UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ANDRÉ LUIZ DUARTE CAVALCANTE

ARQUITETURA BASEADA EM AGENTES E AUTO-ORGANIZÁVEL PARA A MANUFATURA

Porto Alegre
2012

ANDRÉ LUIZ DUARTE CAVALCANTE

ARQUITETURA BASEADA EM AGENTES E AUTO-ORGANIZÁVEL PARA A MANUFATURA

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Doutor em Engenharia Elétrica.

Área de concentração: Controle e Automação

ORIENTADOR: Prof. Dr.Ing. Carlos Eduardo Pereira

Porto Alegre
2012

ANDRÉ LUIZ DUARTE CAVALCANTE

ARQUITETURA BASEADA EM AGENTES E AUTO-ORGANIZÁVEL PARA A MANUFATURA

Esta tese foi julgada adequada para a obtenção do título de Doutor em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr.Ing. Carlos Eduardo Pereira, UFRGS
Doutor em Engenharia Elétrica pelo Instituto de Automação Industrial e Engenharia de Software da Universidade de Stuttgart – Stuttgart, Alemanha

Banca Examinadora:

Prof. Dr. Rafael Bordini, PUCRS
Doutor pela pela University College London – Londres, Inglaterra

Prof. Dr. João Carlos Ferreira, UFSC
Doutor pela University of Manchester Institute of Science and Technology – Manchester, Inglaterra

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universidade de Paderborn – Paderborn, Alemanha

Prof. Dr. Walter Lages, UFRGS
Doutor pelo Instituto Tecnológico de Aeronáutica – São Paulo, Brasil

Prof. Dr. Altamiro Susin, UFRGS
Doutor pela Institut National Polytechnique de Grenoble – Grenoble, França

Coordenador do PPGEE: _____
Prof. Dr. João Manoel Gomes da Silva Júnior

Porto Alegre, junho de 2012.

RESUMO

Este trabalho aborda os **sistemas de montagem** auto-organizados baseados em agentes para o ambiente industrial. Para isso, traz uma visão ampla dos paradigmas atuais para a manufatura que usam o conceito de agente e que promovem **auto-organização**, mas é focado no paradigma de sistemas de montagem e de produção evolutivos. Além da auto-organização, aspectos de **auto-otimização** nestes sistemas também são considerados.

Portanto, este trabalho aborda os aspectos teóricos e práticos de **sistemas evolutivos** e, em particular, propõe uma **plataforma multiagente** que usa o conceito de **agente mecatrônico** para permitir o desenvolvimento de sistemas auto-organizados e que também possuam a capacidade de otimizar autonomamente algum recurso interno ao sistema.

Um agente mecatrônico é uma entidade capaz de ação autônoma num sistema de manufatura, devido a uma decisão própria ou à solicitação de ação por outros agentes, e tal é conseguido através da definição e execução de funcionalidades pelos agentes mecatrônicos.

A plataforma multiagente proposta foi implementada e validada no âmbito de um projeto da União Europeia o qual possui parceiros acadêmicos e industriais e visa a criação de sistemas evolutivos auto-organizados em um cenário industrial real.

Palavras-chave: Engenharia Elétrica, Automação e Controle, Sistemas Multiagentes, Arquiteturas, Auto-organização, Manufatura.

ABSTRACT

This work deals with agent-based self-organized **assembly systems** for manufac–turing. Although the text aims to provide a comprehensive overview on current paradigms to manufactiuring systems development and execution that use the concept of agent and promote self-organization, the thesis main focus lies in evolvable production and assembly systems paradigms. Beyond self-organization, **self-optimization** aspects of evolvable systems are also considered.

Therefore, this thesis discusses theoretical and practical aspects of **evolvable systems** and, in particular, develops a **multi-agent platform** that uses the **mechatronic agent** concept to enable the development of self-organized systems and has the capacity of autonomously to optimize some internal resource of the system.

A mechatronic agent is an entity that is capable of autonomous action on a manufacturing system due to this own reasoning or by reacting to a requested action from other agents.

The proposed platform was implemented and validated within the scope of a research project funded by the European Union that has both academic and industrial partners and it goals the building of self-organized evolvable assembly systems for a real industrial scenario.

Keywords: Manufacturing Systems, Automation and Control, Multi-agents Systems, Self-organization.

DEDICATÓRIA

Ao amados ***Helena*** e ***Gustavo***.

AGRADECIMENTOS

À Deus pelo dom da vida e a oportunidade de estar aqui a construir novos ou refazer caminhos.

Aos amigos de Portugal pelo apoio, ideias e por terem me ensinado muito. Um muito especial agradecimento ao Luis Domingos Ferreira Ribeiro e ao Rogério Paulo Guerreiro Rosa.

Aos amigos da UFRGS que me receberam de braços abertos. Um muito especial agradecimento ao Edison Freitas, ao Ivan Müller e ao Rodrigo Allgayer.

À Uninova, pela oportunidade da participação no Projeto IDEAS e da experiência internacional.

Ao Programa de Pós-Graduação em Engenharia Elétrica, PPGEE, pela oportunidade de realização de trabalhos em minha área de pesquisa.

Ao Programa RH-DOUTORADO da Fundação de Amparo à Pesquisa do Estado do Amazonas (FAPEAM) pela provisão da bolsa de doutorado.

Um agradecimento especial ao orientador em Portugal, Prof. Dr. José António Barata de Oliveira que foi além da orientação acadêmica e tornou-se um amigo.

Ao também agora amigo Prof. Dr. Carlos Eduardo Pereira, que me abriu um caminho novo de pesquisa e desenvolvimento.

À minha família pelo amor e apoio, muitas vezes sacrificando a convivência.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACL	Agent Communication Language specification - especificação da Linguagem de Comunicação de Agentes.
ACO	Ant Colony Optimization - Otimização por Colônia de Formigas.
ACPI	Advanced Configuration and Power Interface - Configuração Avançada e Interface de Energia.
AGV	Automated Guided Vehicle - Veículo Guiado Autonomamente.
AID	Agent IDentifier - Identificador de Agente.
AMS	Agent Managment System - Sistema de Gerência de Agentes.
BMS	Bionic Manufacturing Systems - Sistema de Manufatura Biônica.
CAD	Computer-Aided Design - Desenho Auxiliado pelo Computador.
CoBASA	Coalition Based Approach for Shop Floor - Abordagem Baseada em Coalizão para o Chão-de-fábrica.
CNC	Computer Numeric Control - Comando Numérico Computadorizado.
CPU	Central Processing Unit - Unidade Central de Processamento.
DCS	Distributed Control System - Sistema de Controle Distribuído.
DF	Directory Facilitator - Facilitador do Diretório.
EAS	Evolvable Assembly Systems - Sistemas de Montagem Evolutivos.
EPS	Evolvable Production System - Sistemas de Produção Evolutivos.
E/S	Entrada/Saída.
EU	European Union - União Europeia.
FCT	Faculdade de Ciência e Tecnologia.
FIPA	Foundation for Inteligent Physical Agents - Fundação para Agentes Físicos Inteligentes.
FMC	Flexible Manufacturing Cell - Célula de Manufatura Flexível.
FMS	Flexible Manufacturing System - Sistemas de Manufatura Flexíveis.
FSM	Finite State Machine - Máquina de Estados Finitos.
GA	Genetic Algorithm - Algoritmo Genético.

HMS	Holonic Manufacturing Systems - Sistemas Holônicos de Manufatura.
IADE	IDEAS Agent Development Environment - Ambiente de Desenvolvimento de Agentes do IDEAS.
IDEAS	Instantly Deployable Evolvable Assembly Systems Project - Projeto de Sistemas de Montagem Evolutivos e Distribuíveis Instantaneamente.
IHM	Interface Homem-Máquina.
IPC	Inter-Process Communication - Comunicação Interprocesso.
JADE	Java Agent DEvelopment framwork - *framework* de Desenvolvimento de Agentes em Java.
MAS	Multi-Agent Systems - Sistemas Multiagentes.
MID	Molded Interconnect Device - Dispositivos Moldados Interconectáveis.
NEIMS	NeuroEndocrine-Inspired Manufacturing System - Sistema de Manufatura Inspirado no Sistema Neuro-Endócrino humano.
OCM	Operator-Controller Module - Módulo Operador-Controlador.
PLC	Programable Logic Controller - Controlador Lógico Programável.
PROSA	Product-Resource-Order-Staff Architecture - Arquitetura Organizacional Produto-Recurso-Ordem.
PSO	Particle Swarm Optimization - Otimização por Partícula de Enxame.
RMS	Reconfigurable Manufacturing Systems - Sistemas de Manufatura Reconfiguráveis.
RTAI	Real-Time Application Interface - Interface de Aplicação de Tempo Real.
RTOS	Real-Time Operational System - Sistema Operacional de Tempo Real.
SA	Simulated Anneling - Têmpera Simulada.
SOA	Service Oriented Architecture - Arquiteturas Orientadas a Serviços.
UFRGS	Universidade Federal do Rio Grande do Sul.
Uninova	Instituo de Desenvolvimento de Novas Tecnologias.

1 INTRODUÇÃO

Em resposta à crescente necessidade de personalização de produtos, a chamada customização em massa, vários paradigmas para os sistemas de manufatura têm sido estabelecidos nos últimos anos. Dentre eles, destacam-se aqueles baseados na agregação de módulos inteligentes.

Este trabalho é focado na descrição de uma arquitetura multiagente, baseada no paradigma de sistemas evolutivos, que é capaz de auto-organização. Esta arquitetura é criada especialmente para o ambiente industrial e vem desenvolver o conceito de agente mecatrônico, possibilitando também que desenvolvedores das aplicações possam incluir processos de auto-otimização.

Este capítulo mostra uma pequena introdução sobre estes assuntos, os quais serão mais amplamente estudados nos capítulos seguintes. Este capitulo também apresenta a estrutura geral do trabalho.

1.1 Motivação

Uma cena de ficção científica da série *Star Trek: The Next Generation*: o Capitão Jean Luc Picard, da Nave Estelar Enterprise NCC-1701-D, fala para o computador, ao lado do assento do capitão: "*Earl Grey Tea. Hot*." Imediatamente, uma luz brilha por segundos no console e, quando some, no lugar surge uma xícara de chá fumegante e deliciosamente aromatizada (MEMORY ALPHA, 2011).

É possível que a ficção de hoje torne-se a realidade de amanhã, pois a humanidade sonha com o dia em que gerar "tudo" a partir da mera manipulação dos átomos individuais dos quais as coisas se formam.

De igual modo, também viria a ser o ápice da personalização dos produtos e dos meios de produção, levando cada qual a produzir o que necessitasse ou desejasse de acordo com seus gostos e maneiras, com uma completa fusão do homem com a tecnologia.

Tal personalização, contudo, já é sentida e/ou necessitada nos dias de hoje. Assim, quer-se comprar um carro com a "sua cara", beber um café de acordo com o seu gosto, ou mobilar a casa com a sua estética.

Apesar de primatas superiores já terem sido vistos a utilizar gravetos e pedras como ferramentas, o fato é que, somente com o ser humano, temos uma maior elaboração na construção e uso delas. De fato, são as ferramentas e a cultura que tem definido a humanidade na atualidade (KAKU, 2011).

Para alguns, o homem é mais que simplesmente *homo sapiens sapiens*, pois já alcançou a essência do *homo sapiens tecnologicus*, no qual a fusão de tecnologia e biologia se faz em um único ser (LONGO, 2010). E isto somente é possível por causa da personalização da tecnologia, o que afeta diretamente a manufatura.

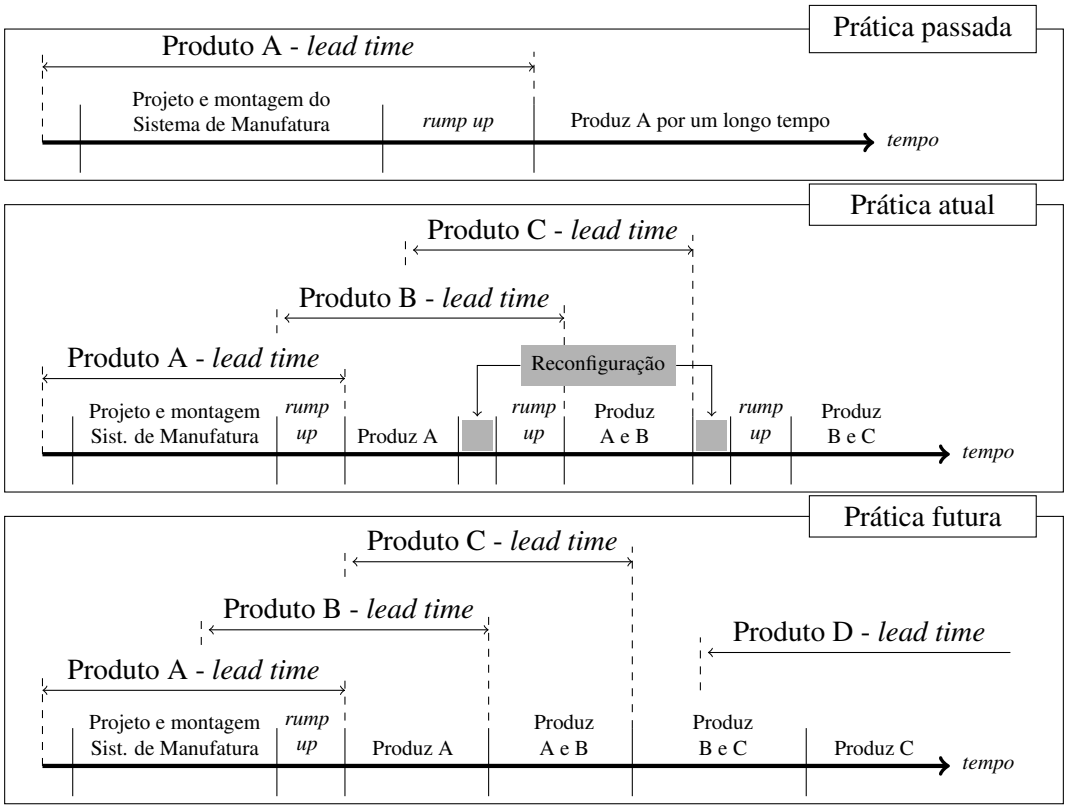

Figura 1: Evolução da Manufatura.

Portanto, além dos objetivos clássicos da manufatura de produzir maiores quantidades e melhor qualidade, a produção de produtos personalizados já é hoje uma tendência mundial (WESTKAEMPER, 2006).

O grande impacto da customização[1]no setor produtivo pode ser visto na Figura 1, que mostra a atual necessidade de uma rápida re-configuração das linhas de montagem. Enquanto o tempo de desenvolvimento de um produto foi reduzido graças ao uso de *Computer Aided Design* (CAD) e outras técnicas, a fase de produção, não. O desafio agora é, pois, de uma rápida (re)configuração para permitir a produção de uma grande gama de produtos customizados (KOREN et al., 1999). Para a prática futura, deseja-se que o tempo de reconfiguração seja próximo a zero, uma vez construído o sistema produtivo, este deve adequar-se à entrada de novos produtos, não necessariamente definidos em tempo de projeto do sistema de montagem.

Para tratar deste problema o conceito de agilidade no ambiente industrial foi concebido. Agilidade (YUSUF; SARHADI; GUNASEKARAN, 1999) é mais do que é fornecido pelo conceito de sistemas flexíveis (*Flexible Manufacturing System* - FMS).

A flexibilidade em um sistema produtivo está associada à sua capacidade de lidar com uma quantidade definida de diferentes produtos ou famílias de produtos, através da modificação autônoma de suas ferramentas, em tempo de processo (ELMARAGHY, 2005; MEKID; PRUSCHEK; HERNANDEZ, 2009).

[1]Neste texto, optou-se por utilizar os neologismos e jargões próprios da academia e indústria, a fim de se ter mais fluidez na leitura pelos interessados em manufatura, sistemas multiagentes e auto-organização. Assim, se utiliza o neologismo *customização* como sinônimo de *personalização*, os neologismos *componentizar*, *componentização* e afins, *plugar*, *plugabilidade* e afins, bem como manteve-se vários termos em seus originais em inglês, notadamente quando relacionados a um elemento específico de um paradigma da manufatura ou sistema.

A agilidade, por sua vez, está associada com a capacidade de a manufatura mudar rapidamente, e lidar com um subconjunto qualquer de produtos, mesmo que anteriormente não definidos. É, pois, a capacidade de realizar uma modificação no sistema devido a uma mudança ambiental relevante, de maneira rápida e eficiente, o que vai além de estar preparado para receber uma mudança anteriormente planejada.

Como boa parte dos produtos atuais são produtos tecnológicos, pode-se dizer que sistemas ágeis são uma tecnologia de personalização que visa a personalização da tecnologia.

Dentre os atuais desafios que a customização impõe à manufatura, encontra-se a necessidade da criação de sistemas de montagem capazes de realizar a autoadaptação, a auto-organização, a auto-otimização, a autocorreção, a autoproteção e o autoconhecimento.

Para responder a estes desafios, diversos paradigmas para a manufatura tem sido propostas nos anos recentes, os quais utilizam os conceitos da manufatura ágil (SANCHEZ; NAGI, 2001). Esses paradigmas apresentam-se de forma promissora no meio acadêmico, porém, o objetivo atual vai além da pesquisa e alcança a sua utilização prática em sistemas industriais (MONOSTORI; VANCZA; KUMARA, 2006).

Dos paradigmas recentes, *Evolvable Production System* (EPS) (ONORI, 2002) traz para a manufatura o conceito de sistemas evolutivos e apresenta-se como capaz de inserir a auto-organização no ambiente industrial. Uma forma de auto-organização, particularmente importante, é a capacidade do sistema em adaptar-se à entrada e saída de elementos do próprio sistema, como parte de sua operação normal, o que é também chamado de plugar-e-produzir (*plug-and-produce*) (RIBEIRO et al., 2009).

Como, em anos recentes, sistemas multiagente (MAS) têm sido sistematicamente estudados e desenvolvidos visando-se diversas áreas de aplicação, dentre elas a manufatura (SHEN et al., 2006), sistemas ágeis de manufatura, em especial em seu aspecto tecnológico (GUNASEKARAN, 1999), estão aptos a usar o paradigma multiagente para alcançar seus objetivos. De fato, EPS faz uso intensivo do conceito de agentes.

Apesar de vários esforços estarem em andamento (RIBEIRO et al., 2011), não há ainda uma plataforma multiagente específica para o ambiente industrial, ou seja, uma plataforma que suporte o conceito de agente mecatrônico, isto é, aquele agente capaz de realizar a sinergia entre eletrônica, computação, comunicação e mecânica, elementos básicos da manufatura moderna. Isso porque agentes mecatrônicos devem realizar uma dupla função de saber como e quando agir, e ainda possuir mecanismos para as ações no ambiente físico.

Além disso, dentre as características desejadas para sistemas formados por agentes mecatrônicos, citam-se a auto-organização e a auto-otimização, as quais permitirão à plataforma de agentes mecatrônicos o desenvolvimento de sistemas que realizam o *plug-and-produce* e a otimização do uso dos recursos do sistema produtivo.

As arquiteturas de agentes deliberativos são a base do desenvolvimento de soluções para os sistemas que requerem raciocínio, tais como os sistemas que realizam a auto-otimização; por outro lado, arquiteturas de agentes reativos são especialmente importantes para sistemas que tenham respostas rápidas, como em um processo de auto-organização.

Por exemplo, a auto-organização pode ser obtida através de um sistema multiagente formado por agentes reativos, os quais respondem às requisições externas por seus serviços quando necessário. A auto-otimização pode ser conseguida em um sistema baseado em agentes deliberativos, que respondem às mudanças no meio através de análise e da cognição do meio e de seus objetivos.

Logo, há um certo conflito com as arquiteturas existentes. A solução para tais conflitos se encontra na construção de uma arquitetura híbrida para os agentes, onde há tanto o raciocínio quanto reação aos eventos que são externos ao agente.

1.2 Definição do problema, questão de pesquisa e hipótese de trabalho

1.2.1 Definição do problema

Um sistema de montagem é formado por: diversos módulos mecatrônicos (os recursos), que realizam algumas atividades sobre os produtos sendo montados (o processo); os insumos para a montagem dos produtos; um sistema de transporte para levar os produtos aos módulos; e os produtos propriamente ditos, sendo montados, geralmente colocados em suportes (paletes).

Módulos, produtos e elementos do sistema de transporte podem ser visualizados como agentes inteligentes, e o sistema de montagem é, então, um sistema multiagentes.

Em um sistema de montagem baseado em agentes, puramente reativo, mudanças externas geram eventos que podem disparar um processo de auto-organização, contudo, em tal arranjo a auto-otimização é de difícil implementação. Em um sistema de montagem baseado em agentes, puramente deliberativo, a auto-otimização é conseguida pelo raciocínio de qual a melhor ação a ser tomada em uma dada circunstância, porém, em tal arranjo, a auto-organização somente seria alcançada após muito tempo de processamento, quando todos os envolvidos tivessem convergido para o mesmo objetivo, o que pode ser impraticável.

1.2.2 Questão de pesquisa

Pode-se, então, expressar a questão de pesquisa motivada pelo problema apresentado como:

> Como conseguir verdadeiro *plug-and-produce* em um sistema multiagente auto-organizado, que possa, ao mesmo tempo, otimizar autonomamente o uso de alguns de seus recursos?

1.2.3 Hipótese de trabalho

A **hipótese** de trabalho será:

> Uma arquitetura híbrida para o sistema, o qual é formado por agentes em duas camadas, onde uma camada realiza as deliberações necessárias e uma outra camada, que reage rapidamente aos eventos de organização. Tal arquitetura tem a capacidade de prover tanto auto-otimização quanto resposta de auto-organização em um sistema multiagente.

Em especial este trabalho interessa-se em como um sistema auto-organizado, aplicado à manufatura, poderá auto-otimizar um recurso interno importante, que é o tempo de transporte dos itens a produzir entre os diversos módulos e subsistemas de montagem.

1.3 Objetivo

O presente trabalho visa propôr uma arquitetura para auto-organização, a qual consegue com sucesso o *plug-and-produce* e que suporte a auto-otimização de algum de seus parâmetros internos, em especial a escolha da melhor rota para os produtos sendo produzidos no sistema de montagem.

Como objetivo secundário tem-se a localização da proposta no contexto das pesquisas em manufatura e em agentes, através de uma discussão conceitual e teórica sobre vários aspectos do ambiente industrial, da auto-organização e da auto-otimização. Isto tudo visando a manufatura de produtos customizados, a motivação para este tipo de automação.

1.4 Abrangência e limites

Este trabalho foi desenvolvido no âmbito das atividades do Projeto IDEAS (*Instantly Deployable Evolvable Assembly System*), o qual é um projeto da União Europeia dentro de seu "Programa-Quadro Sete" (FP7). O núcleo deste projeto é a uma plataforma de desenvolvimento de sistemas multiagentes para a manufatura, baseada no conceito de agentes mecatrônicos, chamada de IADE (*IDEAS Agent Development Environment*).

Contudo, o presente trabalho traz uma abordagem um pouco mais ampla. A Figura 2 dá uma ideia da abrangência e dos limites desde trabalho no âmbito do projeto IDEAS e de seu núcleo, o IADE.

