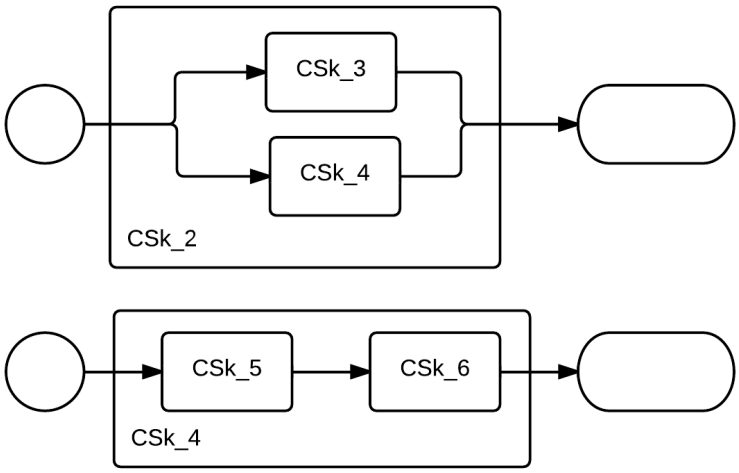

Figura 3.10: SubSkills da CompositeSkill da Figura 3.8

assim, cada um destes módulos representa um nível de abstração no work-flow, podendo ser guardados e reutilizados na construção de outras *Skills*.

Uma das principais vantagens desta abordagem, que advêm da possibilidade de associar parâmetros entre *SubSkills*, é a possibilidade de criar DSk's com ciclos. Isto só é possível porque os parâmetros de saída das *SubSkills* podem ser usados para atualizar a expressão avaliada pela DSk. A Figura 3.11 apresenta um exemplo da utilização deste recurso.

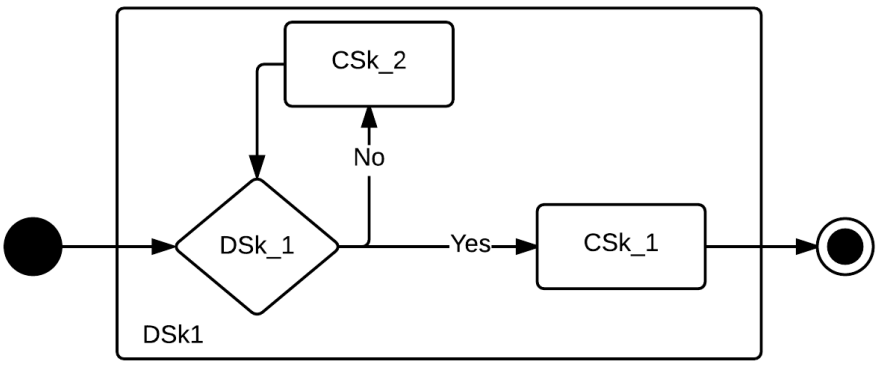

Figura 3.11: Exemplo de Decision Skill com Ciclo

Esta abordagem acaba com as limitações existentes anteriormente, a custa de uma utilização de memoria superior.

Capítulo 4

Implementação

Este capitulo descreve a implementação dos conceitos usados neste trabalho assim como a ferramenta usada para os testes e validação.

4.1 Modelo de dados das Skills

A classe *Skill* (Figura 4.1), representa uma das estruturas de mais importantes desta arquitetura. A *Skill* é uma representação das funcionalidades que um agente fornece ao sistema.

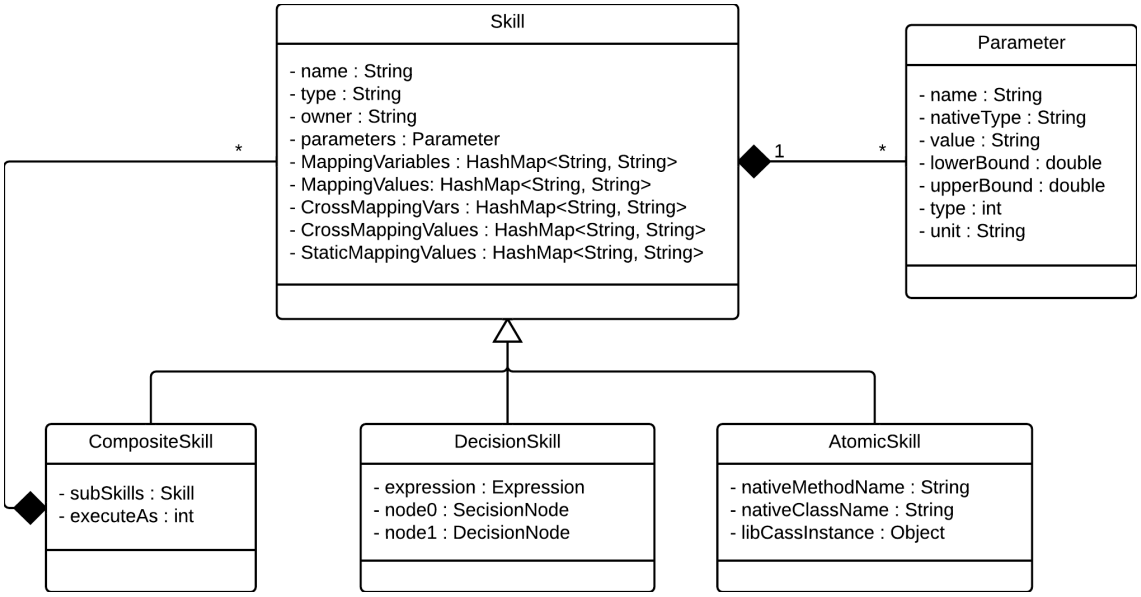

Figura 4.1: IADE - Classe Skill

Os campos mais importantes que compõem a class *Skill* são:

- **Name:** Este campo é o identificador da *Skill*, contudo, podem existir várias *Skills* com o mesmo **Name** no mesmo agente se a parametrização for diferente.
- **Type:** Identifica o tipo da *Skill* (CSk, ASk ou DSk).
- **Owner:** Este campo identifica qual o agente responsável por executar a *Skill*, seja este predefinido ou negociado. Quando este campo está vazio, o agente vai proceder a uma negociação para escolher um *Owner*.
- **Parameters:** É uma lista de *Parameters* (Figura 4.1). Um *Parameter* ou parâmetro, é uma estrutura de dados que contêm toda a informação relacionada com uma variável necessária à execução da *Skill* a que pertence ou a um output dessa mesma execução.

  A Classe *Parameter* é composta pelo *name* que é o identificador do parâmetro, o *nativeType* identifica o tipo do parâmetro, o *value* é o campo que guarda o valor do parâmetro, os *lowerBound* e *upperBound* definem os limites aceitáveis para o valor do parâmetro, o *type* indica se o o parâmetro é de output ou input, por fim o campo *unit* guarda a unidade do parâmetro (por exemplo kilograma, metro, etc).
- **MappingVariables:** Esta lista é responsável por associar os parâmetros de uma CSk ou DSk com os parâmetros das suas *SubSkills*.
- **MappingVariables:** Este campo associa os nomes com os valores dos parâmetros listados no campo **MappingVariables**, permitindo um rápido acesso quando estes são necessários na preparação da execução de uma *SubSkill*.
- **CrossMappingVariables:** Esta lista é responsável por associar parâmetros entre *SubSkills*.
- **CrossMappingValues:** Este campo associa os nomes com os valores dos parâmetros listados no campo **CrossMappingVariables**, permitindo um rápido acesso quando estes são necessários na preparação da execução de uma *SubSkill*.
- **StaticMappingValues:** Associa os nomes de parâmetros de *SubSkills* a valores estáticos.

A classe *Skill* é extendida por três classes que representam *Skills* com características distintas. A *Atomic Skill* (ASk) representa uma funcionalidade básica que está directamente relacionada com com um controlador. Para ligar a ASk ao controlador, são necessarios os campos *nativeMethodName* que guarda o nome do método a ser invocado quando a *Skill* for executada, e o *nativeClassName* que contêm o nome da classe que implementa o método. A *Composite Skill* (CSk) tem o objetivo de agrupar outras *Skills* que juntas

compõem uma funcionalidade mais complexa. As *Skills* agrupadas podem ser executadas sequencialmente ou paralelamente. Para guardar as *Skills* agrupadas, a CSk inclui a lista *subSkills*, e para indicar o tipo de execução (sequencial ou paralela), é utilizado o campo *executeAs*. A *Decision Skill* (DSk) é uma *Skill* capaz de escolher executar uma de duas *Skills* através da avaliação de uma *expression*, que não é nada mais que uma expressão booleana que recebe valores de parâmetros da DSk. Este tipo de *Skill* contêm também as classes *node0* e *node1* que encapsulam as *Skills* a ser executadas dependendo do resultado da avaliação da *expression*, o *node0* é executado caso a *expression* seja verdadeira, caso contrário, é executado o *node1*.