Dentre as diferenças, o conceito de auto-otimização não é preocupação primária do IADE, enquanto aqui é relevante; também a nomenclatura usada inicialmente neste trabalho vai sofrer uma leve modificação em relação à nomenclatura usada no IADE. Contudo boa parte da arquitetura aqui mostrada foi inspirada nos trabalhos realizados no projeto IDEAS, ou serviu de base para a definição do núcleo do IADE.

Além disso, esta proposta conta com uma implementação de uma aplicação diferente de um sistema de montagem evolutivo, com o fim de se verificar a abrangência da proposta, isto é, a possibilidade de desenvolver outros tipos de aplicações. A aplicação escolhida, no caso, foi um sistema de controle distribuído (DCS).

O IADE, além de possuir várias das contribuições do autor, contém várias ferramentas de desenvolvimento, distribuição e supervisão de agentes mecatrônicos em um sistema evolutivo. Tais ferramentas não fazem parte do escopo deste trabalho.

1.5 Divisão do trabalho

Para alcançar o seu objetivo, este trabalho está assim divido:

O Capítulo 1 é esta Introdução.

O Capítulo 2 apresenta as características do ambiente industrial e os conceitos básicos que serão usados por todo o texto, tais como a auto-organização e auto-otimização.

O Capítulo 3 mostra alguns paradigmas modernos para a manufatura, bem como alguns trabalhos recentes sobre o uso de agentes na manufatura. Além disso, um apanhado

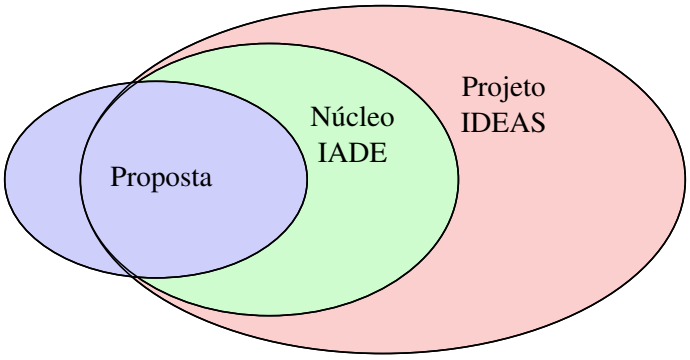

Figura 2: Abrangência e limites desta proposta em relação ao projeto IDEAS.

de soluções para o problema do agendamento e transporte, bem como os projetos europeus recentes de sistemas baseados em agentes também são visualizados.

O Capítulo 4 apresenta a proposta para um sistema auto-organizado, baseado em agentes e especifico para a manufatura, o qual também vai permitir auto-otimização.

O Capítulo 5 mostra a implementação da proposta e as especificidades do *IDEAS Agent Development Environment* (IADE), que é o *framework* do projeto *Instantly Deployable Evolvable Assembly System* (IDEAS), o qual nos permitirá validar algumas das ideias propostas nesta tese e implementadas no âmbito do projeto IDEAS.

O Capítulo 6 apresenta alguns experimentos realizados no âmbito do projeto IDEAS, em especial no seu pré-demonstrador no qual algumas das ideias propostas nesta tese foram validadas.

O Capítulo 7 trás, por fim, as conclusões sobre os temas levantados, nomeadamente a resposta de auto-organização, o laço de auto-otimização e como esses elementos se encaixam na arquitetura proposta. Também discute alguns trabalhos vislumbrados para o futuro.

1.6 Contribuição

Este trabalho apresenta uma proposta para uma arquitetura híbrida, baseada em agentes, que possibilita tanto deliberação quanto reação, especificamente desenvolvida para suportar o conceito de agente mecatrônico e, portanto, específica para a criação de aplicações na manufatura, em especial para aplicações que levam à customização em massa.

Sua maior contribuição está em agregar algumas ideias encontradas na literatura sobre agentes, sistemas evolutivos e sistemas auto-otimizáveis em uma proposta mais simples, a qual usa agentes mecatrônicos para alcançar auto-organização, mas também com a otimização de alguma métrica no sistema.

Como generalidade, este trabalho se classifica como uma aplicação da *inteligência artificial distribuída (Distributed Artificial Intelligence DAI, em inglês) DAi* (WHITBY, 2003; RUSSEL; NORVIG, 2004) à agilidade na manufatura (YUSUF; SARHADI; GUNASEKARAN, 1999). Como particularidade, ele restringe-se à implementação do paradigma multiagente em um *Evolvable Production System* (EPS) (ONORI, 2002) a fim de conseguir a auto-organização (DE WOLF; HOLVOET, 2005) e a auto-otimização (GAUSEMEIER; FRANK; STEFFEN, 2006).

2 CONCEITUAÇÃO TEÓRICA

Este capítulo aborda os conceitos encontrados na literatura que irão basear este texto, dentre eles os conceitos associados ao ambiente industrial, aos agentes inteligentes e sistemas multiagentes, bem como lista alguns dos paradigmas da manufatura que lidam com flexibilidade e agilidade.

2.1 Definições

Algumas palavras são tão corriqueiras que quase nunca se pensa sobre elas, todavia, até porque são muito comuns, as pessoas tendem a criar o seu próprio conceito a respeito delas. Então, é sempre bom expressá-las de forma clara e precisa para se evitar interpretações errôneas. Este é o objetivo desta seção.

2.1.1 Módulo

Um conceito simples mas que deve ser declarado para perfeito entendimento é o de módulo. Há várias definições de módulo na literatura, umas mais restritivas que outras. Por exemplo, em (IDEAS Project, 2010a) "um módulo é uma unidade auto-contida (mecânica) que possui uma função específica e a possibilidade de interagir com outros módulos" (IDEAS Project, 2010a). Exemplo: um módulo mecânico pneumático é mostrado na Figura 3.

Contudo, este conceito não leva em consideração a existência de outros tipos de módulos em um sistema mecatrônico típico, tais como os módulos de software ou eletrônicos. Com base nestas observações, pode-se expressar a definição de módulo como:

Definição 1: Um módulo é uma unidade autocontida que possui uma função específica, uma interface bem definida e a possibilidade de interagir com outros módulos.

Figura 3: Exemplo de módulo mecânico pneumático: cilindro linear para *clamping* da Festo. Retirado do catálogo de produtos *online* da Festo no endereço: `http://www.festo.com/cms/en-gb_gb/2497.htm`.

Nota-se que aqui não há distinção se o módulo é um software, um hardware eletrônico ou mecânico etc. Entretanto, o uso do termo módulo, sozinho, pode significar apenas o módulo mecânico (o que é mais comum no chão-de-fábrica), mas aqui estará associado com o conceito de módulo mecatrônico. Então, o termo módulo será usado como sinônimo de módulo mecatrônico, mas o contexto torna clara a distinção de qual tipo de módulo se está a falar. Portanto, para todos os outros tipos, um qualificativo não deixa dúvidas sobre o tipo de módulo a que se refere.

2.1.2 Módulo mecatrônico

Sistemas mecatrônicos são definidos como os sistemas que integram componentes mecânicos, eletrônicos, de comunicação e de computação de uma forma sinergética (GAUSEMEIER; KAHL; POOK, 2008).

Um sistema mecatrônico pode ser desenvolvido para ser modular, isto é, que tenha uma funcionalidade e interfaces bem definidas, tanto mecânicas, quanto eletrônicas ou de software, tornando o sistema mecatrônico um módulo mecatrônico. Para os objetivos deste trabalho, o conceito de módulo mecatrônico é:

Definição 2: Um módulo mecatrônico é um módulo formado por partes integradas mecânicas, eletrônicas, de comunicação e de software, com capacidade de percepção e atuação físicas.

2.1.3 Agente e agente racional ou inteligente

Um agente é definido como aquele que age. Em computação, um agente pode se apresentar como racional ou inteligente (RUSSEL; NORVIG, 2004), se há algum mecanismo intrínseco nele que define como deve agir. Para este trabalho, um agente é uma entidade (em geral de software que executa em um hardware apropriado) que pode perceber e/ou atuar em seu meio ambiente (através de sensores e atuadores), o que nos leva a seguinte definição de agente racional ou inteligente:

Definição 3: Um agente racional ou inteligente é aquele agente que é capaz de tomar uma decisão sobre como atuar, a partir do que o agente percebe do seu meio através de sensores.

Então um agente racional conta, por si mesmo, de meios para discernir como está o seu meio ambiente em um determinado momento, realizar uma avaliação precisa de como e quando atuar para alcançar o seu objetivo e também os meios de realizar a atuação no ambiente, modificando-o. Neste trabalho, deste ponto em diante, ao citar-se agentes subentende-se agentes racionais ou inteligentes, exceto se explicitamente indicado em contrário.

2.1.4 Implementação de agentes racionais

Do ponto de vista da implementação, há muitas semelhanças entre um agente e um objeto.

Um objeto é uma entidade de abstração de dados, que mantém dados dentro de um contexto. Objetos armazenam dados (o seu estado) e externam comportamentos, isto é, ações possíveis de serem realizados sobre os dados. Isso faz com que objetos tenham um significado no sistema.

Um agente é uma entidade similar, pois também é utilizado como uma abstração. Entretanto, além de manter dados contextualizados e conter comportamentos, um agente

é uma entidade autônoma. Em geral, um agente possui uma linha de execução temporal (*thread*) independente do restante do sistema.

Essa diferença entre objetos e agente é fundamental: métodos dos objetos (que formam o seu comportamento) são chamados externamente, enquanto os comportamentos dos agentes são chamados por ele próprio de uma forma reflexiva e pró-ativa. Da mesma forma, agentes diferem de objetos ativos, porque estes últimos apenas possuem *threads* para permitir comunicação assíncrona entre objetos. Contudo, devido às semelhanças, em geral agentes são implementados através de objetos que são executados em *threads* próprias, no entanto, estes executam os comportamentos do agente.

2.1.5 Arquiteturas de agentes racionais

Sistemas baseados em agentes podem apresentar duas abordagens no que tange à sua arquitetura: baseada em agentes reativos ou baseada em agentes deliberativos.

Agentes ditos deliberativos são aqueles que possuem um mecanismo de raciocínio em que as ações a serem tomadas são escolhidas por um mecanismo de inferência e/ou deliberação (WOOLDRIGE, 2002, caps. 3 e 4). Nos agentes reativos, o mecanismo de raciocínio está inscrito em seu código, isto é, é explícita, não necessitando de nenhuma inferência ou deliberação na escolha da ação a ser realizada (WOOLDRIGE, 2002, cap. 5).

A vantagem de agentes reativos é a sua rápida resposta, enquanto a desvantagem é que não têm meios de contextualizar a si mesmos e o seu ambiente. A vantagem dos deliberativos é sua flexibilidade, pois eles sempre irão contextualizar a si mesmo e o seu ambiente, enquanto sua desvantagem é o intenso consumo de processamento, com uma eventual demora na execução da ação.

2.1.6 Agente mecatrônico

Unindo-se as definições de agente inteligente e módulo mecatrônico, nasce, então, a definição de agente mecatrônico:

Definição 4: Um agente mecatrônico é um agente racional ou inteligente contido em um módulo mecatrônico, ou seja, é capaz de perceber, decidir quando e como agir, e atuar no mundo físico, autonomamente.

Para fins práticos, um agente mecatrônico será identificado com o módulo mecatrônico que o executa, pois há uma relação unívoca entre eles. Note que esta definição não exclui a possibilidade da existência de outros módulos mecatrônicos que não executem agentes mecatrônicos, mas todo agente mecatrônico estará associado a um módulo mecatrônico.

É estranho a existência de um módulo mecatrônico associado a mais de um agente racional. Mesmo se uma abordagem multiagente for utilizada no seu desenvolvimento, isto é, vários agentes racionais (software) sejam criados para controlar um módulo mecatrônico, do ponto de vista do sistema maior a que este módulo será conectado, este subsistema multiagente seria encarado apenas como um agente. Para que um sistema multiagente possa ter uma interface única, um agente de agregação, ou coalizão, se faz necessário. Tal arranjo insere uma estrutura hierárquica de agentes no sistema e, como tal, pode ser feito em diferentes níveis e complexidades. A existência de um agente que permite a coalizão de vários outros agentes é a chave para garantir a modularidade em diferentes níveis.

De fato, a modularidade do sistema exige essa unicidade, pois necessita de uma interface bem definida e, portanto, única. Apesar de o hardware de computador que executa o agente mecatrônico de algum módulo mecatrônico poder não estar fisicamente conectado a ele, entretanto, ainda assim faz parte do mesmo módulo. Se, por qualquer motivo houver a necessidade de mais de um agente mecatrônico por módulo mecatrônico, o projeto do sistema exige maior refinamento de suas partes, em geral a quebra daquele módulo, por ser complexo, em módulos menores.

Sistemas formados por um único agente não são mais que um sistema monolítico tradicional[1], que seja capaz de receber informações, tomar decisões e atuar em seu ambiente, isto é, modelam um tipo de sistema. Dessa forma, um sistema de agente simples, na manufactura carece de interação com outros tipos de sistemas e/ou agentes. Sistemas formados por vários agentes são mais interessantes, porque funcionam segundo o paradigma da divisão e conquista, mais ainda, permitem uma abstração maior do sistema e a gerência mais natural da sua complexidade. Entretanto sistemas multiagentes necessitam de algoritmos distribuídos e formas eficientes de comunicação de dados e decisões.

Inserir propriedades autonômicas em uma sistema multiagente é uma linha de pesquisa de vanguarda. Para aplicações na manufactura ainda há outro pormenor, por lidar com sistemas mecatrônicos, os quais possuem características próprias.

durante sua operação.

2.2 Comunicação entre agentes: protocolos FIPA

Tendo em mente que um sistema interoperável necessariamente deve ser aberto e padronizado, conforme alerta de HORN (2001), a *Foundation for Intelligent Physical Agents* (FIPA) definiu um conjunto de padrões para permitir a interoperabilidade entre agentes físicos, independente da linguagem utilizada para o seu desenvolvimento.

FIPA definiu vários protocolos e regras para comunicação entre agentes em vários níveis: aplicação, arquitetura, comunicação, gerenciamento de agentes e transporte de mensagens.

Dentre os conjuntos de especificações FIPA, algumas de interesse especial são as *Agent Communication Language specifications* (ACL), mostradas esquematicamente na Figura 4.

Para os objetivos do trabalho, serão analisados o *FIPA Contract Net Interaction protocol* e o *FIPA Request Interaction protocol*, que correspondem ao mecanismo de negociação e execução de funcionalidades na arquitetura, além, é claro da `ACLMessage`, a qual define as mensagens trocadas entre participantes de protocolos FIPA.

2.2.1 *FIPA Contract Net Interaction protocol*

A ideia por detrás do *FIPA Contract Net Interaction protocol* é permitir que vários agentes interajam de tal forma que um agente iniciador solicita um serviço a vários agentes que tem a capacidade de o executar e, através de negociação, decidir qual é aquele agente (ou agentes) mais apto(s) a realizar o serviço.

A Figura 5 mostra um diagrama de sequências com a dinâmica do protocolo. Há dois tipos de agentes na negociação: um iniciador e m participantes. O iniciador emite uma mensagem *Call for Proposal* (CFP) para todos os agentes participantes envolvidos na negociação. Os que puderem executar a funcionalidade contida na proposta chamada

[1] Exceto pelo seu tamanho, pois agentes costumam ser pequenos e tendem a realizar uma única ou poucas ações.

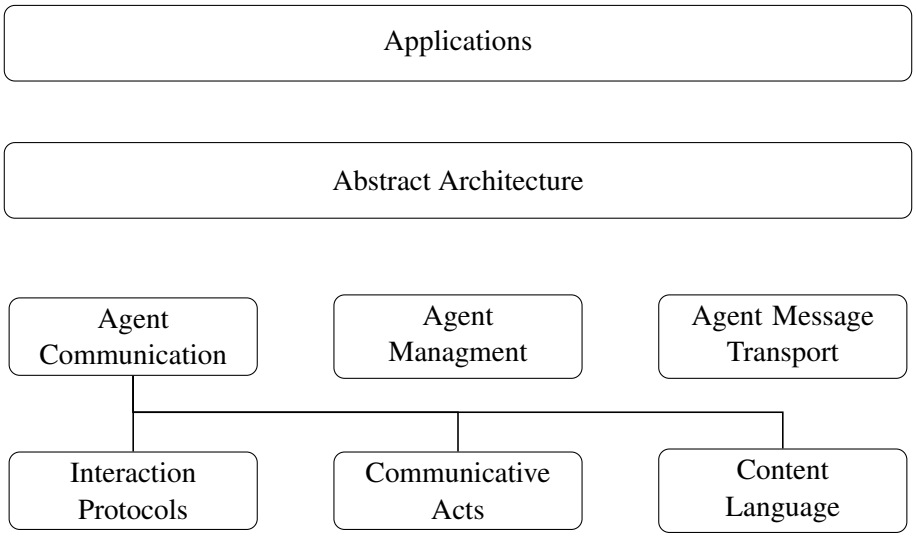

Figura 4: Especificações da FIPA ACL. Fonte: (FIPA, 2002a).

respondem com a mensagem *Propose* e os que não puderem, com a mensagem *Refuse*. É claro que alguns, eventualmente em falha, não conseguirão receber ou responder à chamada para propostas (então somente n participantes respondem) . Em geral, um agente que tem a capacidade de executar um serviço pode enviar *Refuse* por estar ocupado no momento em que recebe o CFP. Após um tempo de espera, o iniciador encerra a chamada de propostas e analisa as propostas recebidas, escolhendo através de algum critério definido pela aplicação qual ou quais agentes farão parte da execução. Para as propostas aceitas, envia uma mensagem *Accept Proposal* e para as rejeitadas, *Reject Proposal*. Então o iniciador fica no aguardo da ocorrência de duas mensagens possíveis: *Inform* que termina a execução da proposta quando positivamente executada, ou *Failure* quando da ocorrência de uma falha qualquer na execução do agente.

2.2.2 *FIPA Request Interaction protocol*

O protocolo *FIPA Request Interaction* é utilizado para solicitação de um serviço desde um iniciador a um participante; este é o jeito FIPA de replicar uma arquitetura do tipo cliente/servidor tradicional. Assim, uma mensagem é enviada por uma iniciador (o cliente) a um participante específico (o servidor) requisitando um serviço.

A requisição pode ser aceita ou não naquele momento. Dentre as diversas causas que um participante pode recursar uma solicitação é por estar ocupado. Se aceito, o participante informa o iniciador que irá realizar o serviço requisitado (mensagem *Agree*) e então o realiza. Há duas mensagens possíveis de retorno: se o serviço é realizado, é enviado de volta ao iniciador uma mensagem *Inform*, caso contrário, uma mensagem *Failure* é enviada. Essa sequência de operações pode ser visualizada na Figura 6.

Opcionalmente, o participante pode evitar o envio da mensagem *Agree*, executando diretamente a ação e somente informando o resultado. Isso diminui o tráfego de mensagens na rede.

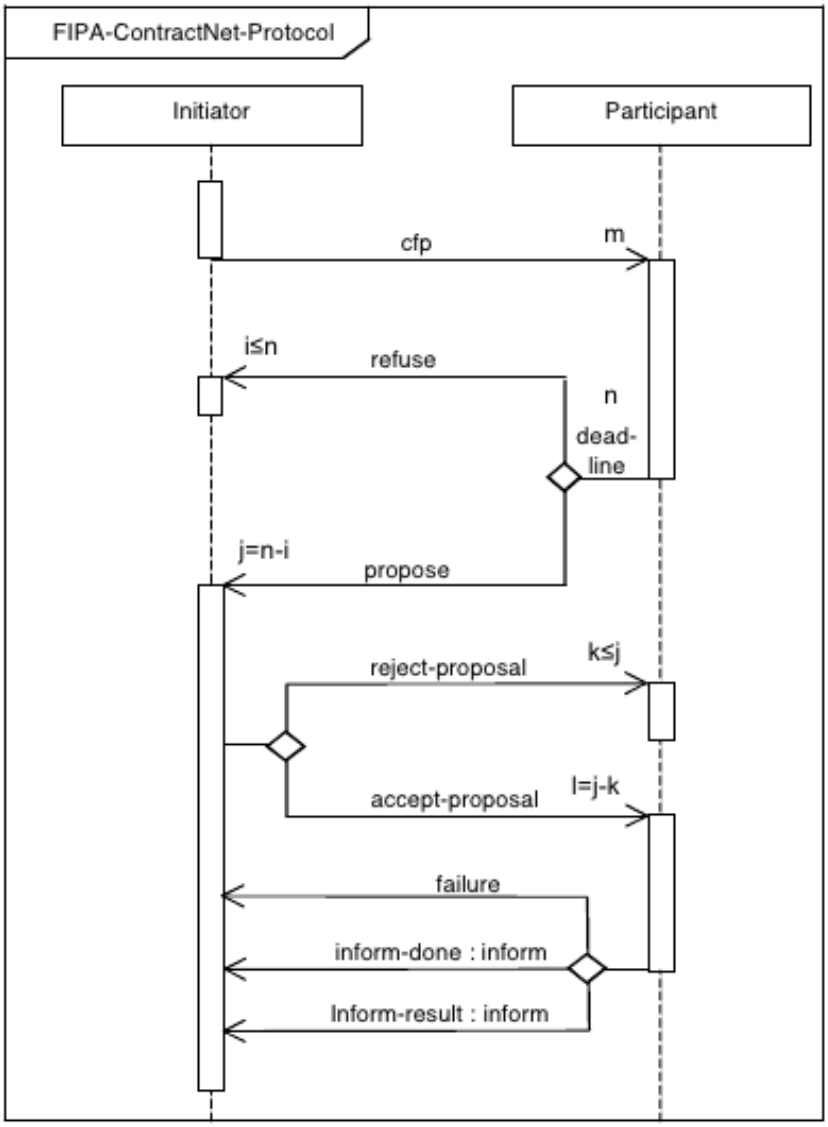

Figura 5: *FIPA Contract Net Interaction protocol*. Fonte: (FIPA, 2002b).

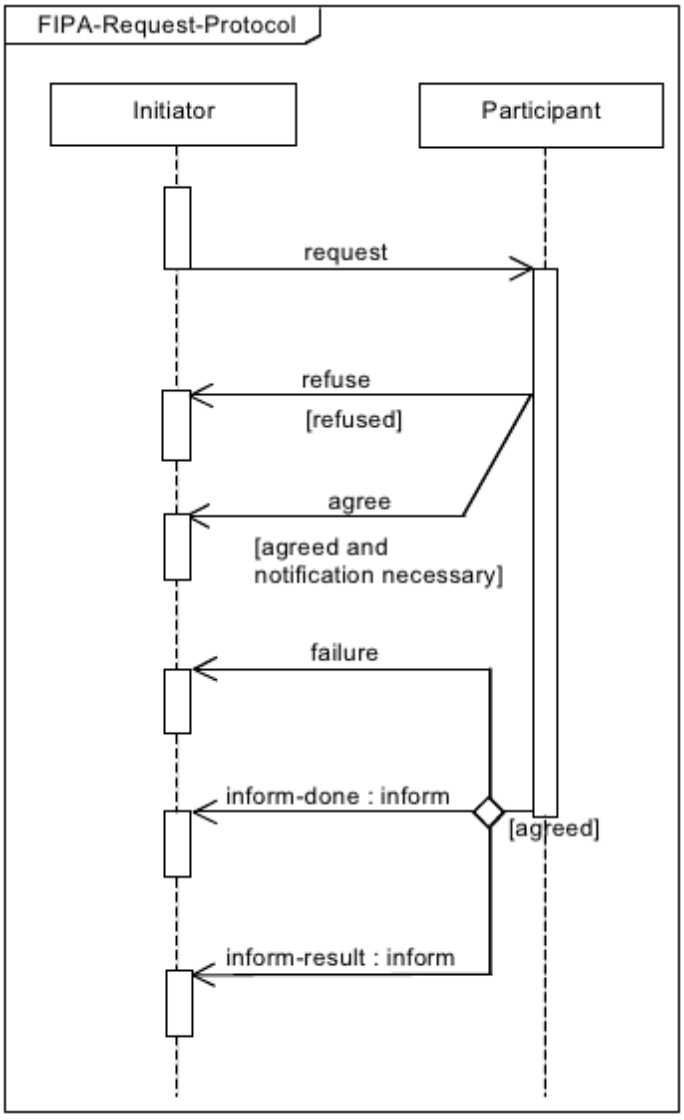

Figura 6: *FIPA Request Interaction protocol*. Fonte: (FIPA, 2002c).

2.2.3 Conteúdo das mensagens FIPA

Mensagens FIPA são formadas por diversos campos, os quais incluem:

- o destinatário, isto é, a identificação do agente no sistema, chamado de `AID`;
- o conteúdo, na forma de uma sequência de caracteres, os quais podem representar objetos serializados;
- campos relativos ao controle das mensagens, tais como a identificação da conversa, o tipo da mensagem, a temporização máxima, etc.;
- um campo especial, chamado Ontologia, que é opcional mas muito útil para dar escopo e/ou significado à mensagem dentro um certo contexto de trocas de mensagens;
- parâmetros da mensagem, as quais permitem à aplicação definir parâmetros do usuário para cada mensagem

O conteúdo das mensagens, associado à ontologia e propriedades, deixa à aplicação a capacidade de criar a sua própria estrutura de comunicação e protocolos de mais alto nível.

2.3 O ambiente industrial

A pesquisa no domínio da informática industrial, quando aplicada aos sistemas produtivos, tem tradicionalmente girado em torno de quatro grandes áreas ou dimensões (VER Figura 7): planejamento, agendamento, transporte e controle. Tratam-se de áreas desafiadoras que foram identificadas desde os primeiros esforços no desenvolvimento de máquinas flexíveis (JERALD et al., 2005).

Planejamento - Refere-se a divisão sucessiva da macro tarefa de produção, gerando-se as *partes* do sistema de montagem, as quais podem ser mais facilmente desenvolvidas. Em especial, a criação do fluxo de atividades (*workflow*) para a montagem de produtos é de natureza tal que exige trabalho específico de planejamento (VAQUERO; SILVA; BECK, 2011).

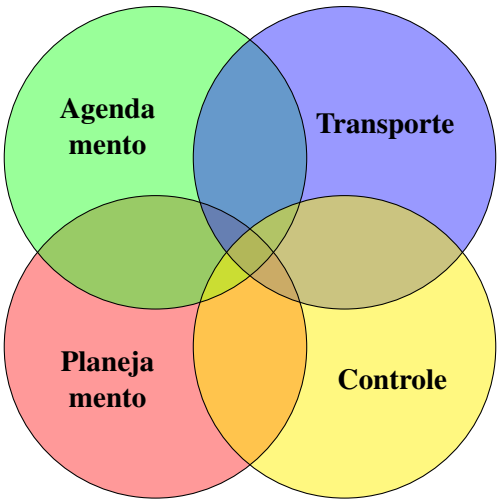

Figura 7: Principais dimensões da pesquisa em sistemas de produção.

Agendamento - É uma espécie de planejamento, mas em um nível de detalhamento maior, e se refere ao conjunto de ferramentas e técnicas que tentam aliar os recursos disponíveis na produção (e.g. tempo, máquinas e pessoas) com as necessidades da demanda (atividades a serem realizadas usando os recursos disponíveis) que, sempre flutuando, resultam em remanejamento ou re-agendamento incessante das atividades.

O controle - Refere-se às técnicas que mantêm o funcionamento de um sistema produtivo dentro de parâmetros estabelecidos, qualitativos e quantitativos. Ao se perceber desvios desses parâmetros são feitos ajustes, quase sempre automáticos, a fim de que o funcionamento do sistema não se altere. Para grandes desvios, entretanto, ajustes automáticos podem não ser suficientes e uma intervenção de um sistema externo, ou usuário, se faz necessária.

Transporte - Refere-se a parte do sistema de manufatura que realiza a movimentação física das partes de um produto entre os diversos elementos que compõem o sistema de montagem. Em geral, pode-se perceber três tipos gerais de sistemas de transporte:

AGV (*Automated Guided Vehicle*) qualquer entidade de transporte autônoma com a habilidade de realizar a livre movimentação (exceto por algumas restrições espaciais ou de segurança) dentro do sistema.

Esteiras (*Conveyor*) um elemento em uma rede que condiciona os caminhos possíveis que podem ser tomados no ambiente de produção.

Entroncamentos (*Handover*) um elemento que interliga outros elementos do sistema de transporte de vários tipos, em geral, *esteiras* com *esteiras*, *esteiras* com AGVs e vice-versa.

A Figura 8 mostra um exemplo, esquematicamente, de um sistema de produção típico, com sistemas de transporte, recursos de produção e produtos sendo processados. Em particular, este possui quatro esteiras, duas células de produção, cada qual com dois AGVs e dois recursos de produção, e diversos entroncamentos, que são os nodos na rede de esteiras ou *pick & place* da esteira para os AGVs e vice-versa. As setas na figura, representam os caminhos que os produtos seguem dentro do sistema. Tecnologias baseadas em trilhos também são possíveis, entretanto, sua modelagem computacional é idêntica ao das esteiras. Tal sistema pode ser estudado consoante a área que queira averiguar, se do ponto de vista do planejamento, agendamento, transporte ou controle.

Como pode ser percebido na Figura 7, contudo, há vários pontos de contato entre todas as quatro dimensões descritas, o que suscita questões de integração.

2.3.1 Integração e adaptabilidade

Os processos de manufatura discretos funcionam tradicionalmente como linhas de montagem. Podem ter as mais diversas formas: em linha, em garfo, *buffers* circulares, híbridas etc. e o termo "linha de montagem" é hoje genérico e o seu uso é histórico, devido ao fato da indústria de automóveis do início do séc. XX utilizar uma produção em massa baseada em linha.

Cada linha deve ser ajustada a um determinado produto e aos seus operadores, quando do início de seu funcionamento. Uma mesma linha de montagem pode servir para montar vários produtos diferentes, ou mais rotineiramente, vários modelos diferentes de um mesmo produto.

Do ponto de vista da linha e de seu ajuste (o seu *setup*), pouco difere se é um produto diferente ou um modelo diferente de um mesmo produto, exceto que o tempo gasto neste

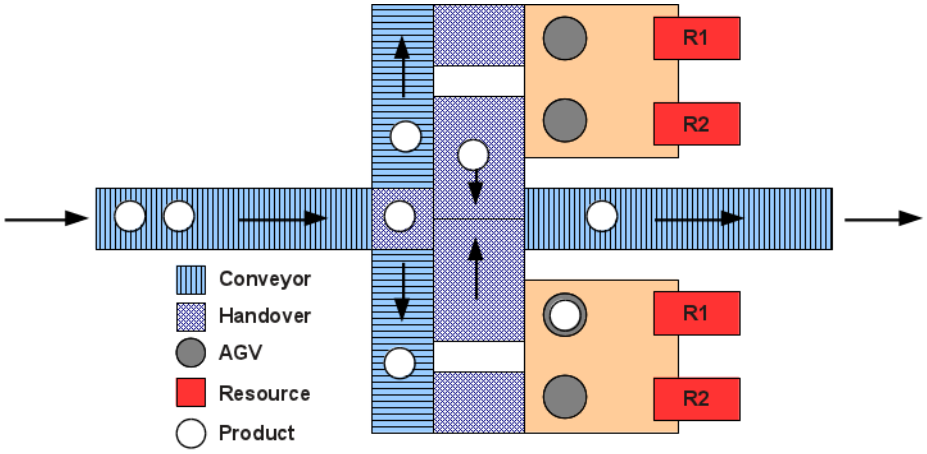

Figura 8: Exemplo de um sistema de produção.

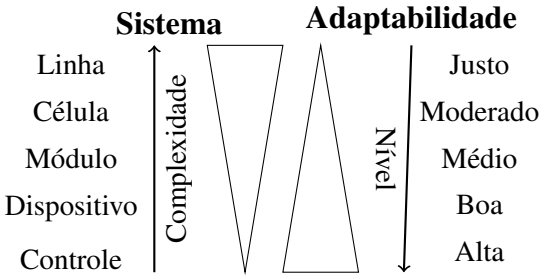

Figura 9: Requerimentos de adaptabilidade. Fonte: (IDEAS Project, 2010b).

último caso será menor, porque haverá um possível aproveitamento de ferramentas, bem como a minimização da curva de aprendizagem do processo produtivo, seja por operários humanos ou robóticos.

Portanto há uma necessidade de fluidez nas operações e que os softwares (e as máquinas) possam ser reutilizadas o mais rapidamente possível. Uma visão integrada se faz necessária, em todas as dimensões: planejamento, agendamento, controle e transporte, bem como há necessidade de adaptabilidade em todos os níveis: adaptabilidade nos dispositivos e ferramentas usadas em uma célula de trabalho, da célula propriamente falando, e até da linha como um todo.

Para se ter adaptabilidade em um nível, o nível imediatamente inferior deve também fornecer serviços de adaptabilidade. Então, quanto mais inferior o nível considerado, maior adaptabilidade seus componentes devam possuir, como pode ser visualizado, esquematicamente, na Figura 9.

A proposta aqui apresentada visa inserir adaptabilidade em múltiplos níveis da fábrica a fim de permitir integração e unir alguns dos pontos de contato mostrados na Figura 7, em especial, a execução do agendamento sob demanda, isto é, o agendamento é realizado durante a execução do processo produtivo, conforme é necessitado, com a consequente otimização do sistema de transporte.

Entretanto, para se conseguir essa integração, requer-se a coordenação das partes do sistema e, para tal, uma interligação para fins de cooperação e troca de informações se faz necessário.

2.3.2 Agilidade e integração na rede

Um aspecto particularmente importante em um processo produtivo é a capacidade dos sistemas que lhe formam de interagirem uns com os outros, através de uma rede de comunicação. Em um sistema ágil, a rede também tem de o ser.

Quando se pensa em agilidade em uma rede, tem-se em mente certas características, tais como, a capacidade da rede e de seus elementos de:

- inserção e remoção automática de nodos;
- configuração autônoma de um nodo;
- adaptação da rede e de seus nodos a mudanças ambientais significativas, tais como, congestionamento, mudança de objetivos e/ou prioridades, disponibilidade de banda, balanceamento de carga etc.;
- escalabilidade;
- disponibilização automática de seus serviços aos outros sistemas;
- descoberta e utilização automática dos serviços prestados por outros sistemas.

Para que a rede permita essas características é necessário a sua adaptação às mudanças ambientais, a otimização autônoma do uso dos recursos de rede, como banda, energia, rotas etc., bem como o gerenciamento autônomo dos nodos. Em outras palavras, as mesmas propriedades autonômicas visadas para o sistema de manufatura são igualmente desejadas para as redes industriais.

Um nodo em uma rede é um dispositivo que tem capacidade de comunicação e alguma capacidade de processamento e memória. Em verdade não se quer que o dispositivo gaste recursos com a comunicação; isso é um efeito colateral. O que se deseja é que ele execute algumas funções no sistema fabril, isto é, a sua aplicação, de forma remota e controlada, preferencialmente com o mínimo de custo associado com algo que não seja àquela aplicação.

Pode-se pensar que a agilidade de uma rede seja apenas a troca de protocolos em tempo de execução. A agilidade na rede, contudo, vai muito além da troca de protocolos que, por si, consiste em um processo de (re)configuração. Agilidade é adaptação mais organização. Adaptação na rede significa adequar a rede aos nodos e estes à rede, em geral pelo ajuste autônomo de alguns de seus parâmetros: largura de banda, energia, segurança, protocolos, etc. A organização implica em agrupamento de nodos de forma coerente, isto é, uma rede auto-organizável significa que a rede se adapta autonomamente para a realização de objetivos e políticas definidas em alto nível.

Para os objetivos deste trabalho, a comunicação entre agentes mecatrônicos, numa rede de módulos mecatrônicos, será, justamente, o critério para a criação dos agentes em camadas. A auto-organização dos agentes também irá impor uma auto-organização na rede.

Uma vez a rede montada é possível a interligação e integração de dispositivos em um célula de manufatura.

2.3.3 Agilidade e integração na célula

Um ambiente fabril conta com vários tipos de máquinas estáticas e móveis que são as ferramentas de produção e formam os sistemas de montagem. Boa parte das ferramentas de produção necessitam recolher informações da produção para os sistemas de níveis

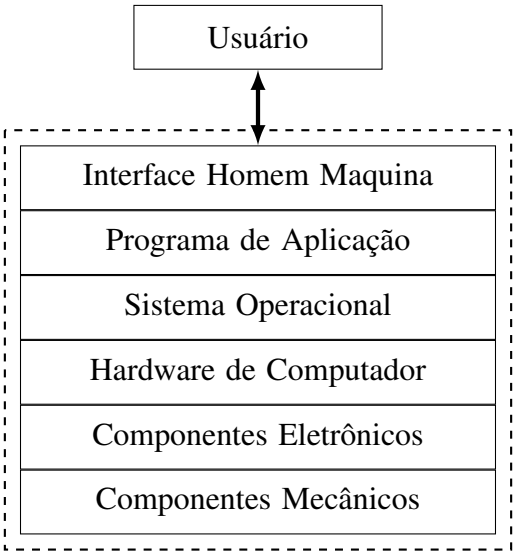

Figura 10: Componentes de um módulo mecatrônico.

mais altos na hierarquia de automação e necessitam agir de alguma forma no processo produtivo. Pode-se caracterizar tais ferramentas, que possuem partes mecânicas, eletrônicas, de comunicação e de computação, como sistemas mecatrônicos, então se apresentam conforme a Figura 10 mostra esquematicamente, formados por:

- Um substrato mecânico, aos quais são fixados sensores e atuadores, tais como sensores de temperatura, pressão, acionadores pneumáticos, *grippers* ou motores elétricos;
- Uma eletrônica analógica e digital, que realizam a interface de controle da parte mecânica;
- Um hardware de computador, com processamento e memória, onde são executados o sistema operacional e aplicações;
- Um sistema operacional, normalmente com características de tempo real, que executa as tarefas básicas do sistema;
- Um ou mais programas de aplicação (*tarefas*) que rodam sobre o sistema operacional em questão. De forma geral, uma tarefa especial fará a interface com o usuário (Interface Homem-Máquina - IHM);
- Em todas as camadas acima, elementos de comunicação abrangem as interfaces mecânicas, elétricas, da rede e com o ser humano.

GAUSEMEIER; KAHL; POOK (2008) classificam os sistemas mecatrônicos em duas categorias: a primeira baseada na integração espacial de componentes mecânicos e eletrônicos e a segunda lida com o controle de movimentos de um sistema multicorpo. O objetivo dos referidos autores é criar uma nova base de desenvolvimento de sistemas mecatrônicos.

Segundo aqueles autores, para se conseguir a integração espacial dos componentes que o formam, um sistema mecatrônico deve ser desenvolvido utilizando as mais recentes tecnologias mecânicas e eletrônicas, em especial deve fazer uso do conceito de MID

(*Molded Interconnect Devices*). Dispositivos MID permitem ser reutilizados e são concebidos para ser facilmente conectáveis a um sistema, através de interfaces bem definidas. Esses dispositivos expõem interfaces tanto mecânicas, quanto eletrônicas e de software. O foco da integração espacial está na montagem e integração de tecnologias.

Já os sistemas da segunda classe, que lidam com controle de movimentos, os dispositivos estão aptos a reagir a mudanças nas condições ambientais, identificar condições de operação críticas e otimizar suas atividades pela aplicação dos princípios de engenharia de controle. Entretanto, sistemas deste tipo devem lidar com mais de um tipo de controlador.

Então é possível inserir-se adaptabilidade ao nível do módulo mecatrônico e ao nível do sistema de produção, conforme a Figura 9

2.3.4 Manufatura e sistemas de tempo real

O termo sistema de tempo real é tão comum no chão-de-fábrica de uma indústria que, não raras vezes, passa-se despercebido o seu verdadeiro conceito:

Definição 5: Um sistema é dito de tempo real quando possui especificações temporais, isto é, além de produzir uma saída funcional, a amostragem da entrada, o processamento e a saída acontecem dentro do tempo especificado.

Este é o conceito encontrado em grande parte da literatura (WANG; WU, 1998; KOPETZ, 2011, cap. 1). Isto significa que um sistema de tempo real não está associado, *necessariamente*, à questão de desempenho. Um sistema não pode ser considerado de tempo real apenas porque exige que seus algoritmos executem rapidamente.

Como exemplo disso, em geral deseja-se que um algoritmo de otimização execute sempre o mais rapidamente possível, mas não há um tempo específico, uma vez que o tempo de execução é proporcional à quantidade de itens a serem otimizados. Outro exemplo emblemático é o acesso a banco de dados, o qual requer respostas rápidas, mas não há um tempo especificado para o atendimento das requisições.

Por outro lado, há inúmeros sistemas que podem não demandar o menor tempo, contudo, requerem *aquele* tempo. O exemplo típico são os sistemas de acompanhamento de variáveis contínuas.

Ao se especificar um tempo limite de espera por uma resposta a um algoritmo, ou a um banco de dados, como nos exemplos acima, pode ser que não se consiga a sua execução completa dentro daquele tempo, daí tem-se um problema de desempenho que é gerado pela especificação temporal. Neste caso o sistema, além de ser de tempo real, devido à especificação temporal, também deve executar rapidamente.

Classificam-se os sistemas de tempo real em dois tipos: tempo real rígido (*hard real-time*), ou tempo real brando (*soft real-time*) (KOPETZ, 2011, cap. 1). Diz-se que um sistema é de tempo real rígido se necessita que toda a sua execução, ou execuções, em caso de laços, sejam feitas dentro de um tempo especificado (p.ex. o tempo máximo de execução (*deadline*) especificado), mesmo na ocorrência do cenário de pior caso. Diz-se que um sistema é de tempo real brando se necessita que o tempo especificado (p.ex. o tempo de execução especificado) seja alcançado em média em suas execuções.

Na manufatura, de fato, há inúmeros exemplos de sistemas em que as especificações temporais levam a um problema de tempo real brando, como por exemplo o controle estatístico de processo (sistemas de testes), contagem da produção (sistema de controle de produção), sistemas de esteiras, etc. Ao contrário, não são tão numerosos os sistemas de tempo real rígido, tais como os sistemas de controle de movimento, os quais, em geral, requerem bom desempenho, devidos a tempos bem específicos. Outros sistemas de

acompanhamento de processos contínuos, tais como de reações químicas, temperaturas, etc., apensar de relativamente lentas, requerem um acompanhamento em tempos bem específicos, também são um exemplo de sistemas de tempo real rígido, porém podem não requerer grande desempenho.

Quando do desenvolvimento de sistemas de tempo real, um aspecto relevante a ser considerado é o da rede de comunicação[2], que pode apresentar-se como um limitador de desempenho e/ou inviabilizador do tempo real no sistema.

O problema do desempenho aparece devido à especificação temporal, que limita o tempo disponível para o tráfego das informações na rede, o que pode inviabilizar as tarefas nos nodos terminarem suas atividades no tempo especificado.

A rede também pode inviabilizar um sistema ser de tempo real quando não permite que os pacotes de dados que trafegam possam ter tratamentos diferenciados quanto ao tempo. Por exemplo, os quadros de imagens de um produto, em um sistema de testes, devem ser enviados ao operador (ou outro sistema de análise daquelas imagens) em tempos bem definidos e em uma ordem bem definida; na falha do envio desses quadros em tempos apropriados ou fora de ordem há perda na qualidade da imagem, o que pode ocasionar erros no teste. Dentre vários motivos das falhas nos envios, citam-se as esperas demasiadas nas filas dos roteadores da rede.

Outro aspecto importante no desenvolvimento de sistemas de manufatura é o tipo da plataforma computacional utilizada para a execução das tarefas de tempo real. Tais plataformas requerem hardware e um sistema operacional de tempo real (ALLGAYER; CAVALCANTE; PEREIRA, 2008). Por exemplo, devem possuir hardware e software que permita a troca de contexto de tarefas de menor prioridade para as de maior prioridade sempre que necessário.

Resumindo, somente com base em especificações temporais é que é possível ter-se um sistema de tempo real, caso contrário, pode-se apenas estar lidando com um sistema que simplesmente necessite executar mais rápido (performático), mas não necessariamente de tempo real. Contudo, no ambiente industrial, pode-se ter que lidar com ambos os problemas, tanto o tempo real quanto o desempenho.

2.3.5 Requisitos no ambiente industrial

Como mostrado, o ambiente fabril possui algumas características específicas que impactam nos requisitos para os sistemas de montagem que lhe são desenvolvidos. Em particular:

- Robustez: sistemas de montagem necessitam funcionar mesmo em presença de falhas, ou ter capacidade de suportar fadigas.
- Ambiente agressivo: sistemas de montagem necessitam operar em ambientes físicos agressivos, tais como, em presença de pó, descargas elétricas de variadas frequências, com limitação de aquecimento, tensão ou corrente, etc.
- Repetibilidade: sistemas de montagem devem ser capazes funcionar da mesma forma por um longo tempo.
- Integração: o sistema de montagem deve estar integrado no sistema produtivo, o que requer uma rede capaz de:

[2] Assume-se aqui que a rede de comunicação inclui toda a pilha de protocolos envolvidos na passagem de dados entre os nodos da rede.

  - Adaptação de nodos;
  - Organização de nodos;
  - Otimização dos recursos da rede;

- Interoperabilidade: capaz de operar com sistemas externos, que podem não ter sido descritos no momento do seu desenvolvimento, isto é, deve ser capaz de operar de maneira aberta.
- Compatibilidade: sistemas de montagem devem ser capazes de interoperar com outros sistemas que já existem na produção (sistemas legados).
- Adaptabilidade: o sistema de montagem deve ser capaz de alterar a si mesmo, ou ser alterado, a fim de de adaptar-se à mudanças ambientais.
- Auto-organização: o sistema de montagem deve organizar a si mesmo de forma a usar novas funcionalidades, fornecidas pelos módulos que lhe formam, assim que inseridos, ou deixar de usar tais funcionalidades, quando os respectivos módulos forem retirados.
- Auto-otimização: o sistema de montagem deve ser capaz de otimizar o uso de alguns de seus recursos, por exemplo, o tempo de transporte entre os módulos mecatrônicos.
- Especificação temporal: o sistema de montagem pode requerer a execução de suas tarefas dentro do tempo especificado, sempre (*hard real-time*) ou em média (*soft real-time*).

Como pode ser percebido, as chamadas propriedades ou características autonômicas, isto é, adaptação, auto-organização, auto-otimização, auto-proteção, etc. deveriam estar presentes num sistema de montagem, a fim de que os requisitos acima sejam alcançados.