4.2 Modelo de dados dos Agentes do IADE

Dos agentes que compõem a biblioteca IADE, os agentes CLA e MRA são casos particulares do Mechatronic Agent (MA). O MA é uma classe abstrata (Figura 4.2) que contem toda a lógica e comunicação comum a estes agentes. Os campos mais relevantes que compõem este agente são:

- **Skills:** É uma lista de *Skills* fornecidas pelo agente. Cada *Skill* nesta lista tem toda a informação necessária à sua execução.
- **ExecutorEngine:** É responsável pela execução de *Skills*. Esta classe é abstrata e cada subclasse tem de ter a sua própria implementação.
- **NegotiatorEngine:** É responsável pela negociação das *Skills* a ser executadas por outros agentes. Tal como o ExecutorEngine, esta classe é abstrata e cada subclasse tem de ter a sua própria implementação. Esta classe tem o objetivo de avaliar as propostas recebidas durante o processo de negociação e também por enviar propostas quando um pedido de execução é recebido.
- **MyType:** Este campo contem informação sobre qual o subtipo do agente (MRA, CLA, PA ou qualquer tipo de agente definido na plataforma). É utilizado para identificar o tipo de agente durante o processo de lançamento de agentes na plataforma.
- **YellowPages:** É a classe que dá acesso aos serviços do YPA. Os serviços do YPA têm mecanismos que permitem a pesquisa de *Skills* e respetivos agentes que as executam.
- **ExecutorResponder:** Esta classe implementa a lógica que permite aos MA receberem e responderem a pedidos de execução. O protocolo de comunicação usado é o Protocolo FIPA Request (apêndice A).

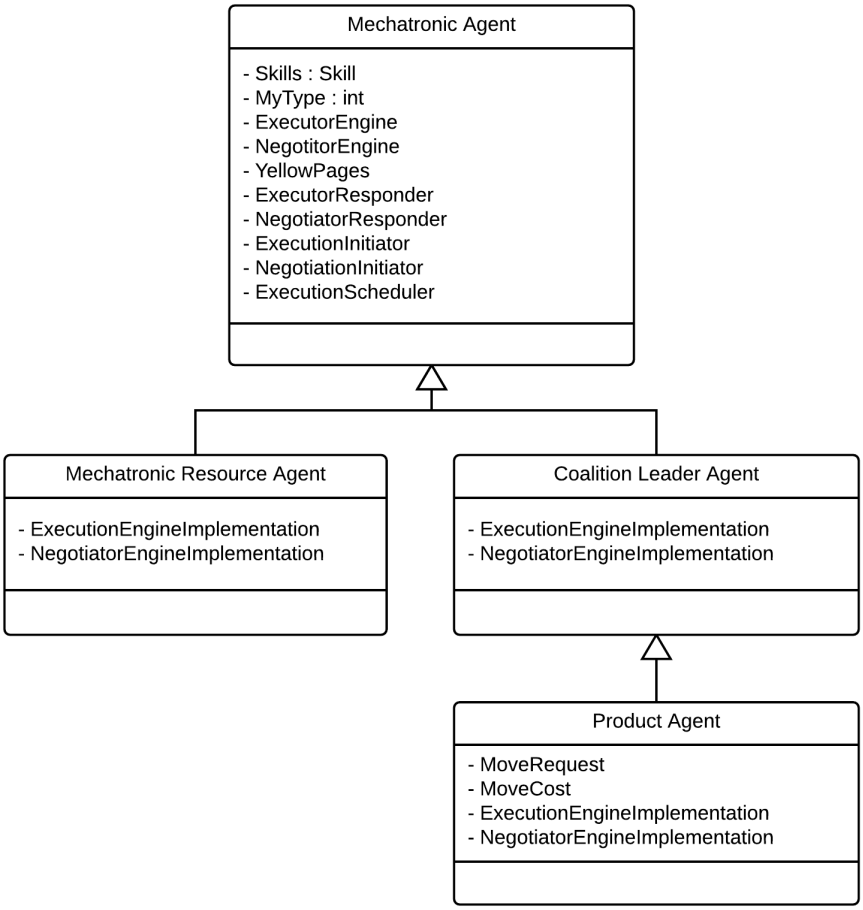

Figura 4.2: IADE - Classe Mechatronic Agent

- **NegotiatorResponder:** É a classe que fornece os mecanismos que os MA necessitam para responder aos pedidos de negociação recebidos. O protocolo utilizado é o FIPA Contract Net (apêndice B).
- **ExecutorInitiator:** Esta classe dá aos MA a capacidade de enviar pedidos de execução. O protocolo de comunicação usado é o Protocolo FIPA Request (apêndice A).
- **NegotiatorInitiator:** É a classe que dá aos MA a funcionalidade de enviar pedidos de negociação. O protocolo utilizado é o FIPA Contract Net (apêndice B).
- **ExecutionScheduler:** É o gestor de execuções. Esta classe gere a lista de execução de *Skills*, garantindo que estas são executadas pela ordem correta.

Os agentes necessitam de uma infraestrutura para publicar as suas *Skills* de como a que outros agentes as possam encontrar quando precisarem.

O YPA é o agente disponibiliza estas funcionalidades, tornando possível a interação entre os restantes agentes na plataforma. Este agente utiliza queries não bloqueantes, o

que significa que um agentes pode fazer um pedido não fica bloqueado a espera de uma resposta. Contem também um serviço de subscrição onde uma gente pode subscrever a eventos relacionados com outros agentes e receber notificações quando o estado do agente subscrito sofre uma alteração. Cada YPA guarda apenas informação relativamente à área onde está inserido, o que diminui bastante a carga no agente e consequentemente melhora os tempos de resposta.

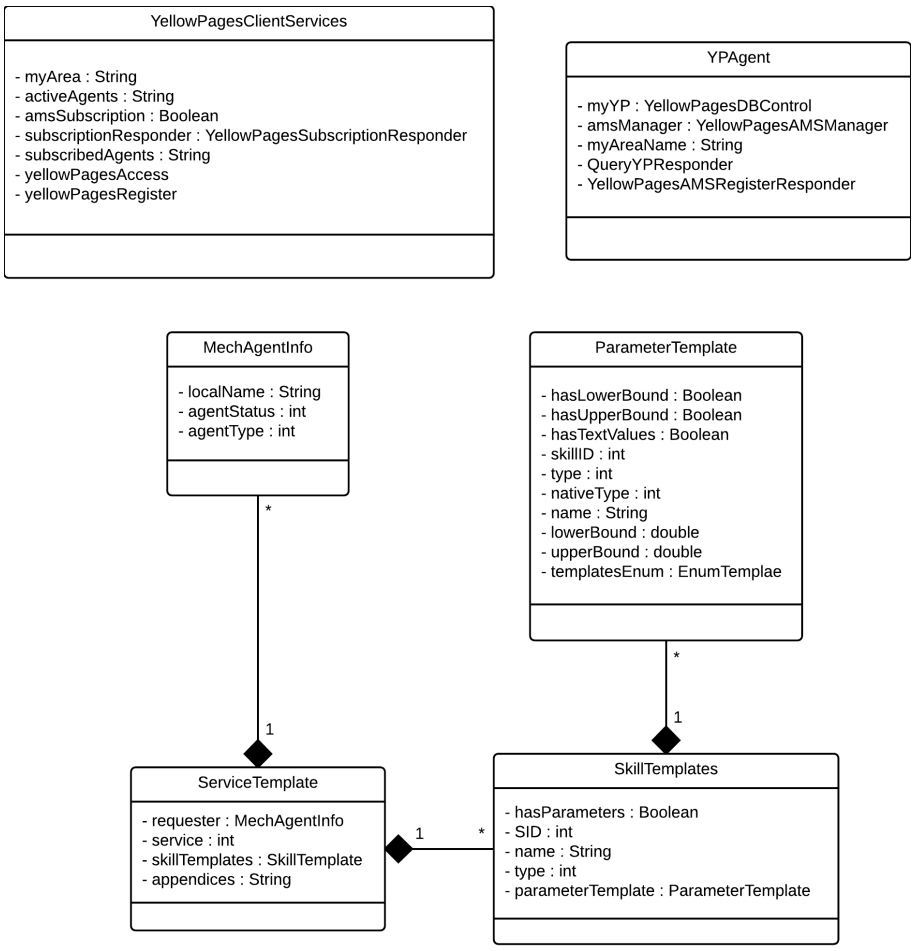

Figura 4.3: IADE - Classe Mechatronic Agent

O Yellow Pages está dividido em duas partes (Figura 4.3), o servidor (YPA) e o cliente. O YPA é composto pelos seguintes campos:

- **myYP:** Este campo é um objeto da classe YellowPagesDBControl, que é responsável por gerir a base de dados de *Skills*. ...
- **amsManager:** É responsável por gerir a lista de subscritores. Quando um agente altera o seu estado interno, esta classe envia uma mensagem de notificação a todos os subscritores.

- **YellowPagesAMSRegisterResponder:** Recebe os pedidos de subscrição, sejam eles de registo ou cancelamento.
- **myAreaName:** Identifica a área pela qual o agente é responsável.
- **QueryYPResponder:** É responsável por responder aos pedidos relacionados com agentes e *Skills* registados na sua área.

O cliente é uma classe que contem os mecanismos para realizar pedidos ao servidor anteriormente descrito:

- **myArea:** Identifica a área onde o cliente está localizado.
- **activeAgents:** Lista de agentes ativos que foram subscritos.
- **subscribedAgents:** Lista que contem todos os agentes subscritos, estejam eles ativos ou não.
- **amsSubscription:** Booleano que indica se existe alguma subscrição do servidor.
- **yellowPagesAgentAID:** Campo que guarda a identificação do servidor da área onde o cliente está localizado.
- **subscriptionResponder:** campo que recebe e processa as notificações relativas às alterações no estado dos agentes.
- **YellowPagesAccess:** Campo responsável por enviar pedidos ao servidor, incluindo registo e cancelamento de subscrições.

A comunicação entre o servidor e o cliente utiliza a classe *ServiceTemplate* que define a estrutura das suas mensagens. A classe *ServiceTemplate* define a estrutura das mensagens trocadas entre o servidor e cliente. O seu campo *service* define qual o tipo de serviço pedido. O campo *requester* é do tipo *MechAgentInfo* que é uma classe que guarda informação básica de um agente (nome, tipo e status), e o campo *skillTemplate* representa também uma classe que guarda informação relevante encontrada na classe *Skill*.

Os campos tanto da classe *SkillTemplate* como da *MechAgentInfo* podem ser utilizadas devolver informação relevante aos agentes que que enviaram um pedido, assim como podem ser utilizados pelos agentes para fornecer ao YPA informação necessária à realização do pedido.

4.2.1 Coalision Leader Agent

A execução de uma Skill num CLA é composta por dois processos principais. O primeiro é a atualização de parâmetros, e o segundo a procura de um agente para a executar, se assim for necessário.

Para atualizar os valores dos parâmetros das *SubSkills* é necessário utilizar os mapeamentos de variáveis (*mappingVariables* e *CrossMappingVariables*) para fazer a correspondência com os parâmetros da *Skill*.

A atualização dos mapeamentos de valores (*mappingValues* e *CrossMappingValues*) permite a utilização de valores dos parâmetros da *Skill* e de outputs de *SubSkills* já executadas, como inputs da *SubSkill* a executar.

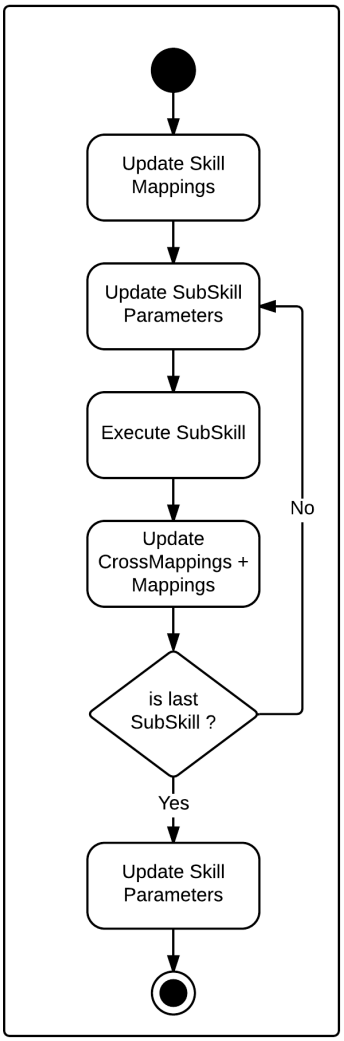

Figura 4.4: Diagrama de execução de uma Composite Skill

A Figura 4.4 representa o processo que ocorre no CLA e no PA durante a execução de uma *Skill*.

O primeiro passo na execução da *Skill* é o preenchimento dos valores da lista *mappingValues* com os valores dos parâmetros da *Skill*. De seguida são atribuídos aos os parâmetros da *subSkill* os seus valores, que são obtidos através da consulta das listas *mappingVariables* e *mappingValues*.

Com os valores dos parâmetros atribuídos, a *subSkill* tem toda a informação necessária para ser executada. Caso a *subSkill* necessite de ser executada por outro agente, é iniciada uma máquina de estados para realizar o pedido de execução (Figura reffig:SkillFSM), como foi descrito anteriormente. Se a *subSkill* for uma CSk que pertence ao próprio CLA, então é iniciado um novo processo de execução onde a *subSkill* é a *Skill* a ser executada utilizando o processo da Figura 4.4.

Após a execução da *subSkill* são atualizados os valores dos *mappingValues* e *crossMappingValues* utilizando os valores dos parâmetros de saída da *subSkill*. O processo é repetido enquanto existirem *subSkills* para executar, e quando estiverem finalmente todas executadas, os parâmetros da *Skill* são atualizados com os valores dos *mappingValues*.

Quando um CLA necessita de pedir a um agente para executar uma *Skill*, é iniciado um processo onde é procurado um agente para fazer a execução, e posteriormente é feito o pedido de execução.

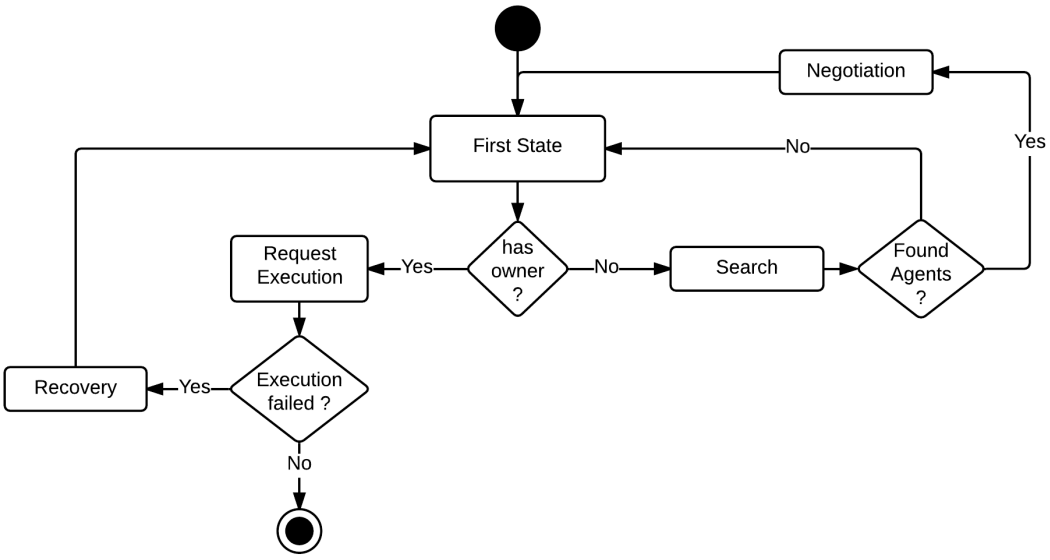

Figura 4.5: Diagrama de pedido de execução de uma Skill

A Figura 4.5 representa o processo de pedido de execução de uma *Skill* por um CLA.

No estado *First State* é verificado se o CLA já conhece um agente capaz de executar a *Skill*, se assim for, avança-se para o estado *Execution*. Caso contrário avança-se para o estado *Search* onde é contactado o YPA que irá devolver a lista de todos os agentes capazes de desempenhar a *Skill* desejada. Se o YPA devolver pelo menos um agente, então prossegue-se no estado *Negotiation* onde de entre os agentes disponíveis é escolhido aquele capaz de executar a *Skill* desejada pelo custo mais baixo baseado numa métrica predefinida. O agente escolhido fica associado à *Skill* como o agente que a vai executar.

No estado *Execution* o CLA envia uma mensagem com a informação que tem sobre a *Skill* a executar ao agente responsável pela sua execução e espera pelo fim da execução. Caso exista alguma falha na execução da *Skill*, cabe ao estado *Recovery* preparar a *Skill* para uma nova execução.

4.3 Principais interações do IADE

Nesta secção são descritas as principais interações presentes no IADE. As interações do IADE foram desenvolvidas para minimizar a quantidade de informação e mensagens trocadas, reduzindo o impacto na performance do sistema. Tendo em conta que os sistemas multi-agente tendem ser pesados do ponto de vista da comunicação.

A Figura 4.6 ilustra a a troca de mensagens que ocorre quando CLAs e MRAs são lançados no sistema. O CLA1 quando lançado, regista-se no YPA e subscreve notificações sobre agentes específicos que sejam do seu interesse. Quando o MRA se regista, o CLA1 recebe uma mensagem de notificação que o informa de uma alteração relacionada com o MRA. No caso do CLA2, o MRA já se encontra registado, portanto a notificação é recebida após o registo. Todas as interações descritas usam o protocolo *FIPA Request* (Apêndice A).