2.4 Sistemas evolutivos, organização e auto-organização

Na literatura, os termos organização e auto-organização são bastante antigos, remontando à filosofia grega. Nos tempos modernos, quem primeiro formulou e desenvolveu os conceitos de organização e auto-organização com vistas ao uso em tecnologia foi Ashby em uma série de artigos na década de 60 (ASHBY, 2004). Mais recentemente, De Wolf e Holvoet, desenvolveram os conceitos de auto-organização e de emergência, a fim de diferenciá-los (DE WOLF; HOLVOET, 2005).

Para Ashby um **sistema** é uma entidade que toma um conjunto de entrada ($I = \{i_i\}$), realiza uma transformação nesta entrada, e gera um conjunto de saídas ($O = \{o_i\}$). Pode ser visualizado como uma máquina de estados que realiza uma transição entre seus estados internos ($S = \{s_i\}$), dada uma certa entrada, a fim de gerar uma determinada saída, então possui um mapeamento de entrada e estados nos próprios estados ($f : I \times S \rightarrow S$) e das entadas e estados na saída. A sua **estrutura** ou **organização** é a descrição dessa máquina de estados, em particular, do mapeamento entre os estados internos e as entradas para os próprios estados ($f : I \times S \rightarrow S$).

Se um sistema muda no tempo, isto é, modifica a sua organização, então houve uma mudança na máquina de estados que realiza a transformação da entrada na saída. Por exemplo, sejam duas estruturas $f : I \times S \rightarrow S$ e $g : I \times S \rightarrow S$ diferentes, o sistema

sai de f, num tempo t_0, para g, num tempo t_1. É claro que tal modificação implica na mudança do comportamento do sistema. Tais estruturas (f e g) são um subconjunto dos mapeamentos possíveis, isto é, é uma maneira de organizar o sistema de tal modo que toma valores na entrada e gera a saída correta, em cada momento.

Ashby foi o primeiro a perceber que, para modificar a estrutura (a organização) de um sistema base, uma entidade externa seria requerida, pois é impossível um sistema que está organizado em um mapeamento f possa se modificar para um mapeamento g por si mesmo, isto é, a organização de um sistema é invariante (SHALIZI, 2001). Entretanto, ao se criar uma estrutura $\alpha(t)$ a qual possua os valores de f em um tempo t_0 e os valores de g em um tempo t_1, então se pode ter uma relação entre o espaço de estados do sistema e um mapeamento que seja válido em um determinado tempo, como visto na Figura 11.

Seja o comportamento do sistema dado por:

$$\alpha(t) = \begin{cases} f(t) & t_0 \leq t < t_1 \\ g(t) & t_1 \leq t < t_2 \\ \cdots & \end{cases}$$

o sistema `Ext` é o responsável por decidir os valores de $t_0, t_1, t_2, \cdots$, tais que o comportamento do sistema $\alpha(t)$ será $f, g, \cdots$.

Se `Ext` fizer parte do sistema em estudo, isto é, for um subsistema, o sistema expandido (`Ext` + $\alpha(\cdot)$) é dito **auto-organizado**.

Exemplo: um sistema pode ser construído para incorporar 1 a n módulos, de tal forma que, a cada instante, o sistema pode ser formado por um subconjunto diferente dos módulos possíveis, isto é, apresenta uma dada organização. Se o agente que insere ou retira módulos for parte do sistema, ele é dito auto-organizado. Por outro lado, se o agente que insere ou retira módulos não for parte do sistema, ele não é caracterizado como auto-organizado, o que pode ser uma questão de convenção.

Ou seja, a auto-organização é uma característica que é definida pelo observador do sistema, então, diz-se que "a auto-organização está no olho do observador" (SHALIZI, 2001).

Para DeWolf e Holvoet, um sistema **auto-organizado** é aquele capaz de manter sua estrutura interna sem interferências externas, isto é, ele "é capaz de promover um aumento intrínseco da ordem de sua estrutura (espacial, temporal ou lógica) de forma autônoma" (DE WOLF; HOLVOET, 2005). Note que esta definição não exclui a entrada ou saída de informações, materiais ou energia do sistema, mas limita tais fluxos que não podem ser de controle. Como pode ser inferido, a auto-organização é uma propriedade de um sistema associada às suas fronteiras (ou seus limites) e, mais uma vez, são propriedades definidas pelo observador do sistema.

Sistemas que modificam sua estrutura (sua organização) no tempo, em resposta a mudanças ambientais, também são chamados de **adaptáveis**.

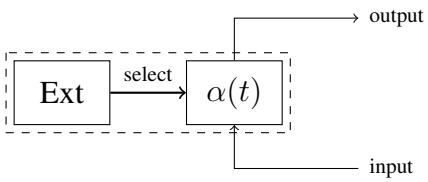

Figura 11: Um subsistema `Ext` selecionando qual estrutura em $\alpha(t)$ usar.

Uma propriedade especial desse tipo de sistema é a ***plugabilidade***. *Plugabilidade* é uma espécie de auto-organização, com ênfase na inserção e/ou remoção de módulos ao sistema (ONORI; BARATA; FREI, 2006). Um sistema é plugável se pode ser inserido ou removido, como um módulo, em um sistema maior, o qual ajusta a sua estrutura para o uso de tais módulos inseridos ou removidos.

Em particular, um dos principais aspectos de *Evolvable Assembly Systems/Evolvable Production Systems* (EAS/EPS) é justamente o de comportar a inserção e remoção de seus módulos constituintes como parte de suas operações normais.

Então, pode-se dizer que:

Definição 6: A auto-organização é um processo em que um sistema adquire ou modifica sua estrutura sem interferências externas de controle.

Note-se que este conceito não implica, necessariamente, na escolha do melhor conjunto de estados (melhor organização). Se isto acontece o sistema é dito **otimizado**, isto é, o sistema é capaz de buscar a melhor solução para a sua estrutura, dado um objetivo.

Por outro lado, um sistema é dito **adaptativo** quando o sistema modifica a sua estrutura interna para tentar manter o comportamento previamente definido, quando da presença de perturbações externas.

Por fim um sistema de montagem evolutivo pode ser definido como um sistema dinâmico e auto-organizado, que consegue modificar a sua estrutura devido a mudanças ambientais relevantes. Mudanças ambientais relevantes é um conceito definido em projeto. Em especial uma mudança relevante é a inserção, remoção ou reposição de módulos ao sistema, o que caracteriza a plugabilidade, a qual é parte do funcionamento normal do sistema. O que está conforme a definição para EAS/EPS de (ONORI; BARATA; FREI, 2006).

Uma análise de um *Flexible Manufacturing System* (FMS) com base nestes conceitos faz perceber que ele é um sistema adaptável, mas dificilmente seria auto-organizado. É adaptável porque é possível modificar a programação de uma CNC que o compõem para adequá-lo à entrada de um novo produto a ser produzido. Contudo, tal alteração é realizada por um agente externo, então ocorreu um fluxo de controle no sistema. Neste caso, dificilmente o programador do novo processo seria incluído "dentro" do FMS.

Por outro lado, um EPS, como um sistema evolutivo, é facilmente considerado auto-organizado, porque ele se adapta, autonomamente, seja à entrada de um novo produto a ser produzido, seja à entrada de novos módulos mecatrônicos no sistema. Tal auto-organização se apresenta porque novos processos produtivos para o sistema estão escritos nos produtos entrantes, enquanto os módulos mecatrônicos somente contém informações sobre sua habilidade ou funcionalidade na montagem. Portanto, não há um fluxo de controle externo ao sistema, no entanto um processo de organização se estabelece (seria preciso colocar o produto "fora" do sistema para não considerá-lo auto-organizado).

2.5 Auto-organização e Emergência

Dentre os diversos paradigma recentemente desenvolvidos sob a bandeira da agilidade, EAS/EPS clama por possuir emergência: "(...) isto é o que é chamado de propriedade emergente, e é a base de EAS. Sistemas tem propriedades emergentes que não são encontradas em suas partes. Você não pode predizer as propriedades do sistema completo por pegar o sistema em pedaços e analisar essas partes" (ONORI; BARATA; FREI, 2006).

Mas pode-se questionar: há, de fato, emergência em EPS? a auto-organização e a emergência em EPS são o mesmo fenômeno?

Para responder tais questões, o termo emergência merece um definição, que é o objetivo dessa seção. A primeira questão acima levantada, contudo, somente será respondida quando da análise dinâmica de um sistema EPS, o que é feito no Capítulo 5.

Na literatura o termo aparece associado a vários fenômenos físicos, biológicos e computacionais (SHALIZI, 2001). Por exemplo, no funcionamento de um formigueiro, uma comunidade de formigas apresenta um comportamento coletivo complexo com o surgimento de padrões identificáveis (o caminho de formigas), a partir de interações simples entre as partes que lhe formam. Sistemas biológicos deste tipo são a inspiração para diversos algoritmos, tal como *Ant Colony Optimization* (DORIGO; BIRATTARI; STUTZLE, 2006). Da mesma forma, a formação de sociedades artificiais, como a Internet, também mostra fenômenos de emergência (BARONE et al., 2003).

DE WOLF; HOLVOET (2005) fazem uma diferenciação de auto-otimização e emergência. Segundo eles "um sistema exibe emergência quando há emergentes coerentes no nível macro que dinamicamente são gerados a partir das interações entre as partes do sistema no nível micro. Tais emergentes são uma novidade em relação às partes individuais do sistema". Entretanto, dizer que emergentes não são capturados pelo comportamento de suas partes é um mal entendido. Novidade radical surge porque o comportamento global não é *facilmente* entendido a partir dos comportamentos das partes. O comportamento coletivo sempre está implícito no comportamento das partes, apenas que as propriedades emergentes não podem ser estudas a partir de um reducionismo, isto é, colocar fisicamente um sistema de lado, olhá-lo em suas partes, e descrever o seu comportamento global a partir do funcionamento das partes; mas o sistema pode ser estudado no contexto no qual se encontra.

Outras propriedades que um emergente possui são a coerência e a persistência. Coerência se refere a uma correlação consistente e lógica das partes. Emergentes parecem um todo integrado, portanto coerente. Também tendem a manter algum senso de identidade no tempo, isto é, são um padrão persistente.

Sistemas que possuem emergentes devem necessariamente ser dinâmicos, isto é, que podem modificar-se no tempo. Por exemplo, a temperatura em um êmbolo com um gás é função da pressão aplicada sobre o êmbolo; a temperatura é o emergente que surge da interação caótica das moléculas do gás no interior do êmbolo.

Um sistema que apresenta emergentes é naturalmente robusto, porque não há uma entidade qualquer no sistema que detenha toda a informação sobre o seu comportamento global, caso contrário o comportamento global não seria emergente. Os emergentes são relativamente insensíveis às perturbações ou erros. Um aumento no dano causado por perturbações e erros irá deteriorar a performance, mas a degradação é 'elegante'.

Como não há uma única entidade que defina o comportamento global, a entrada ou saída de elementos do sistema acontece espontaneamente e sem interferência de nenhum dos demais elementos que formam o sistema, tornando-o extremamente flexível.

Pode-se, então, definir emergência como:

Definição 7: "Um sistema exibe emergência quando há emergentes coerentes no nível macro que dinamicamente aparecem das interações das partes no nível micro. Tais emergentes são uma novidade radical com respeito às partes individuais do sistema", conforme (DE WOLF; HOLVOET, 2005).

Emergente é um termo geral para denotar o resultado do processo de emergência: propriedades, comportamento, estrutura, padrões etc.

Um sistema auto-organizado pode não apresentar emergência. Por exemplo, se um agente (controlador) contiver toda a informação necessária para ditar o comportamento de um sistema multiagente, pode haver auto-organização, mas não emergência, pois não há novidade no comportamento global, em relação às partes que formam o sistema, apesar de haver um acréscimo de ordem devido às informações contidas no agente controlador.

Por outro lado, pode-se ter emergência sem auto-organização. Por exemplo, em um sistema termodinâmico estacionário, da física, tal como em um pistão, onde um gás entra em equilíbrio com o peso do êmbolo, apresenta emergentes como temperatura e pressão do gás, a despeito da aleatoriedade com que os átomos se deslocam internamente ao gás (nenhuma auto-organização, portanto).

Há outras abordagens sobre emergência[3], mas aqui opta-se pela definição de DeWolf por crer a mais próxima aos objetivos visados.

[3] Ver uma coletânea de artigos somente sobre emergência e causalidade em ARSHINOV; FUCHS (2003)

3 TRABALHOS RELACIONADOS

Este capítulo apresenta um estudo de trabalhos recentes sobre o uso de agentes na manufatura, além dos trabalhos relativos às diversas dimensões de um sistema de manufatura, em especial, agendamento e transporte.

3.1 Abordagens ágeis para a manufatura

Para se conseguir o objetivo da customização em massa e sob a bandeira da agilidade diversos paradigmas tem sido propostos: *Bionic Manufacturing Systems* (BMS) por (UEDA, 1992) e outros; *Holonic Manufacturing Systems* (HMS) por (GOU; LUH; KYOYA, 1998; VAN BRUSSEL et al., 1998; BUSSMANN; MCFARLANE, 1999; BABICEANU; CHEN, 2006) e outros, *Reconfigurable Manufacturing Systems* (RMS) por (KOREN et al., 1999; MEHRABI; ULSOY; KOREN, 2000) e outros; *Evolvable Assembly System* (EAS) (ONORI, 2002; BARATA; SANTANA; ONORI, 2006) e *Evolvable Production System* EPS (ONORI; BARATA; FREI, 2006; RIBEIRO et al., 2009).

Todos esses paradigmas foram concebidos devido às limitações de *Flexible Manufacturing System* (FMS) (ELMARAGHY, 2005; FERGUSON et al., 2007) em conseguir a completa flexibilidade do sistema produtivo. Entretanto, graças a disseminação de máquinas CNC (VER subseção 3.1.1), as quais são a base de FMS, é que a indústria foi capaz de propor customizações e gerar a demanda atual pela customização em massa.

Esse paradigmas, que serão abordados de forma ampla nesta seção, tem alguns pontos em comum, notadamente o uso de dispositivos inteligentes, mas também possuem diferenças significativas, por exemplo, EPS apresenta o conceito de agentes, enquanto BMS de órgãos, e HMS de *holons*.

3.1.1 Flexible Manufacturing Systems

Os sistemas de manufatura evoluíram das fábricas artesanais com baixo volume e alta variabilidade de produtos (o que levava a uma alta variabilidade de máquinas de produção de propósitos gerais) para as linhas de produção com alto volume e baixa variabilidade de produtos (as quais necessitavam de poucas máquinas dedicadas) (ELMARAGHY, 2005). Com as linhas de produção deu-se origem, então, à produção em massa.

As máquinas para produção em massa seguem o conceito de manufatura “centrada no PLC”, o qual opera em ciclos:

(i) leitura de todas as entradas;

(ii) execução de uma lógica pré-definida sobre essas entradas (o processo) o qual gera as saídas (atuações);

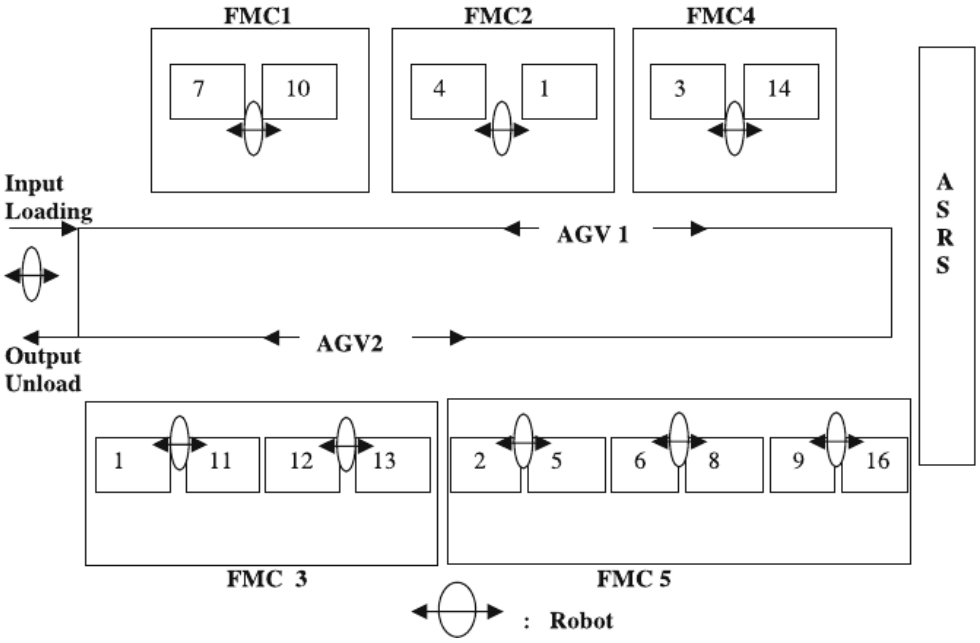

Figura 12: Exemplo de um FMS. Fonte: (JERALD et al., 2005).

(iii) ajuste das saídas conforme a lógica de processamento; volta ao passo (i).

Como esta sequência pode ser executada por uma máquina quantas vezes forem necessárias, e sempre da mesma forma, tem-se um ganho em qualidade, pela capacidade de repetição, e um incrível aumento da quantidade de itens produzidos, pela redução do tempo de cada ciclo e a não parada da produção. Mas não há nenhuma flexibilidade no processo. Se uma customização em um produto é requerida, um novo programa deve ser gerado e depois gravado no PLC.

É claro que tal processo pode ser programado em um PC industrial ou qualquer outro equipamento que possua processamento, bem como pode ser aplicado a vários níveis da empresa. Aqui, o conceito deste ciclo é referido como "centrado no PLC" apenas por motivos históricos.

Para permitir mais flexibilidade, uma abordagem chamada *Flexible Manufacturing System* (FMS) foi concebida. Em sistemas flexíveis, uma máquina flexível é construída de tal forma que suporte toda as variações do processo considerado.

Em geral, uma máquina flexível é formada por uma ou mais máquinas de comando numérico computadorizado (CNC) e algum sistema de transporte automatizado que permita aos produtos em produção serem movimentados entre estações que irão realizar um processo pré-definido. Se uma customização for necessária, basta que o produto seja movimentado para a estação adequada, responsável pelo processo específico no sistema que faz a customização.

A Figura 12, mostra um exemplo de um FMS composto por: cinco células flexíveis (*Flexible Manufacturing Cell* - FMC), cada um com duas a seis máquinas de comando numérico (CNC), um sistema automático de recuperação e armazenamento (AS/RS), dois AGVs idênticos, e robôs dedicados para carregamento e descarregamento de paletes nos AGVs, além de um sistema de carga e descarga para o FMS.

Entretanto, em um FMS, é um computador que gerencia e controla o processo como um todo, requerendo-se a programação de todas as etapas e customizações de forma antecipada. Se uma customização não previamente programada for requerida, uma repro-

gramação das máquinas é necessária, o que implica parada do processo produtivo e um tempo para reconfiguração, replanejamento, reagendamento, etc.

À medida que o tempo desde a concepção até a chegada ao mercado (*time-to-market*) dos produtos se torna cada vez menor, a tarefa de reprogramação de tais máquinas se torna cada vez mais desafiadora.

Além disso, o nível de produção de uma máquina flexível é bem menor quando comparado com as linhas de produção tradicionais, o que a segregou para uso na produção de produtos específicos e de baixa a média quantidades.

A grande consequência da aplicação de FMS, entretanto, foi o despertar para o processo de customização. Uma vez aparecendo no mercado alguns produtos feitos "sob encomenda", os consumidores passam a exigir a customização de todo o restante.

3.1.2 Sistemas de manufatura reconfiguráveis

O termo sistemas de manufatura reconfiguráveis (RMS) é usado desde a década de 90. Para KOREN et al. (1999), "um Sistema de Manufatura Reconfigurável (Reconfigurable Manufacturing System - RMS) é desenvolvido desde o início para rápidas mudanças em sua estrutura, bem como em seus componentes de hardware e software, de forma a ajustar a capacidade produtiva e funcionalidades, dentro de uma família de produtos, em resposta às súbitas mudanças no mercado ou em requerimentos regulatórios".

Um RMS tenta modificar o seu software e o seu hardware dinamicamente, a medida que detecta um evento que dispara o processo de reconfiguração. O objetivo final de um RMS é permitir reconfiguração simultânea de (KOREN et al., 1999): (i) todo o sistema, (ii) o hardware de máquina, e (iii) o software de controle.

Dessa forma, o termo tomou uma conotação genérica, e os paradigmas posteriores clamam a si mesmos de "reconfiguráveis", desde que usem soluções de hardware e/ou software reconfiguráveis. De fato, uma forma de reconfiguração, por exemplo, baseada em FPGAs[1] e *Reconfigurable* RTOS (GÖTZ, 2007) (sistemas de tempo real reconfiguráveis), foram gerados recentemente e tentam alcançar aqueles objetivos.

"De uma perspectiva *top-down*, sistemas de manufatura modernos são desafiados a incorporar capacidades de reconfigurabilidade, propriedades auto-x[2] e inteligência, de tal forma a ter sucesso no atual mercado competitivo global, no qual a variedade e complexidade dos produtos aumenta, enquanto o seu ciclo de vida diminui, requisitos de qualidade aumentam e margens de lucro diminuem. Felizmente, quando considerando-se uma perspectiva *bottom-up*, pode-se identificar os avanços consideráveis que tem sido feitos nos últimos anos em computação, comunicação e tecnologias da informação. Esses avanços tem permitido o desenvolvimento de arquiteturas de computação que permitem o desenvolvimento de máquinas inteligentes e reconfiguráveis baseadas em sistemas embarcados distribuídos de tempo real (*Distributed Real-time Embedded Systems* - DRES)" (PEREIRA; CARRO, 2006).

Pode-se dizer que RMS, como FMS, teve um sucesso relativo, mas ainda está em franca evolução. Todavia, todas as demais abordagens herdaram o termo reconfiguração e todos os seus sistemas são conhecidos coloquialmente como sistemas reconfiguráveis, apesar de terem diferenças técnicas fundamentais, segundo a abordagens: holônica, orgânica ou baseada em agentes.

[1] VER (ALTERA, 2009) e (XILINX, 2009)

[2] Auto-organização, auto-otimização, auto-proteção, etc.

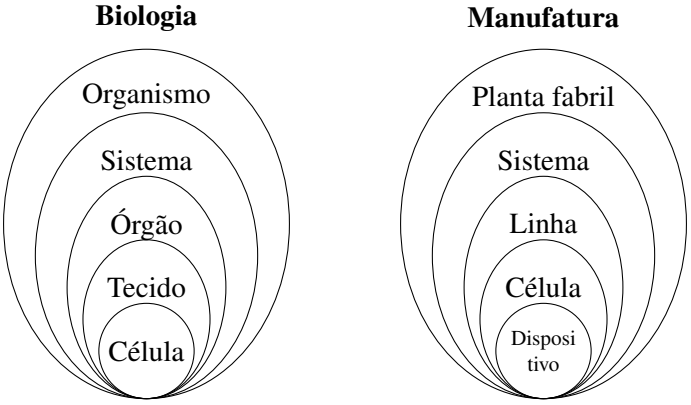

Figura 13: Hierarquia biológica e fabril.

3.1.3 Bionic Manufacturing Systems

Sistemas de manufatura inspirados nos sistemas biológicos são uma abordagem relativamente recente que tenta fazer um paralelo entre o funcionamento de um sistema biológico e de um sistema de manufatura.