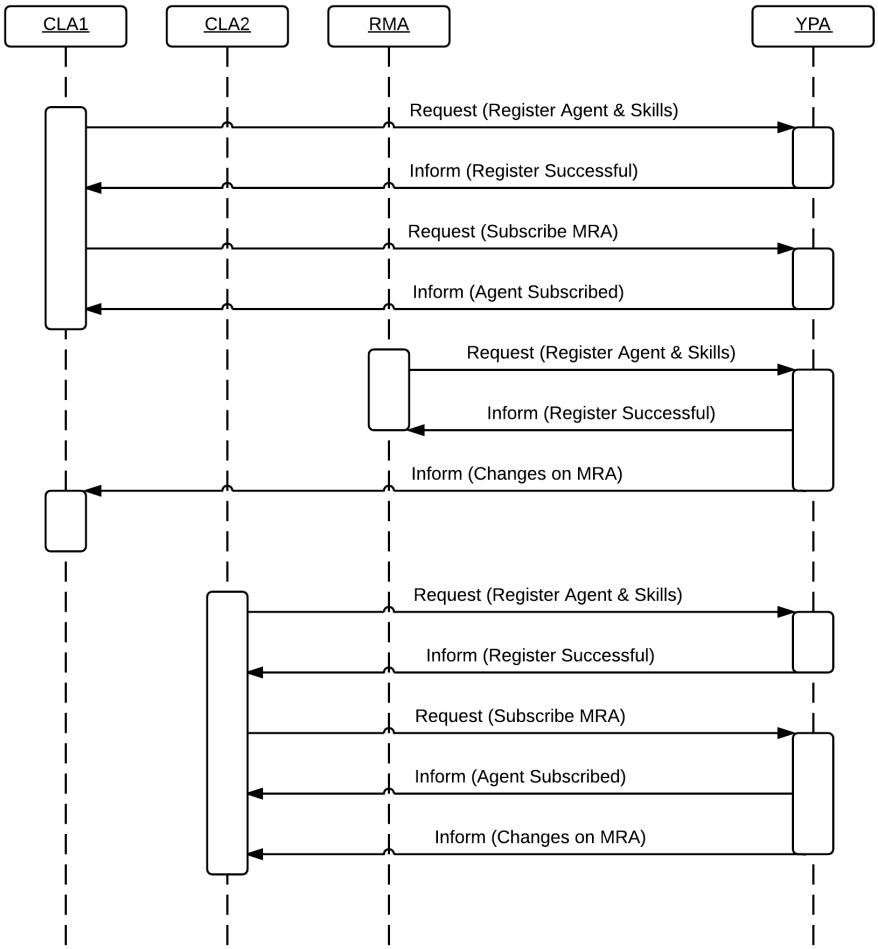

Figura 4.6: Registo e subscrição de agentes

Quando um CLA quer executar uma *Composite Skill*, o campo *Owner* é usado para determinar se é necessário fazer uma negociação. Se o *Owner* tiver um valor então o CLA escolhe esse agente para executar a *Skill*, caso contrário, o CLA inicia um processo de negociação (Figura 4.7) para escolher um agente capaz de fazer a execução. O processo de negociação utiliza o protocolo *FIPA Contract Net* (Apêndice B).

O processo de negociação do CLA (Figura 4.7) consiste em pedir uma proposta a todos os agentes capazes de executar a *Skill* desejada que pertençam a mesma área. O processo é iniciado com o pedido da lista de agentes ao YPA. Após obter a resposta, o CLA envia um pedido de proposta (CFP) a todos os agentes da lista recebida. Os agentes que receberem o pedido respondem com uma proposta de execução baseada nas suas condições operacionais, que neste caso estão relacionadas com a carga de trabalho. Quando todos os agentes responderem, o CLA seleciona a melhor proposta e envia ao agente escolhido uma mensagem *Accept-Proposal* e rejeita as restantes propostas.

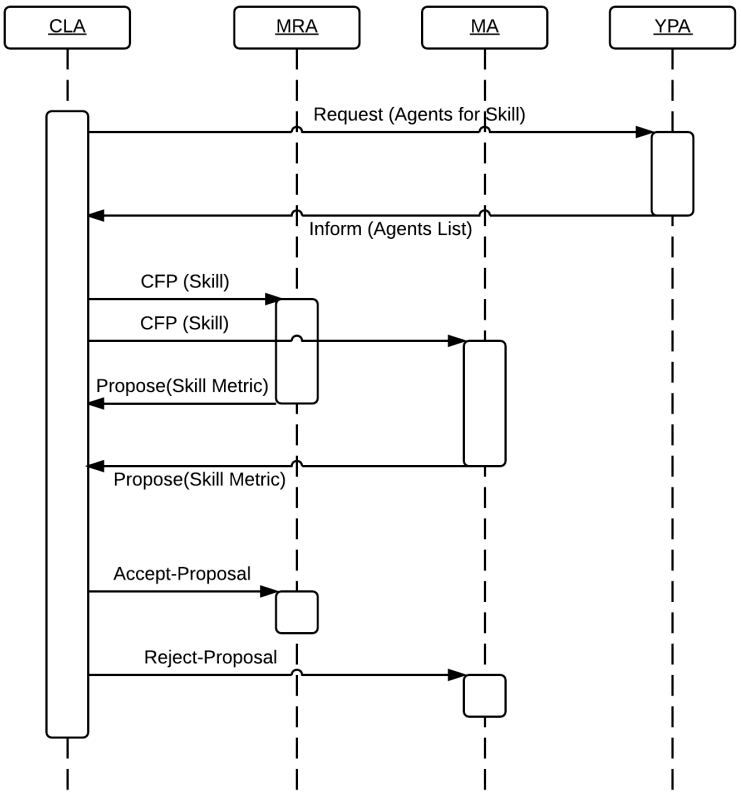

Figura 4.7: Negociação do CLA

A Figura 4.8 ilustra um pedido de execução numa situação onde um MRA é abordado por dois agentes. O primeiro agente a enviar o pedido de execução recebe uma mensagem *agree* da parte do MRA, que por sua vez inicia a execução solicitada, se mais algum agente enviar um pedido de execução ao MRA antes da execução ser terminada, o agente recebe uma mensagem *refuse* e deverá procurar outro agente para executar a sua *Skill*.

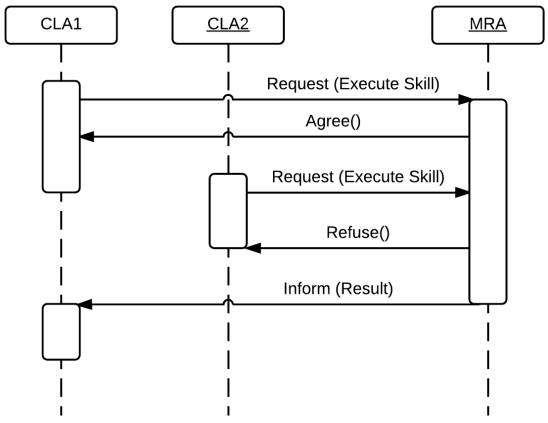

Figura 4.8: Execução de Skill

Capítulo 5

Validação e Testes

Este capitulo detalha os testes realizados para a validação da implementação.

Os testes realizados têm o objetivo de demonstrar as novas funcionalidades e desempenho inerentes à nova implementação da *Skill* no sistema.

Uma versão do sistema, a que vamos chamar *IADEv1*, anterior ao desenvolvimento deste trabalho foi também testada, com o propósito de comparar o seu desempenho com o do novo sistema, a que vamos chamar *IADEv2*.

Para a realização dos testes foi criada um configuração para cada tipo de *Skill* no sistema:

- Sequencial Skill.
- Concorrent Skill.
- Decision Skill.

Estas configurações foram ser testadas em ambos os sistemas, com o objetivo de validar o funcionamento das *Skills*, e também de identificar se existem melhorias de desempenho na sua execução.

Para validar as novas funcionalidades introduzidas neste trabalho, foram também criadas duas configurações:

- Manipulação de parâmetros.
- Recursividade.

Como estas novas funcionalidades não podem ser executadas pelo sistema *IADEv1*, estas configurações foram apenas testadas no sitema *IADEv2*. Como tal, objetivo destes casos é teste é simplesmente validar o funcionamento das respetivas funcionalidades.

Por fim foi feito um teste de carga para estudar a evolução do desempenho dos sistemas com o aumento do número de *agentes*.

Este capitulo está dividido numa secção por cada caso de teste, que por sua vez então divididas em duas partes, descrição e resultados. Na primeira descreve-se a configuração do teste, e na ultima apresenta-se os resultados.

5.1 Execução de Sequencial Skill

5.1.1 Descrição

Para testar o desempenho das *Sequencial Skills* em ambas as versões do sistema, foi criado um *PA* que irá executar uma única *Sequencial Skill*. Esta *Sequencial Skill*, apresentada abaixo na Figura 5.1, é composta por três *ASk* que são executadas sequencialmente.

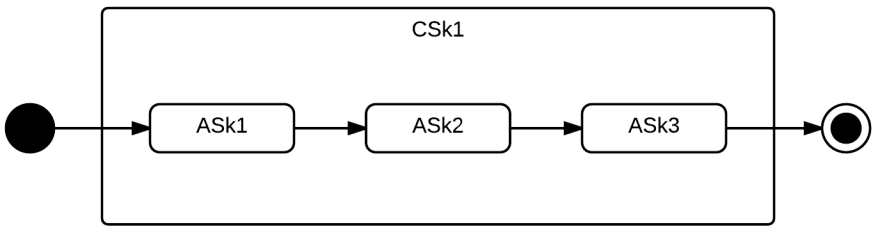

Figura 5.1: Sequencial Composite Skill

Em baixo, a Tabela 5.1, contem os parâmetros de configuração das várias *Skills*. A *CSk* **CSk1** tem um parâmetro de entrada e irá executar as suas *SubSkills* sequencialmente. As *ASk* têm um parâmetro de entrada que é alimentado com o parâmetro **value** da *CSk* **CSk1**.

	Input Param.	**Output Param.**	**I/O Param.**	**Func./Expr.**
CSk1	value	-	-	Sequencial
ASk1	In	-	-	-
ASk2	In	-	-	-
ASk3	In	-	-	-

	Mapping	**CrossMapping**	**StaticMapping**
CSk1	ASk1.In <- value ASk2.In <- value ASk3.In <- value	-	-

Tabela 5.1: Configuração da Sequencial CSk

5.1.2 Resultados

O cenário anteriormente descrito, foi testado trinta vezes para cada versão do sistema. Como pode ser visto na Tabela 5.2, o versão *IADEv2* apresenta melhorias no desempenho na execução deste tipo de *Skills*.

	Amostragem	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
IADEv1	30	8,60 ms	2,20 ms	6 ms	15 ms
IADEv2	30	6,73 ms	2,12 ms	5 ms	12 ms

Tabela 5.2: Tempo de execução do teste de Sequencial Skill

5.2 Execução de Concurrent Skill

5.2.1 Descrição

Para testar o desempenho das *Concurrent Skills* em ambas as versões do sistema, foi criado um *PA* que irá executar uma única *Concurrent Skill*. Esta *Concurrent Skill*, apresentada abaixo na Figura 5.2, é composta por três *ASk* que são executadas concorrentemente.

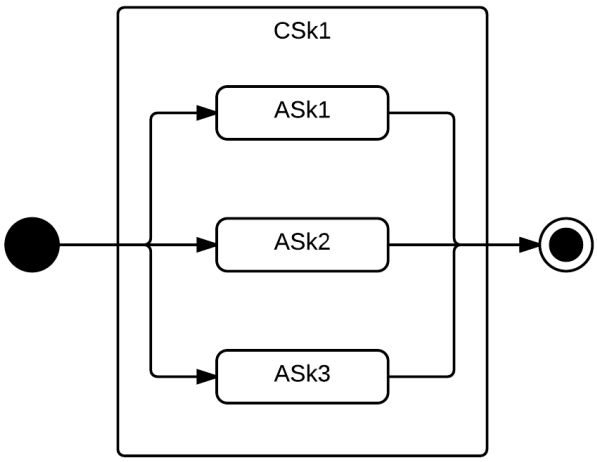

Figura 5.2: Concurrent Composite Skill

Em baixo, a Tabela 5.3, contem os parâmetros de configuração das várias *Skills*. A *CSk* **CSk1** tem um parâmetro de entrada e irá executar as suas *SubSkills* concorrentemente. As *ASk* têm um parâmetro de entrada que é alimentado com o parâmetro **value** da *CSk* **CSk1**. A execução da *Skill* termina quando todas as suas *SubSkills* terminarem a sua execução.

	Input Param.	**Output Param.**	**I/O Param.**	**Func./Expr.**
CSk1	value	-	-	Concurrent
ASk1	In	-	-	-
ASk2	In	-	-	-
ASk3	In	-	-	-

	Mapping	**CrossMapping**	**StaticMapping**
CSk1	ASk1.In <- value ASk2.In <- value ASk3.In <- value	-	-

Tabela 5.3: Configuração da Concurrent CSk

5.2.2 Resultados

O cenário anteriormente descrito, foi testado trinta vezes para cada versão do sistema. Como pode ser visto na Tabela 5.4, o versão *IADEv2* apresenta melhorias no desempenho na execução deste tipo de *Skills*.

	Amostragem	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
IADEv1	30	7,90 ms	2,19 ms	6 ms	16 ms
IADEv2	30	6,47 ms	2,13 ms	4 ms	12 ms

Tabela 5.4: Tempo de execução do teste de Concurrent Skill

5.3 Execução de Decision Skill

5.3.1 Descrição

Para testar o desempenho das *Decision Skills* em ambas as versões do sistema, foi criado um *PA* que irá executar uma única *Decision Skill*. Esta *Decision Skill*, apresentada abaixo na Figura 5.3, é composta por duas *ASk* das quais apenas uma será executada dependendo do parâmetro de entrada fornecido.

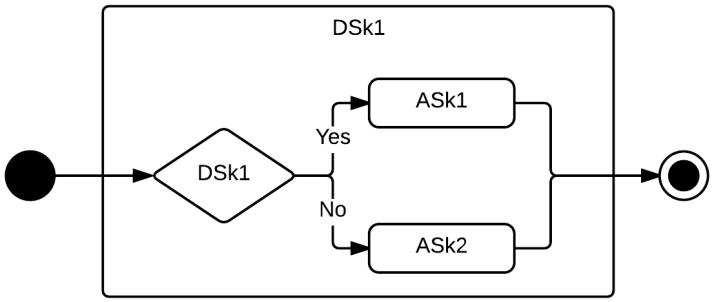

Figura 5.3: Decision Skill

A configuração da *Skill* para este caso é bastante simples (Tabela 5.5), a **DSk1** tem um parâmetro do tipo *Input* chamado **value**, que é usado com parâmetro de entrada para ambas as *SubSkills*. A *SubSkill* a executar e escolhida pela validação **value == true**.

	Input Param.	**Output Param.**	**I/O Param.**	**Func./Expr.**
DSk1	value	-	-	value == true

Tabela 5.5: Configuração da Decision Skill

5.3.2 Resultados

O cenário anteriormente descrito, foi testado trinta vezes para cada versão do sistema. Como pode ser visto na Tabela 5.6, o versão *IADEv2* apresenta melhorias no desempenho na execução deste tipo de *Skills*.

	Amostragem	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
IADEv1	30	4,03 ms	1,52 ms	2 ms	8 ms
IADEv2	30	3,30 ms	1,12 ms	2 ms	6 ms

Tabela 5.6: Tempo de execução do teste de Decision Skill

5.4 Manipulação de parâmetros

5.4.1 Descrição

Para validar e testar o desempenho da Manipulação de parâmetros, foi criado um *PA* que irá executar uma única *Sequencial Skill*. A Figura 5.4 apresenta uma CSk em modo de execução sequencial, idêntica à encontrada na Figura 5.1 estudada no caso de teste das *Sequencial Skills*.

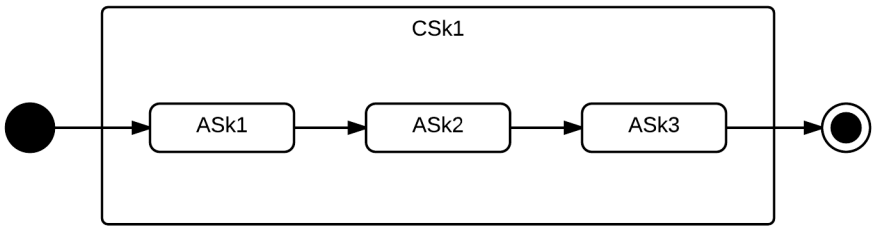

Figura 5.4: Sequencial CSk com manipulação de parâmetros

A grande diferença entre este caso de teste e o teste das *Sequencial Skills*, está no mapeamento entre as suas *SubSkills*. Como se pode verificar na Tabela 5.7, a **CSk1** tem um parâmetro do tipo *Input* que alimenta o parâmetro **In** de **ASk1**. Esta, quando executada, incrementa o valor do seu parâmetro **In** e devolve o valor através do parâmetro **Out**.

	Input Param.	Output Param.	I/O Param.	Func./Expr.
CSk1	valueIn	valueOut	-	Sequencial
ASk1	In	Out	-	Out = In + 1
ASk2	In	Out	-	Out = In + 1
ASk3	In	Out	-	Out = In + 1

	Mapping	CrossMapping	StaticMapping
CSk1	ASk1.In <- valueIn valueOut <- ASk3.Out	ASk2.In <- ASk1.Out ASk3.In <- ASk2.Out	-

Tabela 5.7: Configuração da Sequencial CSk com manipulação de parâmetros

Por sua vez, antes da execução de **ASk2**, o seu parâmetro **In** é atualizado com o valor do parâmetro **Out** anteriormente obtido (ASk2.In <- ASk1.Out). Tendo em conta que tanto **ASk2** como **ASk3** são idênticas a **ASk1**, o processo descrito anteriormente repete-se para estas. Por fim, após a execução de **ASk3**, o valor do seu parâmetro **Out** atualiza o parâmetro **valueOut** de **CSk1** terminando assim a execução.