O conceito foi concebido originalmente por (UEDA, 1992) que observou que os organismos podem se adaptar, auto-organizar, crescer e evoluir. Este conceito foi desenvolvido durante os anos 90, quando as características dos sistemas biológicos, tais como autonomia e comportamento espontâneo, eram copiados para os sistemas de manufatura, os quais eram desenvolvidos como uma combinação de vários sistemas autônomos, relacionados hierarquicamente, e formavam um sistema global com um função particular (THARUMARAJAH; WELLS; NEMES, 1998).

Ueda, em trabalho mais recente, mostra que a manufatura pode ser dividida em pequenos sistemas que realizam serviços autônomos, os quais interagem por meio da troca de informações de dois tipos: tipo-DNA (informações hereditárias, isto, o é histórico do sistema) e tipo-BN (processo de aprendizagem, isto é, as informações correntes). O fluxo de informação dos materiais (tipo-DNA) conjuntamente com o fluxo de informação do trabalho (tipo-DNA) entre entidades da produção (tipo-BN) (robôs, *grippers*, máquinas etc.) definem um produto. Fazendo-se ajustes na informação ou no fluxo, o sistema pode reagir a mudanças no ambiente. O processo de fazer esse ajuste é chamado de *problema de síntese* (UEDA, 2007).

A geração de sistemas hierárquicos é notável em BMS, com a sequência biológica:

$$\text{célula} \rightarrow \text{tecido} \rightarrow \text{órgão} \rightarrow \text{sistema} \rightarrow \text{organismo}$$

sendo mapeada para a manufatura como:

$$\text{dispositivo} \rightarrow \text{célula} \rightarrow \text{linha de produção} \rightarrow \text{sistema produtivo} \rightarrow \text{fábrica}$$

como pode ser visualizado na Figura 13. Por causa disso, o termo BMS passou também a representar *Biological Manufacturing Systems*.

Mais recentemente (GU et al., 2011) usam uma abordagem chamada de *NeuroEndocrine-Inspired Manufacturing System* (NEIMS), no qual os dispositivos são controlados por um mecanismo inspirado no mecanismo de regulação hormonal humano. Neste, o sistema nervoso central controla a produção de hormônios, os quais estimulam positivamente ou negativamente as funções celulares, através de mecanismos de realimentação (*feedbacks*) hormonais.

Da mesma forma, em um sistema fabril, cada célula de manufatura bio-inspirada recebe estímulos positivos e negativos que modificam o estado do seu controlador e afetam sua função. Entretanto, uma realimentação é emitida para o controlador de nível superior, o que influencia outras células aumentando um estímulo ou outro, de tal forma que o sistema todo se compensa até atingir o ponto ótimo. É uma abordagem híbrida, no sentido que usa tanto distribuição quanto hierarquia.

3.1.4 Holonic Manufacturing Systems

Uma outra abordagem chamada de Holonic Manufacturing System (HMS) foi desenvolvida por (VAN BRUSSEL et al., 1998), que criou a sua implementação de referência, mostrada na Figura 14, nos fins da década de 90. HMS é uma abordagem muito usada atualmente (VALCKENAERS; BRUSSEL, 2005; SHEN; WANG; HAO, 2006; SOUSA; RAMOS; NEVES, 2007).

Seu nome vem de *holon*, palavra grega que significa "o todo". Seus conceitos básicos podem ser elencados como se segue (BABICEANU; CHEN, 2006):

- *Holon*: um bloco de construção autônomo do sistema de manufatura, para transformação, transporte, armazenamento e/ou validação de informação e objetos físicos. Um *holon* consiste em uma parte de informação e uma parte de processo físico. Um *holon* pode ser parte de outro *holon*.
- Autonomia: a capacidade de uma entidade criar e controlar a execução de seus planos e estratégias
- Cooperação: um processo através do qual um conjunto de entidades desenvolve e executa planos de processos aceitos mutuamente.
- Holonarquia: Um conjunto de *holons* que podem cooperar entre si para conseguir alcançar um objetivo.

Dessa forma, um sistema holônico é uma holonarquia integrada de todas as atividades desenvolvidas na manufatura, desde a ordem de produção até o projeto, produção e marketing, visando uma empresa de manufatura ágil.

A implementação de referência (VAN BRUSSEL et al., 1998), chamada PROSA, posteriormente desenvolvida em (VALCKENAERS; BRUSSEL, 2005) discute três tipos de *holons*:

(i) *Holon* de ordem de produção (*Order holon*);

(ii) *Holon* de produto (*Product holon*); e

(iii) *Holon* de recurso (*Resource holon*).

Um HMS monta produtos a partir das interações entre os *holons*: uma ordem de produção emite ordens e dispara o processo de produção para os *holons* produto e recursos; o *holon* produto tem a informação do processo, o qual é passado para os *holons* recursos necessários. Estes *holons* e suas interações são mostradas na Figura 14.

Pode-se fazer um paralelo entre agentes e *holons*, pois ambos possuem autonomia, capacidade de cooperação. Agentes também permitem a formação de hierarquias, através de agrupamentos e coalizões.

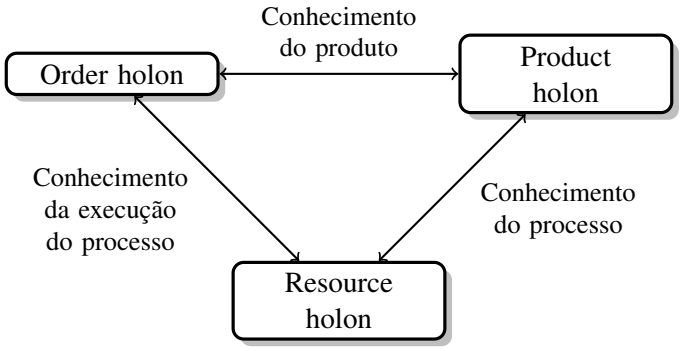

Figura 14: Holonic Manufacturing System (HMS) - Arquitetura de Referência. Fonte: (VAN BRUSSEL et al., 1998).

3.1.5 Evolvable Assembly Systems e Evolvable Production Systems

Os termos *Evolvable Assembly System* (EAS) e *Evolvable Production Systems* (EPS) foram originalmente usados por ONORI (2002); ALSTERMAN; ONORI (2005), em suas abordagens aos sistemas de montagem e ao sistemas produtivos, respectivamente. Contudo, a noção de evolução na manufatura é mais antiga (GAGLIARDI; RAJKUMAR; SHA, 1996; HSIEH, 2002), mas não estava ligada a agentes.

As definições de EAS/EPS são similares e baseadas na necessidade de realizar mudanças em um sistema sempre que o ambiente mude. "[EPS] é baseado em muitos elementos (módulos do sistema) simples, específicos à tarefa e reconfiguráveis, que permitem evolução contínua do sistema de montagem" (ONORI; BARATA; FREI, 2006). Já EAS é visto como "um sistema de montagem que pode co-evoluir conjuntamente com o produto e processo de montagem" (FREI et al., 2009). EPS pode ser associado com os mesmos objetivos de uma arquitetura orientada a serviço (*Service Oriented Architecture* - SOA), mas específica para a manufatura (RIBEIRO; BARATA; COLOMBO, 2008).

Resumindo a literatura, um EAS é um sistema de montagem formado de módulos interconectáveis, dinâmico, auto-organizado e evolutivo, capaz de modificar autonomamente a sua estrutura, conforme modificações ambientais relevantes do sistema de produção.

O conceito de EPS é similar: um EPS é um sistema de produção dinâmico, auto-organizado e evolutivo, capaz de suportar entre outras coisas: integração modular de componentes, mudanças nos produtos e processos, flutuação da demanda e adição, substituição e remoção de componentes durante a sua operação normal.

Como se pode observar, EAS e EPS são conceitos similares, exceto pelo nível da empresa que são considerados: EAS está focado no nível do dispositivo e EPS está focado no nível da fábrica. No nível da célula, dependendo do que se quer salientar, um mesmo sistema pode ser chamado tanto de EAS quanto de EPS. Portanto, a partir deste ponto, os termos serão usados como sinônimos, salvo quando explicitarmos a que nível fabril se refere, caso este em que será devidamente anotado no texto.

Um EPS é formado de uma multitude de subsistemas os quais são modulares, tanto do ponto de vista mecânico quanto funcional. Eles podem ser inseridos, substituídos ou removidos sem alterar essencialmente a funcionalidade do sistema. Cada um destes subsistemas ou elementos que formam o EPS é capaz de fornecer uma tarefa específica que será utilizada quando da necessidade.

O objetivo final para EAS/EPS é ter um sistema completamente *responsivo*, isto é, capaz de se adaptar à entrada de um produto (que faça parte da ampla gama de produtos

que o sistema suporte), não necessariamente definido conjuntamente com o sistema produtivo. EPS garante que não há tempo de reconfiguração, pois o sistema irá se adaptar à entrada em produção de qualquer tipo de produto que possa ser montado a partir das ferramentas de produção disponíveis no sistema. Mais do que isto, EPS deve ser capaz de responder também à entrada e saída de um novo módulo produtivo (um novo subsistema de montagem); EPS deve ser capaz de alcançar o *plug-and-produce* (plugar-e-produzir).

EPS possui vários pontos de contato com BMS, RMS e HMS, notadamente o uso da noção de modularidade de equipamentos no sistema e a existência de entidades inteligentes e autônomas. Sendo-lhes uma evolução, a diferença principal de EPS em relação às outras abordagens diz respeito ao uso sistemático do conceito de agentes, o que implica na existência de módulos inteligentes que podem ser definidos em diversos níveis de granularidade.

Em um EPS, a inteligência do processo está no produto sendo produzido, então o sistema é dito *centrado na palete*, o que é diferente dos paradigmas correntes, na qual os módulos de montagem detém o conhecimento do processo. Essa mudança de ponto de vista traz inúmeros benefícios, mas, ao mesmo tempo, torna a sua adoção mais difícil porque o pessoal envolvido na eventual implantação de um sistema evolutivo necessita mudar completamente o modo de pensar em como o sistema produtivo irá funcionar. Então uma mudança paradigmática é necessária: não mais se programa a máquina flexível para os modelos a serem produzidos, mas os produtos a serem produzidos são programados para usar os módulos de montagem disponíveis.

3.1.6 Resumo dos paradigmas

A Tabela 1 resume as discussões dessa seção, destacando os principais aspectos dos sistemas anteriormente analisados.

Tabela 1: Resumo dos paradigmas da manufatura

Parad.	**Baseado em**	**Característica principal**	**Referência**
FMS	Máquinas flexíveis	As variações no processo devem ser previstas	(ELMARAGHY, 2005)
RMS	Hardware e software reconfigurável	Reconfiguração simultânea do sistema, do hardware e do software	(KOREN et al., 1999) (MEHRABI; ULSOY; KOREN, 2000)
BMS	Células, órgãos, organismos e sistemas	Forte hierarquia	(UEDA, 1992) (UEDA, 2007)
HMS	Holons e holonarquia	Módulos autônomos e formação de hierarquia	(VAN BRUSSEL et al., 1998) (VALCKENAERS; BRUSSEL, 2005)
EPS	Agentes inteligentes e autônomos	Cooperação de agentes mecatrônicos	(ONORI, 2002) (BARATA; CAMARINHA-MATOS; ONORI, 2005)

Usando-se FMS como paradigma para a criação de sistemas de manufactura, deve haver uma previsão de quais os produtos a serem montados durante a criação do sistema produtivo, pois os processos devem ser programados anteriormente à entrada destes produtos em produção. Se necessário, pode haver uma parada na produção para a reprogramação das máquinas flexíveis. Para minimizar a necessidade dessas paradas, tanto o

hardware quanto o software das máquinas podem ser feitos de forma reconfigurável, de tal forma que o sistema pode modificar-se, até certo ponto, para atender a uma demanda não anteriormente prevista. Neste caso, está-se usando o paradigma RMS.

Por outro lado, sistemas baseados nas visões biológica ou holônica, já utilizam a noção de modularidade dos dispositivos e módulos mecatrônicos da produção e tentam unir tais módulos conforme a demanda. Por exemplo em um sistema holônico, o processo produtivo estará descrito em alguns poucos *holons* o que facilita e agiliza a reprogramação.

Com EPS, contudo, os módulos mecatrônicos que formam o sistema passam a ter autonomia completa através do conceito de agentes. Dessa forma, o próprio sistema se ajusta para usar os módulos necessários quando necessário. Este trabalho é baseado neste paradigma, contudo propõe uma arquitetura que permite além da auto-organização que o sistema possa realizar um processo de auto-otimização.

3.2 Arquiteturas multiagente para a manufatura

Arquiteturas multiagente para a manufatura tem sido propostas desde a década de 90. A proposta em (MATURANA; NORRIE, 1996) traz uma arquitetura chamada *Mediator Architecture*, a qual é usada para a construção de sistemas de suporte à tomada de decisão e para a coordenação de atividades de um sistema multiagente. Em particular, a coordenação envolve: (i) geração de sub-tarefas; (ii) criação de comunidades virtuais de agentes; e (iii) execução do processo imposto pelas tarefas.

A arquitetura é baseada em um hierarquia de agentes, criados a partir de um agente `Template Agent`, o qual centraliza toda a informação do processo, um `Data-Agent Manager`, que gerencia a formação de agrupamentos (*clusters*) de agentes, e `Active Mediator` que coordena a formação de agrupamentos de recursos. O agente `Template Agent` é o responsável pela criação, sob demanda, dos demais tipos de agentes, os quais irão criar agrupamentos, também sob demanda, segundo as atividades a serem realizadas e os recursos disponíveis. É uma solução baseada no conhecimento e funcionamento do `Template Agent`, portanto, centralizada, apesar de haver a introdução da formação distribuída de agrupamentos a partir dos outros agentes.

Em (PARUNAK; BAKER; CLARK, 1997, 2001), são mostrado os requisitos do projeto *Autonomous Agent Rock Island Arsenal* (AARIA) o qual lida basicamente com agendamento e controle. Agendamento e controle são as aplicações mais comuns, na manufatura, para um MAS (VER Seção 3.3).

Alguns aspectos importantes de integração da empresa foram tratados em (NAHM; ISHIKAWA, 2005), no qual propõe-se uma arquitetura formada de dois tipos de agentes: `Hibrid Behaviour Agent` (HBA) e `Hibrid Interaction Agent` (HIA). Os autores chamam sua abordagem de híbrida porque permite a execução de comportamentos tanto discretos quanto contínuos. O HIA é o responsável por adaptar a comunicação entre dois HBA de diferente tipo de execução e é, também, o responsável pela comunicação com outros HIAs. Assim, o sistema é formado por uma rede em que HIAs que se interconectam e cada um deles é um agregador dos HBAs.

Arquiteturas específicas para sistemas de montagem, que lidam com o conceito de agentes mecatrônicos, estão ainda em desenvolvimento. Entretanto, um *holon* pode ser modelado como um agente e uma holonarquia como um sistema multiagente, no qual é feita uma coalizão de agentes. Então, os sistemas holônicos podem ser considerados como arquiteturas multiagente para manufatura, como é o caso de *ADAptive holonic COntrol aRchitecture* (ADACOR) (LEITÃO; RESTIVO, 2006), o qual é um sistema holônico, mas

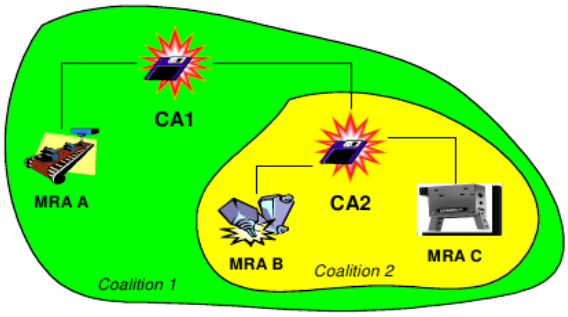

Figura 15: Hierarquia de coalizão no CoBASA. Fonte (BARATA; CAMARINHA-MATOS, 2003).

é modelado usando MAS. Similarmente a este trabalho, ADACOR também foi implementado usando o *Java Agent Development Framework* (JADE).

Para uma noção mais clara de agente mecatrônico, a proposta apresentada por (BARATA; CAMARINHA-MATOS; ONORI, 2005), chamada Coalition Based Approach for Shop Flor Agility (CoBASA), serve como um ponto de partida.

A arquitetura CoBASA propõe vários tipos de agentes e suas interações para o desenvolvimento de sistemas EPS:

MRA *Mechatronic Resource Agent*, que modela um recurso.

AMI *Agent Machine Interface*, que sempre está conectado a um módulo físico e é uma capa ou interface padrão para o acesso ao hardware pelo MRA

BA *Broker Agent*, responsável pela criação de coalizões.

CMgA *Cluster Manager Agent*, responsável pelo gerenciamento de agrupamentos de agentes.

CA *Coordination Agent*, coordena outros agentes, isto é, representa uma coalizão.

Nesta proposta, agentes são agregados por meio de coalizões, definidas por um agente de coordenação, conforme mostrado na Figura 15, onde pode-se ver a linha de montagem, representada na coalizão 1, formada de duas estações, uma de montagem e uma outra que é um módulo mecatrônico complexo, representado na coalizão 2 e formado, por sua vez, por duas estações: um dispensador e uma unidade de colagem.

No CoBASA também há uma definição de *skill*, associada com as habilidades e funcionalidades dos módulos do sistema de montagem. Então, um agente mecatrônico é a entidade que reconhece a requisição de execução e, efetivamente, executa *skills*, os quais realizam algumas atividades no sistema de montagem.

Tal arquitetura foi, posteriormente desenvolvida por (FREI et al., 2009), de forma a incluir a agregação de agentes por meio de ontologias.

Arquiteturas baseadas em agentes, mas que espelham arquiteturas orientadas a serviços também foram propostas. Em (SHEN et al., 2007) propõe-se um sistema formado de agentes mediadores, que realizam coalizão de recursos, mas baseados em um *Agent Web Service* (AWS). Todos os agentes na arquitetura são reativos, e implementam os protocolos para *Web Services*.

Tentativas de unir adaptação e otimização também já foram levadas a efeito, por exemplo, em (BARBOSA; LEITÃO; PEREIRA, 2011) mostra-se um sistema bio-inspirado, o

qual usa *Particle Swarm Optimization* (PSO) para realizar roteamento em um sistema de montagem.

Um revisão bibliográfica sobre o controle da manufatura, baseados em agentes, pode ser encontrado em (LEITÃO, 2009). Uma revisão sobre sistemas de controle em rede pode ser vista em (ZAMPIERI, 2008).

3.3 O problema do agendamento

Um sistema de montagem é formado por vários recursos (equipamentos e insumos) que são unidos para formar produtos, após a execução de um conjunto específico de passos de montagem (i.e. um processo). Como fazer para que os recursos e as partes disponíveis possam ser alocados para cada processo de produção dos produtos?

Basicamente, a escolha dos recursos a serem utilizados para executar determinada tarefa (ou conjunto de tarefas, i.e., um processo) é um problema típico de agendamento. O problema do agendamento na manufactura é frequentemente descrito como um problema chamado de *job shop*, o qual consiste na alocação de um conjunto de tarefas a um conjunto de máquinas no tempo, para um dado período, na tentativa de minimizar (ou maximizar) algum critério. Isto frequentemente implica que tanto as tarefas quanto os recursos, em um dado período de tempo, são conhecidos *a priori*, bem como seus custos (XIA; WU, 2005).

O problema foi demonstrado ser NP-completo[3], resultando em um total de $\{n!\}^m$ agendamentos potenciais para um sistema com n tarefas e m maquinas (GAREY; JOHNSON; SETHI, 1976; LUH, 1998). Agendamento é um problema que tem sido amplamente investigado, de tal forma que alguns estudos do estado da arte e descrição de abordagens diversas para o problema podem ser encontrados na literatura (SHEN, 2002; SHEN; WANG; HAO, 2006). Também digno de nota são os trabalhos de revisão bibliográfica sobre o agendamento e sobre o roteamento em sistemas com AGVs (QIU et al., 2002). Aqui é feito um esboço de revisão bibliográfica do tema.

Há entretanto, algumas premissas no problema do *job shop* que não se enquadram totalmente nos requisitos dos sistemas de montagem ágeis, abordados neste trabalho, em particular (LUH, 1998):

- Rotas alternativas - abordagens modernas devem sustentar sistemas altamente dinâmicos onde os itens podem tomar diferentes rotas para a mesma máquina e podem, por eles mesmos, explorar potenciais diferentes lugares durante a produção. Exemplo: pode ser mais vantajoso a uma palete dar uma volta extra em uma linha circular do que aguardar em uma fila.

- Limitação de *buffer* - geralmente as soluções para o problema de *job shop* assumem que os *buffers* são ilimitados, então, formação de bloqueios, gerenciamento pobre do sistema de transporte e outros pontos são deixados de lado para a geração do agendamento ideal, que não raro não se aplicará a um sistema real.

- Capacidade de transporte - devido ao fato que há especificidades neste domínio, o tempo de transporte é raramento considerado, quando elaborando-se um agendamento. Um sistema de transporte é assumido ideal e que as máquinas estarão prontas para servir aos seus objetivos em um dado *slot* de tempo definido pelo agendamento.

[3]Classes de complexidade computacional podem ser encontradas em (PAPADIMITRIOU, 2003).

- Tempo e determinismo - algoritmos para agendamento são, em princípio, determinísticos, o que sugere que possam ser usados amplamente numa aplicação de tempo real. Entretanto, todos eles executam em tempo exponencial, de forma que o desempenho passa a ser um problema. Para evitar tal problema, em geral, o agendamento é feito *em tempo de projeto*. Entretanto, variações no sistema produtivo levam a necessidade de re-computação de um dado agendamento, o que pode requerer muito tempo para ser feito e inviabilizar o seu uso *em tempo de execução*.

Em um sistema de máquinas flexíveis, onde o problema do agendamento é mais importante, há várias propostas para a sua solução. Em particular (LUH, 1998) propõe um método chamado relaxamento de Lagrange para resolver o agendamento em um FMS de propósito geral (FMS não-dedicado). O objetivo aqui é minimizar o esforço computacional para resolver um agendamento de *job shop* pela introdução de dinâmicas de mercado, onde um custo flutuante é associado à alocação de cada máquina em um dado tempo. As equações são simplificadas pelo método de relaxamento de Lagrange, o que torna o agendamento mais factível, por não consumir tanto tempo de processamento, contudo a solução é apenas *quasi*-ótima.

Em (SINRIECH; KOTLARSKI, 2002) o custo de transporte é explicitamente considerado no agendamento simulado de um FMS com 20 estações de trabalho organizadas em uma linha circular, servida de vários carros transportadores para a retirada do produto da linha e o seu transporte até as estações. A conclusão de seu trabalho é que a capacidade do sistema de transporte impacta significativamente na performance do sistema como um todo, e a melhor performance é conseguida quando se aumenta o número de carros transportadores.

Em (JERALD et al., 2005) um análise comparativa é feita entre algumas técnicas de otimização, para a resolução do problema do agendamento. As técnicas comparadas foram *Genetic Algorithm* (GA), *Simulated Anneling* (SA), *Mimetic Algorithm* (MA) e *Particle Swam Algorithm* (PSA). Esses métodos foram aplicados em um FMS composto de cinco células flexíveis, várias ferramentas e um sistema de transporte baseado em AGVs, os quais são assumidos sempre disponíveis. Sob essas circunstâncias, o algoritmo PSA apresentou melhor performance. A dimensão e natureza do sistema, contudo, não permite que esta conclusão possa ser generalizada para sistemas maiores.