5.4.2 Resultados

A manipulação de parâmetros é uma funcionalidade introduzida na versão *IADEv2*, como tal, não foi possível correr este teste na versão *IADEv1*.

	Amostragem	Média	Desvio Padrão	Mínimo	Máximo
IADEv1	-	-	-	-	-
IADEv2	30	7,97 ms	2,39 ms	6	14

Tabela 5.8: Tempo de execução do teste de manipulação de parâmetros

Ao comparar os resultados obtido com os resultados dos testes anteriores, onde foram executadas três *ASks*, pode verificar-se que o tempos minimos de execução são muito próximos aos da a versão *IADEv2*. Por outro lado os tempos médio e máximo de execução são mais elevados quando comparados com a versão *IADEv2*, mas contiuam mais baixos do que os resultados da versão *IADEv1*.

5.5 Recursividade

5.5.1 Descrição

Este parâmetro centra-se na possibilidade das *Skills* apresentarem um comportamento cíclico, e, avaliar o seu desempenho caso o mesmo exista. Como se pode observar através da Figura 5.5 e da Tabela 5.9 estas representam, a *Skill* e respetiva configuração, criada com o intuito acima referido.

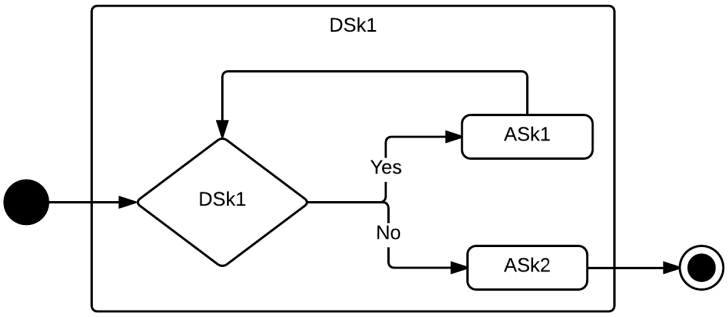

Figura 5.5: Decision Skill Recursiva

A Tabela 5.9 apresenta os parametros, funcionalidades e mapeamentos das *Skills* que se encontram na Figura 5.5. No caso das DSk, a funcionalidade representa a expressão a avaliar.

A *Decision Skill* **DSk1** tem um parâmetro do tipo *I/O*, o que significa que este é um parâmetro de entrada da *Skill* que pode sofrer alterações ao longo da execução e ser utilizado para alimentar um parâmetro de entrada de outra *Skill*.

	Input Param.	**Output Param.**	**I/O Param.**	**Func./Expr.**
DSk1	-	-	startVal	startVal < 3
ASk1	In	Out	-	Out = In + 1
ASk2	-	-	-	-

	Mapping	**CrossMapping**	**StaticMapping**
DSk1	ASk1.In <- startVal	startVal <- ASk1.Out	

Tabela 5.9: Configuração da Decision Skill Recursiva

A *Atomic Skill* **ASk1** tem um parâmetro do tipo *Input* e outro do tipo *Output*, como se pode verificar nos mapeamentos, o parâmetro **In** recebe o valor do parâmetro **startVal**.

O objetivo desta *Skill* é incrementar o valor de **In** e devolver o resultado através do parâmetro **Out**. Após a execução desta *Skill*, o valor do parâmetro **startVal** é atualizado com o valor encontrado em **Out**, sendo posteriormente utilizado na re-execução da *Decision Skill* **DSk1**. Sendo assim a, *Decision Skill* **DSk1** é re-executada até que o valor do seu parâmetro **startVal** seja igual a três, sendo nessa altura executada a *Atomic Skill* **ASk2** terminando a execução.

5.5.2 Resultados

Como as Skills da versão *IADEv1* do sistema não suportam recursividade devido ao comportamento estático dos seus mapeamentos, este teste só foi executado para a nova versão *IADEv2*.

	Amostragem	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
IADEv1	-	-	-	-	-
IADEv2	30	7,83 ms	2,48 ms	5	15

Tabela 5.10: Tempo de execução do teste recursivo

Ao comparar os resultados obtidos com os resultados dos testes anteriores, onde foram executadas três *ASks*, pode verificar-se que o tempos minimos de execução são muito próximos aos da a versão *IADEv2*. Por outro lado os tempos médio e máximo de execução são mais elevados quando comparados com a versão *IADEv2*, mas contiuam mais baixos do que os resultados da versão *IADEv1*.

5.6 Teste de Carga

5.6.1 Descrição

Este caso tem o objetivo de estudar a evolução do desempenho do sistema com o aumento do número de agentes. Para tal, foi criado um cenário dividido em três níveis em que cada nível possui o mesmo número de agentes:

- Nível 1 - Este nível contem apenas RAs.
- Nível 2 - Constituído apenas por CLAs.
- Nível 3 - Onde se encontram todos os PAs.

Para cada um destes níveis foi criada uma *Skill*. Para o nível 1 foi criada a *Atomic Skill* **ASk_1**, esta *Skill* tem um tempo de execução de 500 ms. No nível 2 foi configurada a *Sequencial Skill* **CSk_1**, que executa quatro **ASk_1** (ver Figura 5.6).

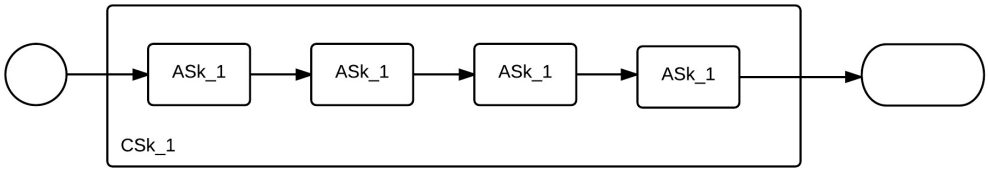

Figura 5.6: Composite Skill executada pelo CLA

O nível 3 é constituído por PAs que executam *Skills* **PA_1**. Esta *Skill* é composta por duas *SubSkills* **CSk_1** (ver Figura 5.7).

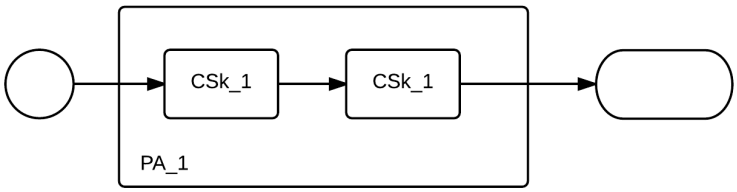

Figura 5.7: Composite Skill executada pelo PA

Este cenário foi executado para quatro cargas diferentes em cada sistema. Primeiro foi executado com três agentes (um em cada nível) e posteriormente com 30, 45 e 60 agentes.

Como referido em [34], o tempo de execução teórico de uma skill pedida por um **PA**, em situações otimas (sem concorrência), é dado pela equação 5.1. Nesta equação, as Interações (I) são o número de ligações entre as *Skills*. O tempo de viagem entre agentes (TTB) é o tempo de processamento de uma mensagem desde que é enviada até ser recebida. O tempo de execução dos MRAs ($ExecTime_R$) e CLAs ($ExecTime_{cla}$) é multiplicado pelo número de *Skills* executadas.

$$ExecTime = 2 * I * TTB + N_{SkillResources} * ExecTime_R + N_{clas} * ExecTime_{cla} \quad (5.1)$$

Durante a execução da *Skill* **PA_1** são executadas um total de 11 *Skills* (8 ASk e 3 CSk). O que resulta no seguinte tempo de execução:

$$ExecTime = 0 + 8 * 500ms + 3 * 2ms = 4006ms \quad (5.2)$$

Na equação 5.2, 500ms é o tempo de execução de uma *Skill* **ASk_1** e 2ms o tempo de execução de uma *Skill* **CSk_1**. Como neste cenário os agentes correm no mesmo computador, não há atrasos significativos na comunicação entre os agentes. Nestas circunstancias, o TTB é negligenciável, tomando o valor zero na equação.

5.6.2 Resultados

O cenário anteriormente descrito, foi testado trinta vezes para ambas as versões do sistema, *IADEv1* e *IADEv2*, com quatro cargas distintas:

- 3 *Agentes* - 1 *Agente* por nível.
- 30 *Agentes* - 10 *Agentes* por nível.
- 45 *Agentes* - 15 *Agentes* por nível.
- 60 *Agentes* - 20 *Agentes* por nível.