Outros métodos incluem abordagens híbridas. Nestas, um problema pode ser dividido em sub-problemas que são resolvidos e então os resultado parciais são integrados em um resultado final. Em (XIA; WU, 2005) é usado PSA para associar operações às máquinas, que é a geração da população potencial, e SA para o agendamento das operações. Em (KACEM; HAMMADI; BORNE, 2002) uma busca em profundidade é feita para pré-alocar tarefas em cada máquina, reduzindo o espaço de otimização para o agendamento no tempo.

Agendamento não é um problema somente para FMS, mas também para HMS. A diferença entre estes dois paradigmas é que HMS usa uma abordagem distribuída para resolver o problema do agendamento. Em (BABICEANU; CHEN; STURGES, 2005), a solução proposta é usar interações entre os *holons* que compõem o sistema. Assim, em sua arquitetura, *holons* de manipulação de materiais geram agendamentos preliminares que são então avaliados por um *holon* de ordem de produção, o qual seleciona a melhor solução baseado em uma métrica. Uma abordagem similar é apresentada por (VRBA; HRDONKA, 2002), que descreve as interações através do uso dos protocolos da *Foundation for Inteligent Physical Agents* (FIPA). Também sobre HMS, a definição de agen-

damento no contexto do PROSA, pode ser encontrado em (VAN BRUSSEL et al., 1998; SOUSA; RAMOS; NEVES, 2007).

Sistemas holônicos são tipicamente implementados usando uma abordagem baseada em agentes. Em (CICIRELLO, 2001) o problema do agendamento foi formalizado com um ponto de vista da Teoria dos Jogos, onde agentes entram em jogos de negociação e tentam maximizar uma métrica, por exemplo o rendimento máximo das tarefas, que lhes serve de parâmetro para decisão local. Um abordagens similar é seguida por (XIANG; LEE, 2008) onde *Ant Colony Optimization* (ACO) é usada como meio de promover coordenação local a fim de o sistema convergir para a solução global.

Apesar dos trabalhos apresentados assumirem que o sistema de transporte esteja sempre presente, uma abordagem baseada em agentes é interessante porque ela admite uma resposta auto-organizada, evitando-se o uso de informações globais, o que pode ser devastador para o processamento do sistema. Como detalhado em (PEETERS et al., 2001), o projeto do agente, ao nível da arquitetura é fundamental para assegurar a convergência do sistema. No caso de (XIANG; LEE, 2008), é baseado em deposição de feromônios que são processados em níveis distintos de abstração no sistema, para priorizar a sobrevivência deste, sobre a otimização, de forma que o agendamento pode não ser o ótimo, mas é rapidamente alcançado. Ainda outra abordagem similar é encontrada em (SALLEZ; BERGER; TRENTESAUX, 2009) o qual introduziu a noção de espaço virtual que mimetiza o chão-de-fábrica real, através de agentes que pode expor previsões de seu comportamento no tempo, o que é usado pelos outros agentes para decidir o quão ocupado um agente está.

Outras abordagens baseadas em agentes incluem (WONG et al., 2006; LU; YIH, 2001; USHER, 2003). No primeiro, duas estratégias de negociação são discutidas: a primeira é a negociação direta entre agentes; a segunda o processo é mediado por um agente supervisor, para evitar decisões míopes (devido a falta de informações locais). Sem surpresa, a segunda abordagem apresenta maior performance. Já (LU; YIH, 2001) propõe uma identidade hierárquica entre os agentes e cada sub-montagem: tarefas executadas por agentes de ordem menos elevada são consumidas por tarefas executadas por agentes de ordens maiores, de tal forma que, por fim, no nível maior, o produto final é montado. Essas montagens e sub-montagens são agendadas através de várias métricas de filas e são feitas camada a camada. Por fim, diferente dos algoritmos tradicionais de agendamento, (USHER, 2003) faz o roteamento de tarefas em tempo real de acordo com o estado atual das diferentes máquinas envolvidas em uma ação (*job*). Isto é feito através de um agendamento em avanço de uma tarefa (a próxima tarefa é agendada enquanto a atual está sendo realizada), através do protocolo FIPA *Contract-Net*.

A Tabela 2 mostra um resumo dessa discussão.

Como pode ser percebido, o problema do agendamento está intimamente ligado ao sistema de transporte, entretanto, os paradigmas de manufatura não definem claramente o papel de um e o consequente impacto sobre o outro.

Para os objetivos deste trabalho, a alocação de recursos é feita sob demanda, um passo a frente. Também o custo do transporte é levado em conta para a decisão local do melhor recurso. Então, não há, de fato agendamento. No entanto, o uso do sistema de transporte é otimizado através de um processo contínuo de escolha dos melhores recursos disponíveis, consoante a métrica usada para o sistema de transporte. Tal somente é possível através de negociação entre os agentes do sistema. Para sistemas em que não ocorre negociação, o agendamento ainda é uma necessidade, por ser o único meio de buscar a otimização do uso dos recursos disponíveis, incluindo o sistema de transporte.

Tabela 2: Resumo de diversos métodos para agendamento

Classe do problema	**Método (Referência)**	**Paradigma**	**Abordagem computacional**	**Controle**	**Planejamento**	**Agendamento**	**Transporte**
Job-Shop Scheduling Problem	Approximation using Lagrangian relaxation method (LUH, 1998)	FMS	Centralizado	–	–	X	–
	Dynamic Scheduling Algorithm (SINRIECH; KOTLARSKI, 2002)	FMS	Centralizado	–	–	X	X
	GA, SA, MA and PSA (JERALD et al., 2005)	FMS	Centralizado	–	–	X	–
Multi-Objective Combinatorial Problem	Hybrid approach for multi-objective flexible job shop problem (XIA; WU, 2005)	Tradicional	Centralizado	–	X	X	–
	Hybrid method for multi-objective flexible job shop problem (KACEM; HAMMADI; BORNE, 2002)	Tradicional	Centralizado	–	X	X	–
Distributed scheduling	Holon-based scheduling (BABICEANU; CHEN; STURGES, 2005)	HMS	Distribuído	–	–	X	–
	FIPA complied holon-based scheduling (VRBA; HRDONKA, 2002)	HMS	Distribuído	–	–	X	–
	PROSA-based holonic scheduling (SOUSA; RAMOS; NEVES, 2007)	HMS	Distribuído	–	–	X	–
	Game-theoretical analysis (CICIRELLO, 2001)	Agent-based	Distribuído	–	–	X	–
	Ant-colony intelligence (XIANG; LEE, 2008)	Agent-based	Distribuído	–	–	X	–
	Pheromone-based algorithm (PEETERS et al., 2001)	Agent-based	Distribuído	–	–	X	–
	Pheromone-based algorithm (SALLEZ; BERGER; TRENTESAUX, 2009)	Agent-based	Distribuído	–	–	X	–
	Agent-base cooperation (WONG et al., 2006)	Agent-based	Distribuído	–	–	X	–
Production control	Agent-based production control framework (LU; YIH, 2001)	FMS	Distribuído (cooperativo)	X	–	–	–
	Single Step Product Reservation (SSPR) (USHER, 2003)	FMS	Distribuído (negociação)	X	X	X	–

3.4 Sistemas auto-otimizáveis

Um sistema é dito auto-otimizado se ele consegue otimizar o funcionamento ou uso de seus recursos autonomamente. Isso pode requerer alterações no sistema, as quais pode ser de dois tipos: paramétricas ou estruturais.

Uma alteração paramétrica ocorre quando algum parâmetro ou configuração (hardware ou software) do sistema é modificado. Exemplo: um otimizador pode calcular os parâmetros de um controlador e atualizar o filtro que implementa aquele controlador.

Uma alteração estrutural é uma modificação física (hardware ou software). Exemplo: dado que o sistema é composto de módulos que serão usados quando da necessidade, esses módulos são então inseridos ou removidos do sistema (pode ser uma remoção lógica ou mesmo física) por uma entidade de mais alto nível no sistema quando lhe convier.

Conforme proposto por (GAUSEMEIER; FRANK; STEFFEN, 2006) sistemas auto-otimizáveis realizam o seguinte processo:

(1) **Análise da situação corrente:** o sistema realiza uma verificação de seus sensores, o estado interno atual do sistema e todo o (ou parte do) histórico de observações já realizadas no passado (se necessário). Um aspecto essencial da análise da situação corrente é examinar o grau em que os objetivos perseguidos já foram alcançados;

(2) **Determinação dos objetivos do sistema:** os quais podem ser determinados por seleção, adaptação ou geração. A seleção é a escolha de uma ou mais alternativas discretas, finitas e fixadas a partir de um conjunto de objetivos dados. A adaptação se refere a uma mudança gradual de um ou mais objetivos, e geração significa a criação de novos ou a destruição de objetivos dentro do conjunto atualmente existente.

(3) **Adaptação do comportamento do sistema:** ao final de um ciclo do processo de auto-otimização, o sistema realiza a adaptação do comportamento do sistema, em termos de mudança paramétrica dos controladores, ou mediante a alteração estrutural do sistema e volta a repetir o passo (1).

Para executar tal processo de adaptação, alguns pesquisadores (como KOCH; OBERSCHELP (2004); BÖCKER et al. (2006)) sugeriram o uso de uma arquitetura modular, chamada OCM (*Operator-Controller Module*), mostrada genericamente na Figura 16.

A proposta de (KOCH; OBERSCHELP, 2004), mostrada na Figura 17, usa um sistema especialista para realizar a cognição, em uma camada chamada operador, e a camada inferior, chamada controlador, é a responsável pelo sensoriamento e ações no mundo físico.

O problema com o este modelo é o fato de ter que coadunar o funcionamento em tempo discreto do sistema especialista (no caso particular o CLIPS) e outras partes do sistema, com o funcionamento em tempo contínuo dos controladores. A grande vantagem é a possibilidade da seleção e troca de controladores, isto é, a realização de uma alteração estrutural no sistema.

(GAUSEMEIER; KAHL; POOK, 2008) posteriormente, dividiram a camada operador em duas sub-camadas: operador cognitivo e operador reflexivo, como mostrado na Figura 18.

Nesta abordagem a formação do laço cognitivo, entre o operador cognitivo e o operador reflexivo, faz com que o processo de auto-otimização se dê em tempo discreto.

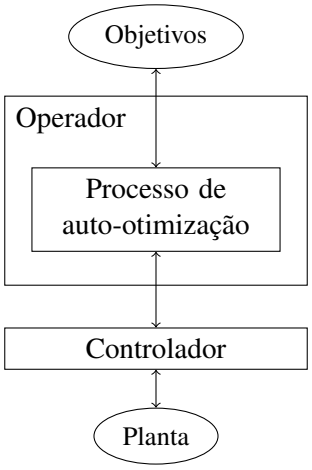

Figura 16: Módulo Operador-Controlador em 2 camadas.

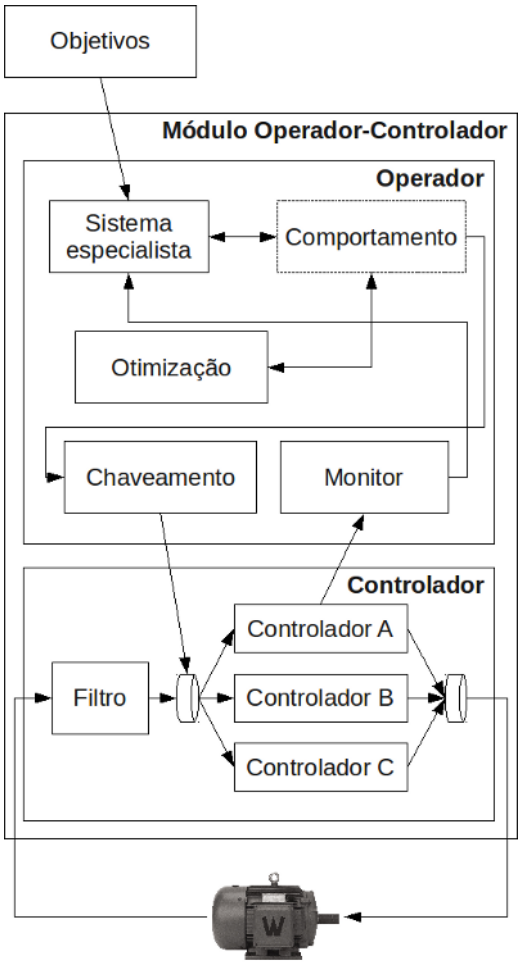

Figura 17: OCM baseado em sistema especialista. Fonte: (KOCH; OBERSCHELP, 2004)

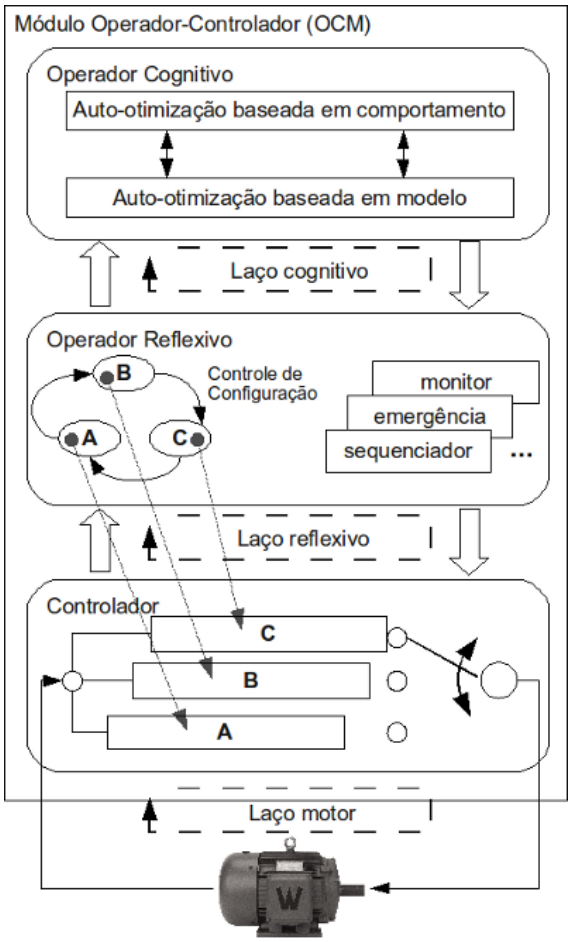

Figura 18: OCM com 3 camadas. Fonte: (GAUSEMEIER; KAHL; POOK, 2008).

A formação do laço reflexivo, entre o operador reflexivo e o controlador, faz com que o processo de seleção do controlador seja dissociado do processo cognitivo. O operador cognitivo basicamente é responsável por gerar eventos que a máquina de estados do operador reflexivo usa para selecionar qual é a melhor configuração atual, em tempo de execução. Note que as configurações e os controladores são definidos em tempo de projeto, mas a escolha do controlador é feita em tempo de execução, o que faz com que o laço motor, que envolve o controlador e a planta, seja modificado.

Além do módulo OCM, um processo de desenvolvimento de sistemas auto-otimizáveis foi igualmente proposto em (GAUSEMEIER; KAHL; POOK, 2008), cujas algumas aplicações podem ser vistas em (DONOTH; KLEINJOHANN; ADELT, 2010).

Em um sistema baseado em agentes, que permita auto-organização, os agentes devem estar aptos a reagir à mudanças ambientais, especificamente permitindo que a organização dos elementos do sistema se altere quase imediatamente. Isto posto, um modelo de agentes reativos parece ser o ideal. Olhando-se de perto a estrutura OCM, verifica-se que é exatamente isso que ocorre na sub-camada operador reflexivo.

Por outro lado, em um sistema baseado em agentes, que permita a auto-otimiza-ção, os agentes devem estar aptos a deliberar as ações a serem efetuadas a partir de sua análises e objetivos presentes. Olhando-se de perto a estrutura OCM, verifica-se que é exatamente isso que ocorre na sub-camada operador cognitivo.

Assim, a estrutura OCM é uma arquitetura de agentes para a sistemas mecatrônicos, mas de forma monolítica, isto é, um único agente que contém tanto a parte deliberativa quanto a parte reflexiva.

3.5 Projetos de Pesquisa Europeus

A proposta deste trabalho é centrada em um sistema evolutivo, isto é, um EPS, o qual é formado por uma multitude de elementos que cooperam entre si para permitir a auto-organização no sistema.

Uma abordagem para a implementação de um tal sistema requer que suas entidades possuam algumas características, a saber:

- autonomia;
- interoperabilidade;
- independência de plataforma;
- encapsulamento;
- disponibilidade/descoberta de serviços;
- proatividade.

Service Oriented Architecture (SOA) é reconhecidamente uma abordagem que permite a construção de entidades que possuem autonomia, interoperabilidade, independência de plataforma, permite o encapsulamento e possui mecanismos de verificação de disponibilidade e/ou descoberta de serviços.

A abordagem *Multi-Agent Systems* (MAS) permite, além dessas características, a proatividade.

Recentes trabalhos com SOA e MAS, aplicados ao ambiente industrial, podem ser vistos em diversos projetos de pesquisa recentes da União Europeia (EU, na sigla em

inglês), os quais são apresentados resumidamente na Tabela 3. De fato, a tabela somente elenca um projeto com uma abordagem FMS tradicional.

A vantagem de MAS sobre SOA é MAS permitir que suas entidades operem independentemente das demais, tornando-as sujeitos ativos no sistema, enquanto em SOA, uma entidade é um sujeito passivo, exclusivamente a reagir a solicitações externas. A diferença, pois, reside na proatividade. Além disso, em um MAS, um sistema conceitualmente similar a SOA pode ser construído a partir do conceito de agentes reativos. Portanto, pode-se dizer que faz-se um SOA através de um MAS (agentes reativos), mas não o oposto (RIBEIRO; BARATA; COLOMBO, 2008).

Seria possível a implementação de um EPS baseado em SOA, tal como a proposta de (SHEN et al., 2007), contudo, como EPS faz uso do conceito de agente, nada mais natural que usar MAS para sua implementação.

Tabela 3: Iniciativas de investigação recentes em SOA e MAS

Paradigma	**Nome do Projeto**	**Tópicos**
FMS	FP7-NMP COSMOS - COSt-driven adaptive factory based on MOdular Self-contained factory units (http://cordis.europa.eu)	Comportamento autônomo de unidades fabris; controle descentralizado multicamada; controle em três níveis interconectados; conectividade interoperável com dispositivos e unidades fabris; inteligência local; colaboração entre equipamentos/-dispositivos para completar uma tarefa específica.
SOA	FP7 IMC-AESOP – ArchitecturE for Service Oriented Process – Monitoring and Control (http://cordis.europa.eu)	Serviços web em tempo-real; interoperabilidade; *plug-and-play*; autoadaptação; segurança; efetividade de custos; consciência no uso da energia; cooperação e integração inter-camadas de alto nível; propagação de eventos; agregação e gerenciamento.
	FP6-IST SOCRADES – Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded deviceS (http://www.socrades.eu/)	Desenvolvimento de uma infraestrutura SOA ao nível do dispositivo para encapsulamento de inteligência e habilidades de sensoriamento e atuação como serviços; gerenciamento e orquestração de serviços ao nível dos dispositivos; definição de uma metodologia para descrição de serviços, descoberta, seleção e composição de recursos.
MAS	FP7-PEOPLE COLLIS.EUS – Soft Collaborative Intelligent Systems (http://cordis.europa.eu)	Técnicas de *soft-computing* tem sido aplicadas com sucesso para analisar operações em sistemas complexos em ambientes de incertezas e desestruturados. Suas habilidades para desenvolver modelos explícitos e implícitos do comportamento do sistema, combinado com as capacidades de representar o estado do conhecimento das entidades computacionais que interagem tem levado a um número de abordagem que aplicam-se ao desenvolvimento de sistemas multiagentes.
	FP6-IST EUPASS – Evolvable Ultra-Precision Assembly Systems (http://cordis.europa.eu)	Arquiteturas MAS para reconfiguração de equipamentos e módulos dirigidos por um requerimentos de produção definidos em uma ontologia de projeto de sistema de montagem; especificação de um novo modelo de agente direcionado às necessidades específicas de montagem modular de precisão.
	FP7 NMP GRACE – InteGration of pRocess and quAlity Control using multi-agEnt technology (http://cordis.europa.eu/)	Desenvolvimento de uma arquitetura de integração de controle de qualidade e processo; desenvolvimento de mecanismos de auto.otimização e autoadaptação; desenvolvimento de sistemas de teste adaptativos; validação de protótipos.
	FP7-NMP IDEAS – Instantly Deployable Evolvable Assembly Systems (http://www.ideas-project.eu/)	Embarcar ambientes de MAS em controladores industriais para explorar aplicações reais e validação no domínio industrial ao nível de dispositivo.

Textos extraído dos respectivos sites indicados.
Uma tabela ampliada pode ser encontrada em (RIBEIRO et al., 2011)

4 PROPOSTA DE ARQUITETURA BASEADA EM AGENTES E AUTO-ORGANIZÁVEL

Este capítulo mostra a proposta de uma arquitetura auto-organizável, baseada em agentes, específica para o chão de fábrica. Ela se fundamenta, em parte, na arquitetura CoBASA (BARATA, 2005), com uma adaptação para permitir o processo de auto-otimização definido para o modelo OCM (GAUSEMEIER; FRANK; STEFFEN, 2006). De fato, a proposta é uma aplicação das ideias de duas camadas do OCM, adaptada para um sistema multiagente em uma abordagem EPS.

4.1 Visão geral

Considerando a questão-chave de pesquisa que é a criação de uma arquitetura multiagente, que não só pode realizar auto-organização, mas também a auto-otimização, específica para o ambiente fabril, e visando-se um sistema de montagem evolutivo, a solução proposta é a de uma arquitetura multiagente em duas camadas, para o projeto dos agentes, e duas camadas de agentes na interligação dos mesmos no sistema, de forma a se ter uma arquitetura híbrida, conjuntamente deliberativa (WOOLDRIGE, 2002, caps. 3 e 4) e reativa (WOOLDRIGE, 2002, cap. 5).

A Figura 19 mostra esquematicamente a visão geral da solução proposta ao problema. A arquitetura é formada por dois tipos de agentes, instanciados conforme a necessidade, e nomeados de agente cognitivo e agente motor.

A divisão em dois tipos de agentes se deve ao critério de funcionalidade: o agente cognitivo é responsável pela lógica da aplicação, seleção e integração dos agentes motores ou outros agentes cognitivos. Então, o ambiente de um agente cognitivo é o conjunto dos demais agentes do sistema. O agente motor é o responsável pela ação no ambiente físico. Portanto, tem-se dois tipos de agentes.

A divisão em duas camadas, internamente aos agentes, segue o critério da comunicação: a camada superior do agente cognitivo é a aplicação propriamente dita do agente, enquanto sua camada inferior é responsável pela comunicação com outros agentes; no agente motor, a camada superior é a responsável pela comunicação e a camada inferior é a responsável pela aplicação, portanto duas camadas.

Um sistema assim baseado é formado por um número arbitrário de agentes cognitivos e motores, postos em relação uns com os outros.

A camada inferior (de comunicação) no agente cognitivo tem dupla função: é responsável pelas mensagens de seleção do agente motor, e pelas mensagens de requisição de ação a ele. A camada superior (de comunicação) no agente motor reage às mensagens de requisição e de ação vindas de um agente cognitivo.

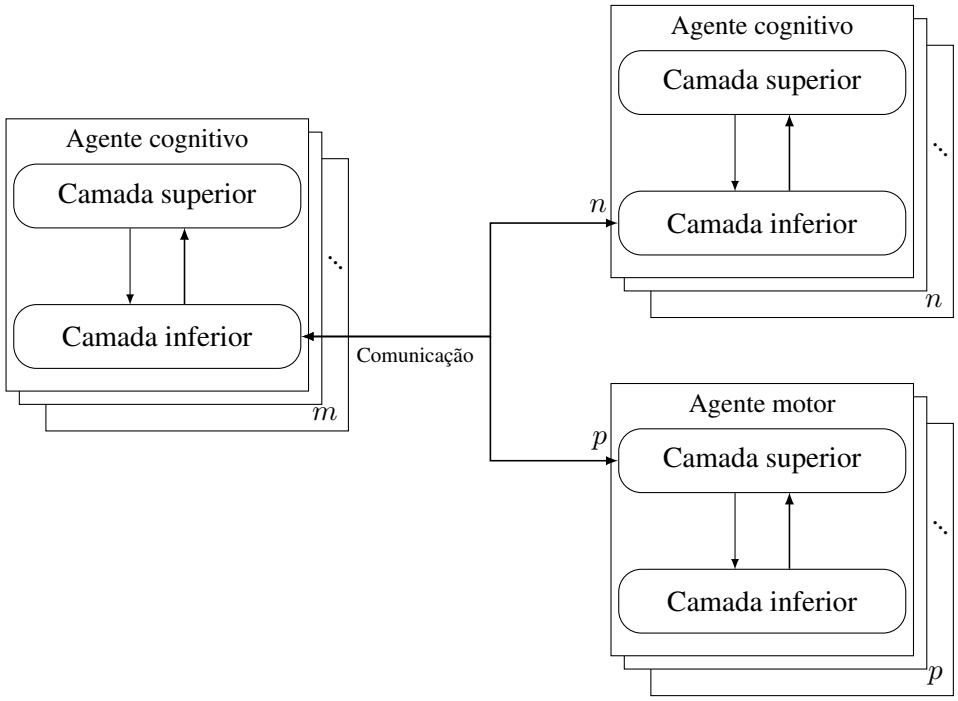

Figura 19: Arquitetura proposta: visão geral. Cada agente cognitivo pode comunicar-se com vários agentes motores e outros agentes cognitivos.

A camada superior no agente cognitivo é a atividade fim do agente, isto é, a análise do ambiente e a escolha (seleção) do agente motor adequado à realização de uma ação física no sistema; no agente motor, a atividade fim é uma ação física do sistema, por meio de sensores e atuadores.

Para uma análise mais apropriada do ambiente, o agente cognitivo pode receber as informações e eventos do mundo físico percebidos pelo agente motor (por exemplo, a posição do módulo no sistema). Isto acrescenta mais uma função às camadas de comunicação: enviar as percepções dos agentes motores aos agentes cognitivos.

Uma questão se coloca logo a partida: por que usar duas camadas na criação do agente? por que não mais?

A grande vantagem de se separar código é a utilização do princípio da divisão e conquista. Por este princípio, pode-se dividir um sistema em pedaços menores e mais maleáveis. Cada pedaço é responsável por uma única função (ou poucas funções), e o programador daquela função pode especializar o código ao máximo, permitindo o seu reuso em várias situações diferentes, ou mesmo a substituição total da implementação de um pedaço com pouca ou nenhuma interferência no restante do sistema. Por outro lado, uma divisão muito intensiva se traduz no aumento da complexidade do sistema, visto que as partes que o formam necessitam sempre de trocar informações entre si. Isto geralmente redunda em perda de performance, mas se ganha em flexibilidade. Então optou-se pela menor divisão a fim de manter o sistema o mais simples possível, com a intenção de sua implementação, sem grandes problemas de desempenho, em módulos com pouca capacidade de processamento (sistemas mínimos). A divisão mínima, portanto, é para duas camadas.

Como a existência de duas camadas aqui proposta é baseada no critério da comunicação, todo o código do agente relacionado ao processo de comunicação vai para uma camada, deixando a outra como responsável pela aplicação propriamente dita. Então, a

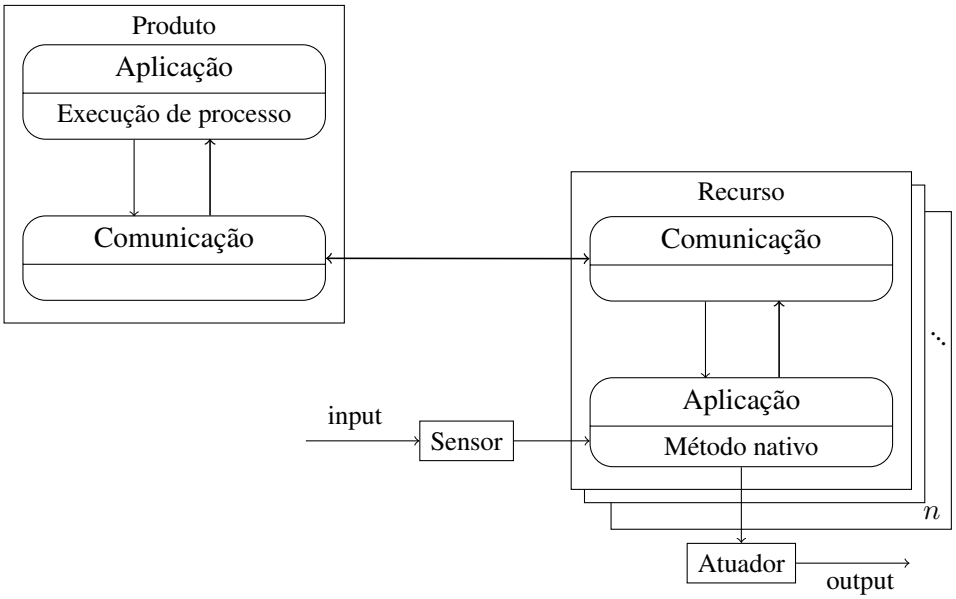

Figura 20: Arquitetura para uma aplicação em um sistema de montagem

abordagem em duas camadas permite a troca da tecnologia de comunicação sem grandes impactos na lógica da aplicação. Essa troca pode ocorrer em tempo de projeto ou de execução, dependendo da implementação mas, em geral, ocorre em tempo de projeto.

Comparando a um agente com código monolítico, cuja dependência da comunicação é completa, uma alteração qualquer na tecnologia de rede significa uma alteração no código da própria aplicação do agente. Em uma abordagem em duas camadas, uma alteração na tecnologia de rede pode ser feita sem afetar (ou afetar minimamente) a aplicação.

Portanto, o uso de apenas duas camadas, divididas pelo critério de comunicação, tende a dar uma boa flexibilidade na criação de código dos agentes sem, contudo, penalizar excessivamente a performance da aplicação.

Outra questão que se coloca é sobre o uso de apenas dois tipos de agentes no sistema: não há mais? Para analisar esta questão, analisa-se um sistema formado de um único tipo de agente, um agente monolítico responsável tanto pela parte cognitiva quanto pela parte motora: neste caso, o agente carrega em si muitas responsabilidades diferentes, replicando a estrutura de um agente de raciocínio tradicional. As limitações de performance podem ser importantes, pois, para qualquer ação, uma análise (às vezes profunda) se faz necessária.

A proposta apresentada divide o problema em dois subproblemas: as atividades que requeiram maior deliberação vão para o agente cognitivo, e as atividades de acesso ao hardware (e.g. sensoriamento e atuação) vão para o agente motor.

Por outro lado, se mais tipos de agentes fossem propostos, dividindo-se ainda mais as funções dos agentes além da cognição e da atuação (e sensoriamento), haveria um aumento inerente na complexidade do sistema. Neste caso, há um custo maior de programação, pela necessidade de mais tipos de interações diferentes a serem criadas em virtude de um maior número de tipos de agentes. Novamente, esse aumento da complexidade pode inviabilizar o uso em sistemas mínimos.

Uma aplicação típica para a arquitetura proposta é o de um sistema de montagem, formado por agentes cognitivos, que modelam os produtos a serem montados, e por agentes motores, que modelam os recursos de produção, que, por sua vez, executam alguma

função de montagem. Tal aplicação pode ser visualizada esquematicamente na Figura 20. Nesta aplicação, os agentes produtos executam processos e os agentes motores executam o acesso ao hardware, isto é, a aplicação é dividida entre dois agentes, um cognitivo, que realiza a execução do processo produtivo, e um agente motor, que realiza uma chamada de método nativo.

Uma outra aplicação possível é fazer tanto agentes cognitivos quanto motores executarem funções continuamente no sistema, sendo o agente cognitivo utilizado para funções de alto nível, por exemplo controle, e os agentes motores para funções de baixo nível, por exemplo sensoriamento e atuação, o que pode ser visualizado na Figura 21. Se o controle requer várias atividades, pode-se criar um sistema multiagente em que vários agentes cognitivos interagem entre si para gerar o controle e, por fim, algum dos agentes cognitivos será o responsável por enviar o sinal de controle correto para os agentes motores, que irão, de fato, executar as ações de controle no sistema. Nesta aplicação, portanto, os agentes controladores executam funções de alto nível (blocos funcionais) e os agentes motores executam o acesso ao hardware.

Com a proposta apresentada pode-se realizar ambas abordagens, deliberativa e reativa, eliminando-se as limitações de cada modelo. O agente cognitivo é mais próximo a um agente de raciocínio tradicional e um agente motor é mais próximo a um agente reativo.

Além dos agentes cognitivos e motores, que são mecatrônicos, uma vez que executam em um módulo mecatrônico, alguns outros tipos de agentes (não mecatrônicos) são necessários na proposta. Tais agentes são mostrados na Seção 4.6 e seguintes.

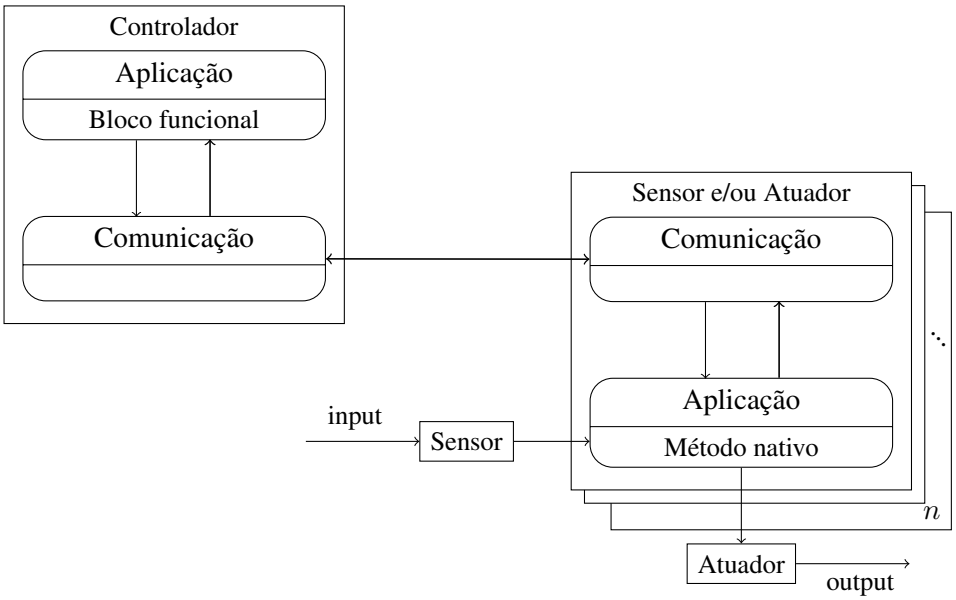

Figura 21: Arquitetura para uma aplicação em um sistema de controle.

4.2 A noção de funcionalidades

Em um EPS, cada agente, associado a um módulo mecatrônico, é chamado de agente mecatrônico e é responsável pela execução de uma única ou poucas atividades no processo produtivo. Tais atividades são as funcionalidades ou habilidades de tal módulo no sistema.

Cada agente mecatrônico, portanto, possui um conjunto de funcionalidades ou habilidades e é capaz de executá-las sob requisição externa ou por decisão interna. Então, uma

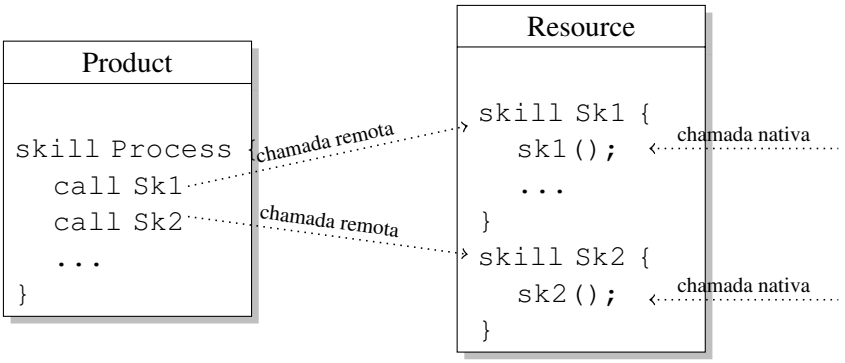

Figura 22: Exemplo de funcionalidades em agentes cognitivo e motor.

funcionalidade é uma entidade que encapsula as informações que definem uma ação específica de um agente mecatrônico no sistema. Essa ação pode ser tanto física quanto lógica, por exemplo, o acionamento de um válvula ou a execução de uma expressão matemática, ou ainda, a execução de um processo produtivo.

Na arquitetura proposta, a comunicação entre os agentes cognitivos e motores se realiza, principalmente, através da alocação e requisição de funcionalidades no sistema. As funcionalidades são definidas por todos os agentes mecatrônicos e o objetivo final do agente é a sua execução.

Há duas formas de um agente começar a execução de uma funcionalidade sua: quando requisitado por um agente externo, ou por autoexecução, em geral na inicialização do agente.

Do ponto de vista do agente que as executa, as funcionalidades são executadas localmente, mas elas podem ser de dois tipos: ou processos ou procedimentos locais.

Um processo é um conjunto de "chamadas" a outras funcionalidades, agrupadas de uma forma lógica, por exemplo, como um *workflow*. Uma chamada a uma funcionalidade nada mais é que a requisição da execução daquela funcionalidade pelo agente que contém o processo a um agente capaz de executar tal funcionalidade.

Durante a execução do processo, a chamada de uma funcionalidade pode ser local ou remota, isto é, a requisição para sua execução é no mesmo agente que executa o processo ou é em um outro agente. Então, a requisição de execução de uma funcionalidade é equivalente a uma chamada de um procedimento local ou remoto pelo agente que contém o processo.

Uma chamada de procedimento local é uma chamada a uma função escrita em uma linguagem de programação qualquer (e.g. método nativo para linguagens orientadas a objetos, como Java ou C#) que está no mesmo espaço de endereçamento do chamador. Uma chamada de procedimento remoto é uma chamada de função em outro espaço de endereçamento, em geral, em outra máquina.

A Figura 22 mostra a definição de dois agentes mecatrônicos: um agente cognitivo, chamado `Product`, que define uma funcionalidade chamada `Process`, a qual é um processo, e um agente motor, chamado `Resource`, que define duas funcionalidades chamadas `Sk1` e `Sk2`, respectivamente. Cada uma dessas funcionalidades executa uma chamada de procedimento nativo, chamados `sk1()` e `sk2()`. A funcionalidade seria, então, uma casca para uma chamada de procedimento local.

Do ponto de vista de quem chama uma funcionalidade não há diferença entre se ela é executada local ou remotamente, exceto pelo tempo de resposta.

Do ponto de vista de um agente que implementa a funcionalidade chamada, contudo, há diferença substancial se ela é executada localmente como um processo ou como procedimento nativo, pois a execução de um processo requer um motor de execução específico, o qual é substancialmente diferentes da execução direta de um procedimento escrito em uma linguagem de programação (e.g. chamadas de código nativo).

Essa diferença á particularmente importante com agentes motores, que tendem a ser executados em sistemas mínimos. Assim, sua construção deve ser mantida o mais simples possível: agentes motores executam localmente chamadas de código nativo, enquanto agentes cognitivos executam um conjunto de chamadas de outras funcionalidades, agrupadas de uma forma lógica.

Resumindo, a proposta apresenta dois tipos de agentes: um agente cognitivo e um agente motor. A diferença entre eles é que o agente cognitivo permite a execução de funcionalidades como um processo, isto é, cada uma de suas funcionalidades são executadas como uma série estruturada de chamadas de outras funcionalidades (locais ou remotas), enquanto o agente motor executa suas funcionalidades como chamadas de procedimentos locais.

Então, um agente cognitivo deve possuir um meio de executar um processo, ou seja, necessita de um motor de execução de um grafo de funcionalidades, o que é chamado um **plano de processo**. Para a criação desse motor de execução, também há necessidade de uma forma para descrever um processo. Possuindo-se uma forma para descrever um processo e uma forma para executá-lo, ele pode ser dinamicamente elaborado e enviado a um agente cognitivo para execução.

Uma alternativa para um processo, em um agente cognitivo, é um bloco funcional. Neste caso, o motor de execução de funcionalidades do agente cognitivo executa uma funcionalidade como uma função do tempo que é periodicamente chamada. Na Figura 23 é mostrado um agente cognitivo (`Modelo`) que implementa um bloco funcional e faz um `set()` em outro agente cognitivo (`Controle`). Isto pode ser usado para implementar um sistema de controle distribuído.

Por outro lado, um agente motor apenas executa uma chamada de procedimento local, o qual é, em geral, estático e definido em tempo de projeto. Algumas linguagens de programação, que possuem características reflexivas, permitem a instanciação de bibliotecas e classes em tempo de execução, definidas *a posteriori* (em geral, Java, C#), permitindo que uma funcionalidade possa ser associada a um agente em tempo de execução, mas o caráter de chamada local sempre permanece.

Se um agente deve executar uma funcionalidade remotamente, então deve haver algum meio de descrever tal funcionalidade e passá-la para a camada de comunicação, a

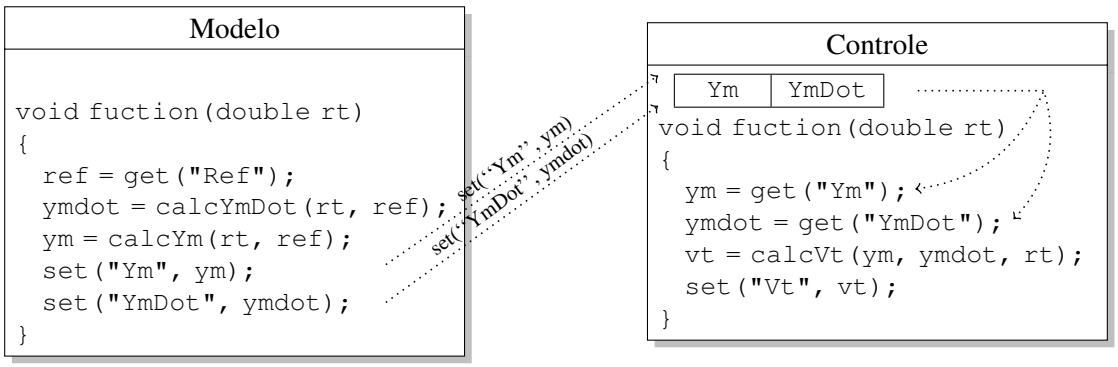

Figura 23: Agente cognitivo implementando um bloco funcional

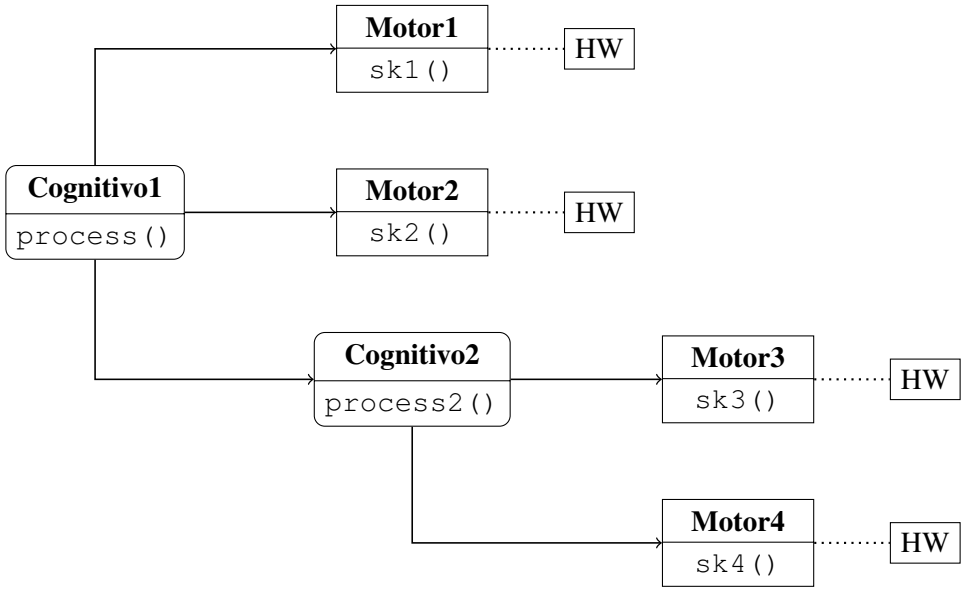

Figura 24: Hierarquia de agentes mecatrônicos chamando funcionalidades remotas.

qual deve transformar sua forma nativa em uma forma serializada para que faça parte da mensagem a ser transmitida.

Da mesma forma que um agente motor, todo agente cognitivo é, por si mesmo, um agente mecatrônico, o que significa que o agente permite a solicitação da execução de uma funcionalidade sua por um outro agente ou por si mesmo (e.g. *autostart*). Tal arranjo permite a criação de uma estrutura hierárquica de agentes, refletindo o sistema físico de maneira mais apropriada.

4.3 Estrutura hierárquica de agentes

A arquitetura proposta possui a capacidade de esconder a complexidade do sistema através de uma estrutura hierárquica de agentes. Na Figura 24, pode-se ver que o agente cognitivo `Cognitivo1` executa remotamente as funcionalidades nos agentes `Motor1`, `Motor2` e `Cognitivo2`. Do seu ponto de vista, todas as chamadas são atômicas, isto é, indivisíveis e são simplesmente chamadas às funcionalidades remotas. Entretanto, o agente `Cognitivo2` implementa a sua funcionalidade como um processo, isto é, através de um grafo que representa o conjunto de suas atividades, que, por sua vez, encapsulam as chamadas aos agentes `Motor3` e `Motor4`. Note que o agente `Cognitivo1` pode nem perceber a existência de `Motor3` e `Motor4` no sistema.

Dessa forma, em um sistema complexo que possua vários componentes mecatrônicos, estes podem ser colocados juntos para formar um módulo mecatrônico maior. Neste caso, o software de controle de cada sub-componente pode ser implementado como um agente motor, que externa algumas funcionalidades, as quais são agregadas em uma única interface através de um agente cognitivo, que, por sua vez, externa as funcionalidades do módulo complexo ao restante do sistema.