As tabelas em baixo apresentam os resultados obtidos nos quatro cenários para ambos os sistemas:

	3 Agentes	**30 Agentes**	**45 Agentes**	**60 Agentes**
Amostragem	30	30	30	30
Média	4025 ms	6693 ms	8476 ms	11401 ms
Desvio Padrão	43 ms	453 ms	754 ms	404 ms
Mínimo	4012 ms	5709 ms	7231 ms	10769 ms
Máximo	4249 ms	7507 ms	9932 ms	12181 ms

Tabela 5.11: Tempo de execução no sistema *IADEv1*

	3 Agentes	**30 Agentes**	**45 Agentes**	**60 Agentes**
Amostragem	30	30	30	30
Média	4019 ms	5927 ms	7051 ms	9011 ms
Desvio Padrão	6 ms	437 ms	735 ms	299 ms
Mínimo	4013 ms	4975 ms	5856 ms	8474 ms
Máximo	4040 ms	6659 ms	8617 ms	9915 ms

Tabela 5.12: Tempo de execução no sistema *IADEv2*

Como se pode verificar, tanto pelas tabelas como pela figura abaixo, com uma carga de 3 *Agentes* ambos os sistemas têm um desempenho próximo do valor teórico calculado na equação 5.2.

Com o aumento do número de *Agentes*, o desempenho de ambas as versões do sistema vai reduzindo, passando a versão *IADEv2* a apresentar um desempenho superior ao da versão anterior.

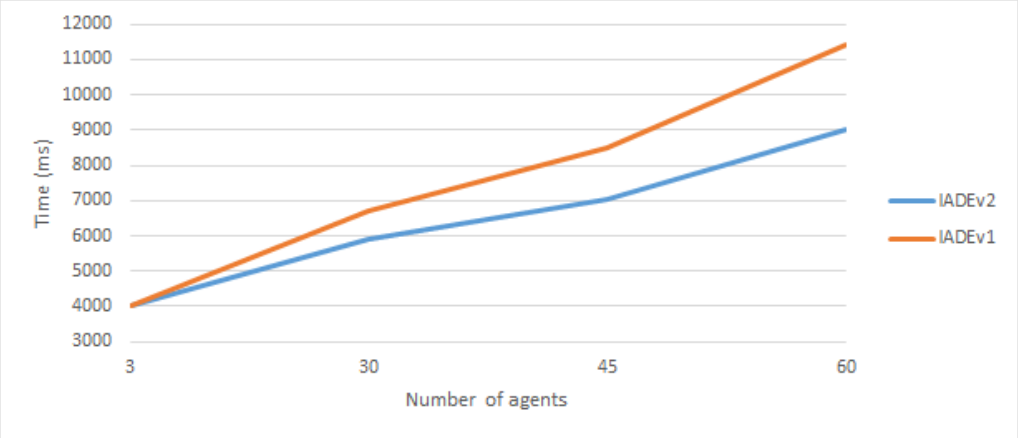

Figura 5.8: Comparação do tempo de execução

Com este resultado pode concluir-se que a versão *IADEv2* tem um desempenho superior à versão *IADEv1*.

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Conclusões

O trabalho desenvolvido atingiu vários objetivos.

Em primeiro lugar, foram implementados dois novos casos de uso, a troca de parâmetros entre *SubSkills*, e a recursividade. Estes caos de uso não causaram um impacto negativo no desempenho do sistema, e trazem a possibilidade de criar novos cenários de configuração que não eram anteriormente suportados, melhorando assim a usabilidade.

Em segundo lugar, foi estudado o desempenho dos sistemas. Ambos os sistemas foram submetidos a um teste de carga no qual o sistema *IADEv1* apresentou uma degradação de desempenho muito superior à do sistema *IADEv2*, concluindo assim, que o sistema *IADEv2* tem um desempenho superior.

6.2 Trabalho Futuro

Como trabalho futuro é necessário testar a nova versão do sistema num cenário distribuído.

É também necessário estudar o uso da troca de parâmetros entre *SubSkills* em cenários complexos, compostos por vários níveis de CLAs, onde esta funcionalidade pode ser usada para juntar várias *Skills* numa única *Skill* mais complexa. Esta prática pode reduzir tanto o número de CLAs do sistema como o número de mensagens trocadas entre Agentes, o que pode resultar numa melhoria tanto do desempenho como da escalabilidade do sistema.

Bibliografia

[1] R. F. Babiceanu e F. F. Chen. "Development and applications of holonic manufacturing systems: a survey". Em: *Journal of Intelligent Manufacturing* 17.1 (2006), pp. 111–131.

[2] J. Barata e L. M. Camarinha-Matos. "Coalitions of manufacturing components for shop floor agility - The CoBaSA architecture". Em: *International Journal of Networking and Virtual Organisations* 2.1 (2003), pp. 50–77.

[3] J. Barata, L. Ribeiro e M. Onori. "Diagnosis on Evolvable Production Systems". Em: *2007 IEEE International Symposium on Industrial Electronics*. 2007, pp. 3221–3226. DOI: 10.1109/ISIE.2007.4375131.

[4] J. Barata. "The cobasa architecture as an answer to shop floor agility". Em: *Manufacturing the Future. Concepts-Technologies-Visions*. 2006, p. 908.

[5] J. Barata e M. Onori. "Evolvable assembly and exploiting emergent behaviour". Em: *Industrial Electronics, 2006 IEEE International Symposium on*. Vol. 4. IEEE. 2006, pp. 3353–3360.

[6] Z. M. Bi, S. Y. T. Lang, W. Shen e L. Wang. "Reconfigurable manufacturing systems: the state of the art". Em: *International Journal of Production Research* 46.4 (2008), pp. 967–992.

[7] P. T. Bolwijn e T. Kumpe. "Manufacturing in the 1990s—productivity, flexibility and innovation". Em: *Long Range Planning* 23.4 (1990), pp. 44–57.

[8] S. Bussmann e D. C. McFarlane. "Rationales for holonic manufacturing control". Em: *Proc. of Second Int. Workshop on Intelligent Manufacturing Systems*. 1999, pp. 177–184.

[9] L. M. Camarinha-Matos e H. Afsarmanesh. "Virtual Enterprise Modeling and Support Infrastructures: Applying Multi-agent System Approaches". Em: *Selected Tutorial Papers from the 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School on Multi-Agent Systems and Applications*. EASSS '01. London, UK, UK: Springer-Verlag, 2001, pp. 335–364.

[10] L. M. Camarinha-Matos. "Multi-agent systems in virtual enterprises". Em: *International Conference on AI, Simulation and Planning in High Autonomy Systems*. 2002, pp. 27–36.

[11] G. Cândido, A. W. Colombo, J. Barata e F. Jammes. "Service-oriented infrastructure to support the deployment of evolvable production systems". Em: *Industrial Informatics, IEEE Transactions on* 7.4 (2011), pp. 759–767.

[12] J.-L. Chirn e D McFarlane. "Application of the holonic component-based approach to the control of a robot assembly cell". Em: *IEEE Conference on Robotics and Automation, San Fransisco*. Vol. 268. 2000.

[13] J.-L. Chirn e D. C. McFarlane. "A holonic component-based approach to reconfigurable manufacturing control architecture". Em: *Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on*. IEEE. 2000, pp. 219–223.

[14] G. Cândido e J. Barata. "A Multiagent Control System for Shop Floor Assembly". Em: *HoloMAS*. Ed. por V. Mavrík, V. Vyatkin e A. W. Colombo. Vol. 4659. Lecture Notes in Computer Science. Springer, 2007, pp. 293–302.

[15] R. Galan, J. Racero, I. Eguia e J. Garcia. "A systematic approach for product families formation in Reconfigurable Manufacturing Systems". Em: *Robotics and Computer-Integrated Manufacturing* 23.5 (2007), pp. 489–502.

[16] A. Koestler. *The ghost in the machine*. 1968.

[17] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy e H. V. Brussel. "Reconfigurable Manufacturing Systems". Em: *CIRP Annals - Manufacturing Technology* 48.2 (1999), pp. 527–540.

[18] P. Leitao, A. W. Colombo e F. J. Restivo. "ADACOR: A Collaborative Production Automation and Control Architecture". Em: *IEEE Intelligent Systems* 20.1 (2005), pp. 58–66.

[19] P. Leitão. "Agent-based distributed manufacturing control: A state-of-the-art survey". Em: *Engineering Applications of Artificial Intelligence* 22.7 (2009), pp. 979–991.

[20] P. Leitão e F. Restivo. "ADACOR: A holonic architecture for agile and adaptive manufacturing control". Em: *Computers in Industry* 57.2 (2006), pp. 121–130.

[21] J. Li, X. Dai, Z. Meng, J. Dou e X. Guan. "Rapid design and reconfiguration of Petri net models for reconfigurable manufacturing cells with improved net rewriting systems and activity diagrams". Em: *Computers & Industrial Engineering* 57.4 (2009), pp. 1431–1451.

[22] V. Mařík, P. Vrba, K. H. Hall e F. P. Maturana. "Rockwell automation agents for manufacturing". Em: *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. ACM. 2005, pp. 107–113.

[23] M. G. Mehrabi, A. G. Ulsoy e Y. Koren. "Reconfigurable manufacturing systems and their enabling technologies". Em: *International Journal of Manufacturing Technology and Management* 1.1 (2000), pp. 114–131.

[24] L. Monostori, J. Váncza e S. R. T. Kumarac. "Agent-Based Systems for Manufacturing". Em: *CIRP Annals - Manufacturing Technology* 55.2 (2006), pp. 697–720.

[25] P. Neves e J. Barata. "Evolvable production systems". Em: *Assembly and Manufacturing, 2009. ISAM 2009. IEEE International Symposium on*. Nov. de 2009, pp. 189–195.

[26] M. Onori, D. Semere e J. Barata. "Evolvable Assembly Systems: From Evaluation to Application". Em: *Innovation in Manufacturing Networks* (2008), pp. 205–214.

[27] N Papakostas, D Mourtzis, K Bechrakis, G Chryssolouris, D Doukas e R Doyle. "A flexible agent based framework for manufacturing decision-making". Em: *9th Flexible Automation and Intelligent Manufacturing Conference (FAIM), Tilburg*. 1999.

[28] L. Ribeiro, R. Rosa e J. Barata. "A structural analysis of emerging production systems". Em: *IEEE 10th International Conference on Industrial Informatics*. Jul. de 2012, pp. 223–228.

[29] L. Ribeiro e J. Barata. "Re-thinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging IT based production paradigms". Em: *Computers in Industry* 62.7 (2011), pp. 639–659.

[30] L. Ribeiro, J. Barata, G. Cândido e M. Onori. "Evolvable production systems: an integrated view on recent developments". Em: *Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology*. Springer. 2010, pp. 841–854.

[31] L. Ribeiro, A. Rocha e J. Barata. "A product handling technical architecture for multiagent-based mechatronic systems". Em: *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. IEEE. 2012, pp. 4342–4347.

[32] L. Ribeiro, R. Rosa, A. Cavalcante e I Barata. "IADE–IDEAS agent development environment: lessons learned and research directions". Em: *CIRP Conference on Assembly Technologies and Systems (CATS 2012), Ann Arbor*. 2012.

[33] A. Rocha. "An agent based architecture for material handling systems". Tese de mestrado. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2013.

[34] R. Rosa. "Assessing Self-Organization and Emergence in Evolvable Assembly Systems (EAS)". Tese de mestrado. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2013.

[35] K. Ryu e M. Jung. "Agent-based fractal architecture and modelling for developing distributed manufacturing systems". Em: *International Journal of Production Research* 41.17 (2003), pp. 4233–4255.

[36] D. Semere, J. Barata e M. Onori. "Evolvable assembly systems: Developments and advances". Em: *Assembly and Manufacturing, 2007. ISAM'07. IEEE International Symposium on*. IEEE. 2007, pp. 282–287.

[37] W. Shen, L. Wang e Q. Hao. "Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey". Em: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 36.4 (jul. de 2006), pp. 563–577.

[38] W. Shen, Q. Hao, H. J. Yoon e D. H. Norrie. "Applications of agent-based systems in intelligent manufacturing: An updated review". Em: *Advanced Engineering Informatics* 20.4 (2006), pp. 415 –431.

[39] A Tharumarajah. "Comparison of the bionic, fractal and holonic manufacturing system concepts". Em: *International Journal of Computer Integrated Manufacturing* 9.3 (1996), pp. 217–226.

[40] K. Ueda. "A concept for bionic manufacturing systems based on DNA-type information". Em: *Proceedings of the IFIP TC5/WG5. 3 Eight International PROLAMAT Conference on Human Aspects in Computer Integrated Manufacturing*. North-Holland Publishing Co. 1992, pp. 853–863.

[41] K. Ueda, I. Hatono, N. Fujii e J. Vaario. "Reinforcement Learning Approaches to Biological Manufacturing Systems". Em: *CIRP Annals-Manufacturing Technology* 49.1 (2000), pp. 343–346.

[42] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts e P. Peeters. "Reference architecture for holonic manufacturing systems: PROSA". Em: *Computers in Industry* 37.3 (1998), pp. 255–274.

[43] P. Vrba, M. Radakovic, M. Obitko e V. Marík. "Semantic Extension of Agent-Based Control: The Packing Cell Case Study". Em: *HoloMAS'09*. 2009, pp. 47–60.

[44] P. Vrba, P. Tichy, V. Marik, K. H. Hall, R. J. Staron, F. P. Maturana e P. Kadera. "Rockwell Automation's Holonic and Multiagent Control Systems Compendium". Em: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 41.1 (2011), pp. 14–30.

APÊNDICE A

PROTOCOLO FIPA REQUEST

O *FIPA Request*, é um protocolo que permite a um agente pedir a outro agente a execução de uma tarefa. A Figura A.1 ilustra o protocolo. Neste protocolo, o agente *Initiator* inicia um pedido ao agente *Participant*.

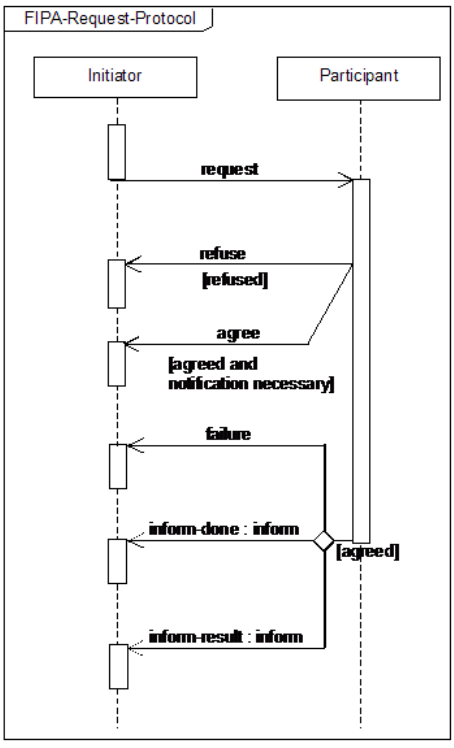

Figura A.1: Protocolo FIPA Request

O *Participant* pode aceitar ou recusar esse pedido. No caso de uma recusa, o agente envia uma mensagem do tipo *refuse* e a comunicação é terminada.

Caso o pedido seja aceite, o agente *Participant* envia uma mensagem do tipo *agree*. As mensagens *agree* e *refuse* são opcionais, portanto o agente pode escolher enviar apenas uma mensagem do tipo *inform* ou *failure*.

No caso de uma mensagem *agree*, o *Participant* processa o pedido que recebeu, e assim que a tarefa esteja concluída, envia uma mensagem *inform* em caso de sucesso ou uma mensagem *failure* caso o pedido não tenha sido executado com sucesso.

APÊNDICE B

Protocolo FIPA Contract Net

O Protocolo *Contract Net* (Figura B.1), permite a um agente iniciar uma negociação com vários agentes. Este protocolo é iniciado por parte do agente *Initiator* que começa por enviar uma mensagem pedindo uma proposta a N agentes *Participant*. Cada *Participant* responde ao pedido com uma mensagem do tipo *refuse* caso recuse o pedido, ou com uma mensagem do tipo *proposal*.

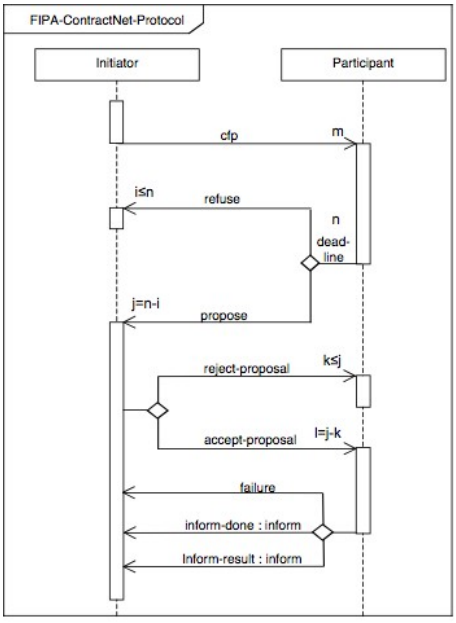

Figura B.1: Protocolo FIPA Contract Net

Quando o *Initiator* receber todas as respostas (propostas e recusas) vai avaliar todas as propostas. Para as propostas recusadas, o *Initiator* envia uma mensagem *refuse-proposal* para os *Participants* correspondentes e a comunicação entre estes agentes termina.

Para todos os *Participants* avaliados com sucesso, o *Initiator* vai enviar uma mensagem *accept-proposal*. Após receber uma mensagem *accept-proposal*, o *Participant* vai processar a tarefa pedida pelo *Initiator*. Quando a tarefa estiver concluída, os *Participants* respondem com uma mensagem *inform* em caso de sucesso, ou *failure* caso contrário.