Um exemplo típico de tal arranjo é o caso de uma máquina de três eixos (VER Figura 25): cada eixo é um dispositivo mecatrônico, o qual possui um agente motor associado, e o módulo tri-axial é implementado como um agente cognitivo que faz uso de chamadas aos agentes responsáveis pelos eixos. Este arranjo pode ser usado em níveis

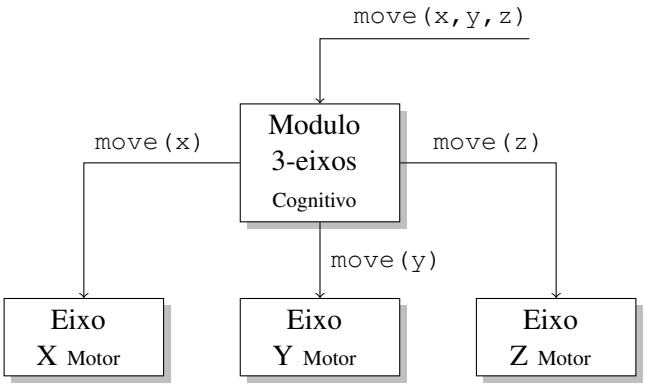

Figura 25: Arranjo de coalizão em uma máquina de 3 eixos.

ainda menores, por exemplo, cada eixo pode ser não um agente simples, mas um sistema multiagente, que possua, por si, ao menos um agente cognitivo e um agente motor.

Assim, a proposta aqui apresentada permite interligar módulos mecatrônicos em vários níveis, escondendo ou não a sua complexidade, de tal modo a favorecer a modularidade e flexibilidade do sistema.

Quando cria-se uma estrutura hierárquica como esta, de fato cria-se várias camadas de software. Novamente se ganha pela divisão e conquista, assim como o reuso de partes do sistema, flexibilidade etc., no entanto, geralmente perde-se em desempenho.

Os critérios utilizados para a divisão em maior ou menor granularidade é dependente da aplicação e o projetista deve balancear flexibilidade e desempenho. Além disso, para uma melhor resposta de auto-organização do sistema, o projetista deve evitar granularidades mais finas, com muitas camadas ou muitos vizinhos na rede formada pelos agentes cognitivos e motores para a execução de funcionalidades complexas.

4.4 Mapeando agentes mecatrônicos aos elementos de um sistema de montagem

Em um sistema de montagem, há alguns elementos típicos: produtos que estão sendo montados, recursos de montagem como *gripers*, *glues*, parafusadeiras, testes etc., elementos do sistema de transporte, tais como esteiras, AGVs, etc. Como é possível mapear tais elementos na arquitetura proposta?

A estrutura básica da plataforma, como visto até agora, é a noção de agente mecatrônico, o qual é uma agente que é executado em um módulo mecatrônico. A característica básica de um agente mecatrônico é a sua capacidade de receber requisição de execução e, efetivamente, executar as funcionalidades requisitadas.

As funcionalidades apresentam-se de dois tipos: execução de processos (visando a execução de funcionalidades remotas) e execução de procedimento local (visando o acesso ao hardware). Com isso, surgem então dois tipos de agentes mecatrônicos: um agente cognitivo que realiza a execução de funcionalidades como um plano de processo (execução de um grafo de funcionalidades), e um agente motor, que realiza o acesso ao hardware.

Para o mapeamento dos elementos físicos do sistema de montagem em agentes, primeiramente tem-se que diferenciar os módulos simples dos subsistemas mais complexos. Cada módulo mecatrônico do sistema pode ser mapeado diretamente a um agente motor. Em geral faz-se isto para quaisquer ferramentas de montagem que não fazem parte do sis-

tema de transporte. Este agente recebe o nome genérico de agente recurso (`Resource Agent`) e é mostrado na Figura 26.

Agentes de recurso, como agentes motores, são particularmente usados em sistemas físicos, uma vez que são os que realizam o acesso ao hardware.

Cada subsistema mais complexo, formado por um ou mais módulos, pode ser mapeado em um agente cognitivo, o qual irá realizar a coalizão dos agentes de recursos responsável pelos módulos simples. Este agente recebe o nome de agente líder de coalizão (`Coalition Leader Agent`) e é mostrado na Figura 27.

Agentes líderes de coalizão, como agentes cognitivos, são particularmente usados para estruturar hierarquias de agentes e módulos no sistema.

Os produtos sob montagem são mapeados a agentes cognitivos e são chamados de agentes produto. Um agente produto é um agente líder de coalizão, porque ambos executam suas funcionalidades como processos, isto é, possuem motores de execução de grafos de funcionalidades. Tal arranjo é mostrado na Figura 28.

O sistema de transporte é mapeado conforme a sua estrutura intrínseca. Da mesma forma como se faz com módulos mais simples ou mais complexos, cada elemento do sistema de transporte é mapeado ou como um agente motor ou como um agente cognitivo, conforme se necessite fazer ou não coalizão de outros agentes. Entretanto, todos os agentes de transporte, como mostrado na Figura 29, realizam o acesso ao hardware.

Uma característica importante dos agentes de transporte é a necessidade de comunicarem entre si para realizar atividades como roteamento. Daí o agente de transporte ser derivado diretamente da classe base `Mechatronic Agent` e não de `Resource Agent`. O sistema de transporte será posteriormente tratado na sessão 4.9.

Essa nomenclatura usada - agente recurso, agente líder de coalizão, agente produto, etc. - é derivada da arquitetura CoBASA (BARATA, 2005) e usada no IADE (RIBEIRO et al., 2012).

A arquitetura CoBASA foi das primeiras que utilizam o conceito de agentes para modelagem de um sistema de montagem, contudo, nem todos os elementos definidos pelo CoBASA se mostram necessários quando utiliza-se a abordagem em duas camadas aqui mostrada. Por exemplo, como agentes de recursos nada mais são que agentes motores, isto é, agentes que fazem o acesso ao hardware, o agente AMI (*Agent Machine Interface*) do CoBASA pode ser completamente suprimido fazendo-se chamadas a métodos nativos através de uma linguagem de programação reflexiva. Mesmo não usando uma linguagem de programação reflexiva, é sempre possível carregar uma biblioteca dinâmica em tempo de execução, e executar um código específico mas não definido conjuntamente com o agente.

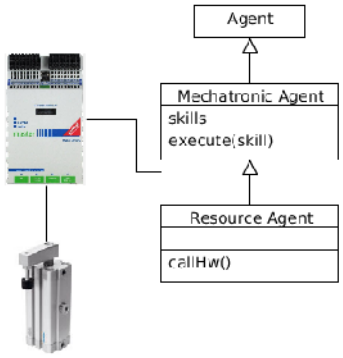

Figura 26: Mapeamento de módulos simples para agente de recurso (motor).

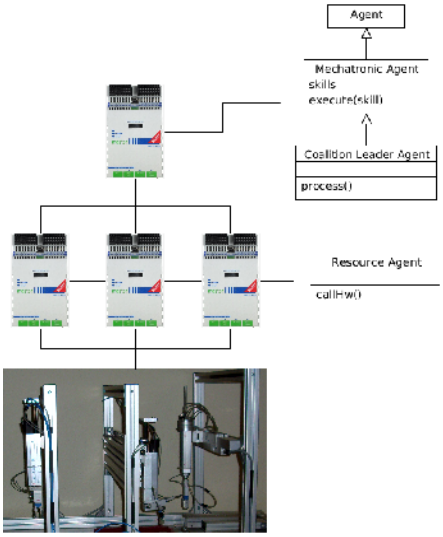

Figura 27: Mapeamento de módulos complexos para agente líder de coalizão (cognitivo).

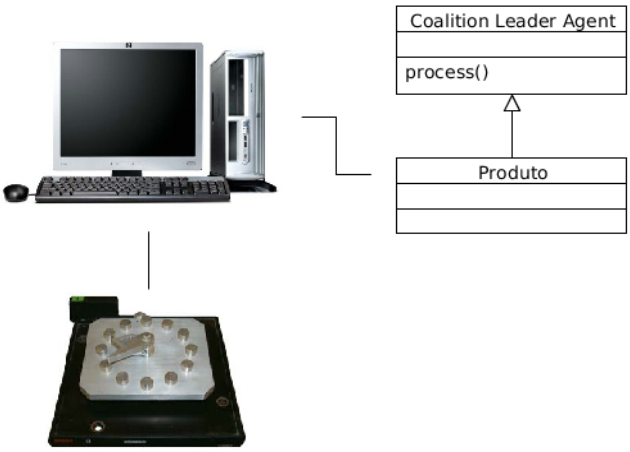

Figura 28: Mapeamento de produtos para agentes líder de coalizão (cognitivos). O PC pode executar n agentes produto.

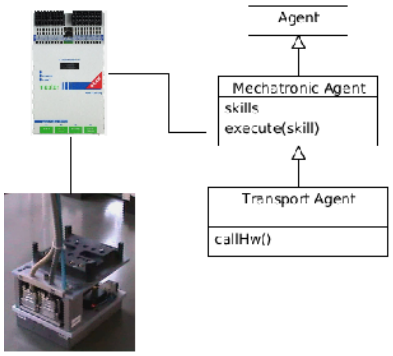

Figura 29: Mapeamento do transporte para agentes mecatrônicos. Notar que o mapeamento pode ser feito tanto para um agente motor quanto para um agente líder de coalizão, dependendo da complexidade do sistema.

Então, no âmbito de uma aplicação de sistemas de montagem, a nomenclatura do CoBASA será a utilizada na implementação desta proposta, pois é a usada no âmbito do projeto IDEAS. Portanto, usa-se os termos agente de recurso para um agente motor e agente líder de coalizão para um agente cognitivo. Da mesma forma, um agente produto será um tipo de agente líder de coalizão (portanto, agente cognitivo), e um agente de transporte será uma espécie de agente de recurso (portanto, agente motor).

A habilidade do agente cognitivo de realizar a coalizão de outros agentes, motores ou cognitivos, é a base para a resposta de auto-organização do sistema. Entretanto, há um aspecto necessário para se ter, de fato, uma resposta de auto-organização: a capacidade de de negociação do agente cognitivo com outros agentes.

4.5 Camada de comunicação

Do fato da proposta criar um sistema que possui duas camadas (ou dois tipos de agentes) decorre a necessidade de ter-se uma interface bem definida ligando uma camada a outra (ou um agente a outro), o que permitiria a substituição ou troca de um dos elementos sem a consequente modificação do outro.

A comunicação entre as camadas de um mesmo agente pode ser feita pela definição de uma interface na linguagem de programação usada para implementar a proposta. A comunicação entre agentes define as interações entre eles. A interface, neste caso, define as mensagens válidas que podem ser trafegadas entre os agentes.

Para o objetivo proposto, um protocolo de três fases é necessário:

- fase para negociação, cujo término indica que um recurso está pré-alocado para a execução por um determinado agente requerente;
- fase para a movimentação do agente requerente (físico) ao recurso; e,
- fase de execução, que é realizada após o agente físico movimentar-se para a posição adequada.

A última fase, de requisição de execução de funcionalidades, pode ser repetida várias vezes para um mesmo par agente/recurso. Nota-se que, para melhorar o desempenho global do sistema, enquanto se está na fase de execução de funcionalidades, a próxima alocação pode ser feita em simultâneo. Resumindo, diz-se que:

$$\texttt{alocação} \longrightarrow \texttt{movimentação} \longrightarrow \texttt{execução}\, n\times \parallel \texttt{próxima alocação}$$

Entretanto não há um protocolo FIPA que realize, de uma vez, todas estas fases. Em geral, em um *FIPA Contract Net protocol*, que é o protocolo padrão para negociação de contratos, a execução segue imediatamente o fechamento do contrato, o que não é o desejado.

A proposta é, então, usar o *FIPA Contract Net protocol* para a fase de alocação de recurso, e o *FIPA Request protocol* para as fases de movimentação e execução de funcionalidades.

Ao usar o *FIPA Contract Net protocol*, o agente escolhido não realiza nenhuma ação, apenas torna-se alocado. Note que um agente pode aceitar uma quantidade variável de alocações, quando por exemplo está associado a uma esteira que lhe confere um *buffer* para produtos em produção.

Pode-se escolher usar um conjunto de protocolos não padrão. Por exemplo, pode-se criar uma interface de métodos remotos que seriam chamados para realizar as atividades de pré-alocação e execução. Entretanto, tal decisão implica em deixar a proposta fechada, o que, apesar do pouco impacto em relação a aplicação em si (que está em outra camada), exige que as implementações necessitem criar mais uma camada de adequação das chamadas remotas em algo mais padronizado (e.g. classes `Dispatcher` e `Helper`), no caso da necessidade de interoperabilidade. A escolha por FIPA deixa a camada de comunicação da proposta padronizada.

A camada de comunicação é a responsável pelo tráfego das informações entre os agentes. Essas informações se resumem à negociação, à eventual movimentação no sistema, à execução de funcionalidades em outros agentes e ao sensoriamento dos agentes motores, por parte dos agentes cognitivos.

Um outra interação ocorre entre agentes do sistema de transporte, os quais necessitam, além de interagir com o hardware, interagir com outros elementos do mesmo subsistema, o que pode também ser feito através de mensagens que seguem o padrão FIPA.

4.6 Resposta de auto-organização e serviços de páginas amarelas

Além dos agentes mecatrônicos, a proposta apresentada insere dois outros tipos de agentes: o agente de páginas amarelas e o agente de distribuição. Esta seção descreve o agente de páginas amarelas e a seguinte o agente de distribuição.

A fim de que um agente mecatrônico possa divulgar as suas funcionalidades para que os demais agentes mecatrônicos possam acessá-las, o sistema deve contar com um serviço de publicação e descoberta de funcionalidades. Portanto, é necessário um agente no sistema que providencie serviços comumente denominados de “serviços de registro” ou “serviços de páginas amarelas” (“yellow pages services”) (BELLIFEMINE; CAIRE; GREENWOOD, 2007, cap. 4).

Há três serviços básicos em um agente de páginas amarelas:

(i) um serviço de registro de funcionalidades por um agente;

(ii) um serviço de busca de funcionalidades de um agente; e

(iii) um serviço de busca de agentes que podem executar uma dada funcionalidade.

Todos os agentes devem registrar suas funcionalidades no serviço de páginas amarelas, logo na sua iniciação, a fim de disponibilizá-las para os outros agentes. Posteriormente, quando um agente cognitivo necessitar saber que agentes são capazes de executar uma dada funcionalidade, utiliza o serviço de busca apropriado.

As buscas por agentes ocorre no processo de negociação: ao usar o *FIPA Contract Net protocol* para alocar uma dada funcionalidade, o agente iniciador da conversação deve enviar a mensagem (*Call for Proposal*) CFP para todos os participantes que podem executar aquela funcionalidade. Para fazer isso, o agente deve, primeiro, buscar no serviço de páginas amarelas os agentes que podem executar a funcionalidade.

Isto significa que uma descrição das funcionalidades, em uma forma que possa ser utilizada no âmbito da comunicação, é necessária, isto é, uma funcionalidade necessita ter uma forma serializável. A busca, parametrizável por agente ou por funcionalidade, também necessita ter sua forma serializável. O agente, para fins de registro ou busca, não necessita de uma forma serializável, mas tão somente um identificador único no sistema.

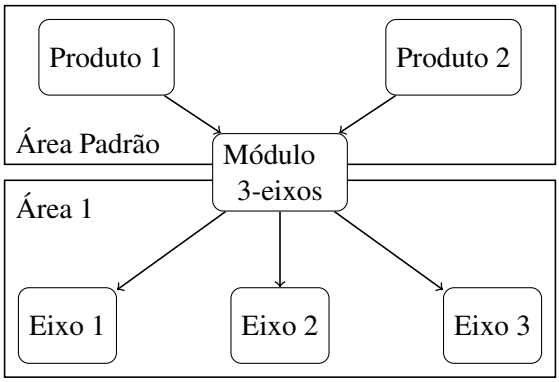

Figura 30: A noção de área.

Contudo uma forma serializável do agente deve ser criada para permitir sua instanciação sob demanda.

Além destes, outro serviço muito útil é o serviço de de-registro do agente, para realizar sua limpeza quando o agente é destruído. Entretanto, o próprio agente de páginas amarelas pode realizar este serviço de limpeza automática se houver meios de detectar a saída do agente do sistema.

O agente de serviços de páginas amarelas realiza, na prática, uma restrição lógica no sistema: para cada agente de páginas amarelas, somente o grupo de agentes mecatrônicos que realizou o seu registro está no escopo de buscas naquele agente de serviços de páginas amarelas.

Essa restrição lógica em geral modela uma restrição física. Por exemplo: em um sistema de três eixos, não há necessidade alguma de o agente de coalizão realizar uma busca além dos agentes dos eixos fisicamente presentes em uma área restrita. Então, pode-se criar um agente de páginas amarelas apenas para aquele subconjunto de agentes mecatrônicos: os três agentes recursos e o agente líder de coalizão (VER Figura 30).

Diz-se, então, que um agente de páginas amarelas define uma área. Daí o sistema terá tantas áreas quanto o número de agentes de páginas amarelas.

Mas a restrição em áreas insere um problema no sistema: quando um produto necessitar buscar por agentes, ele o faz da mesma maneira que qualquer outro agente de coalizão, isto é, faz busca em um agente de páginas amarelas; mas qual? Então, há necessidade de se ter um agente de páginas amarelas de uma área inicial, chamada área padrão, na qual todos os agentes, que externam serviços para produtos, devem se inscrever, independente de quais áreas fazem parte, uma vez que produtos tendem a passar por todas as áreas do sistema.

Essa situação é mostrada na Figura 30, onde os agentes `Eixos` são recursos, e os demais (`Módulo` e `Produtos`) são líderes de coalizão. Neste caso, o agente `Módulo 3-eixos` deve se inscrever tanto no agente de páginas amarelas da `Área Padrão`, a fim de disponibilizar serviços para os produtos, quanto no da `Área 1`, a fim poder pesquisar agentes do tipo `Eixo`.

Uma outra forma de tratar áreas e subconjuntos de agentes mecatrônicos, seria usar uma solução centralizada, em que há somente um serviço de páginas amarelas para todo o sistema e a área seria parte do protocolo de inscrição e busca de agentes e funcionalidades. Entretanto, isto gera um problema devido a centralização do serviço de páginas amarelas, que insere um ponto de falha único no sistema.

Nota-se que a noção de área gera uma restrição na quantidade de agentes mecatrônicos que podem responder a um evento de auto-organização (negociação e execução de funcionalidades), isto é, a área restringe a resposta de auto-organização do sistema.

Outra forma de se restringir a resposta de auto-organização do sistema, ou mesmo eliminar a auto-organização, é definir-se, na descrição de um plano de processo executado por um líder de coalizão, qual é o agente (através do identificador único do agente no sistema) que irá executar uma dada funcionalidade contida no plano. Neste caso, remove-se completamente a resposta de auto-organização do sistema pois não há mais negociação.

O agente de páginas amarelas não é um agente mecatrônico, pois não possui funcionalidades, seja para realizar acesso ao hardware, seja para a execução de *workflows*. Forma uma outra classe de agente que deve ser, necessariamente, adicionada à arquitetura proposta.

4.7 Agente de distribuição (Deployment Agent)

A descrição das funcionalidades são como uma *língua franca* entre os agentes. Elas devem ser publicadas e pesquisadas em serviços de páginas amarelas, devem ser trocadas entre agentes para negociação ou execução e mesmo a própria definição de agente mecatrônico está associado à existência de uma descrição de suas funcionalidades.

Para cumprir todas essas atividades, a descrição de uma funcionalidade deve ser de uma forma tal que possa ser usada durante um processo de comunicação entre agentes. Diz-se então que uma funcionalidade tem uma forma serializada.

A definição de uma forma serializada para funcionalidades permite a criação de uma forma serializada também para os agentes mecatrônicos. Tal forma teria, no mínimo, a identificação única do agente e uma lista de suas funcionalidades em suas formas serializadas.

Isto permite que o agente possa fazer a carga dessas funcionalidades dinamicamente. Neste caso, o código do agente passa a ser genérico, de forma a que pode executar qualquer tipo de funcionalidade. Então, a única diferença entre os agentes cognitivos e motores reside na definição do motor de execução de suas funcionalidades.

Em um agente cognitivo, um motor de execução é criado para executar funcionalidades descritas como um processo, ou como um bloco funcional. Em um agente motor, a funcionalidade encapsula a execução de uma chamada de procedimento nativo.

A existência de uma forma serializada para um agente mecatrônico torna possível a sua instanciação sob demanda. Para isso, um agente especial, o agente de distribuição, toma essa forma serializada e a instancia na máquina local onde é executado.

Através do agente de distribuição, consegue-se uma espécie de mobilidade de código, o que torna o sistema efetivamente dinâmico, conforme mostrado na Figura 31.

Os formatos serializados de agentes mecatrônicos, de funcionalidades e a existência de um agente de distribuição, torna possível o desenvolvimento de uma ferramenta de criação, distribuição e monitoração de agentes para o sistema.

Da mesma forma que o agente de páginas amarelas, o agente de distribuição não é um agente mecatrônico, mas um outro agente que deve ser inserido na arquitetura para permitir a instanciação de agentes mecatrônicos sob demanda.

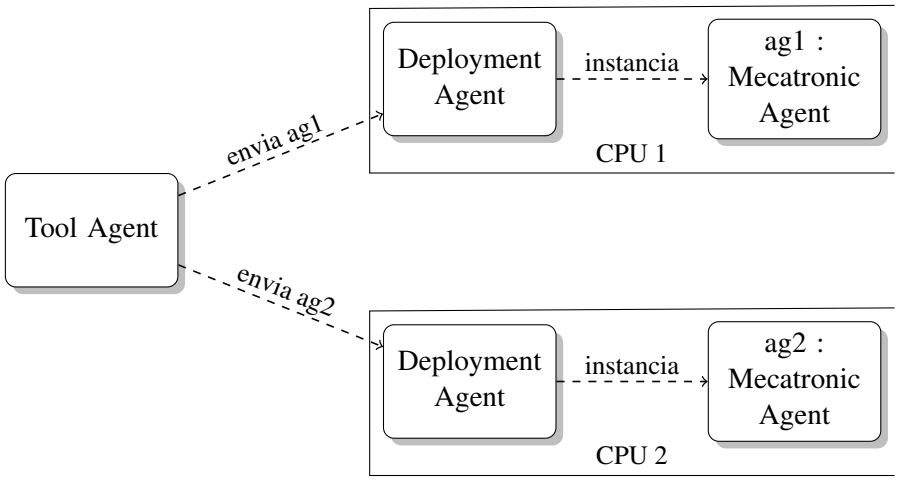

Figura 31: Instanciação remota de agentes.

4.8 Definição dos agentes da plataforma

Para melhor visualizar as características dos agentes propostos, a seguir propõe-se um modelo conceitual a partir de uma visão orientada a objetos e mostrada na Figura 32. Tais definições serão a base de uma plataforma para criação de um sistema mecatrônico multiagente, usando a arquitetura aqui proposta.

De fato, a proposta apresentada neste trabalho, para o caso de um sistema de montagem evolutivo, pode ser resumida neste diagrama, onde tanto `ResourceAgent` quanto `TransportAgent` são agentes motores, porque realizam acesso ao hardware, enquanto `CoalitionLeaderAgent` e `ProductAgent` são agentes cognitivos porque executam suas funcionalidades (seus *skills*) como um plano de processo. Por outro lado, há agentes que não são mecatrônicos, mas fornecem serviços extras e fundamentais para a plataforma.

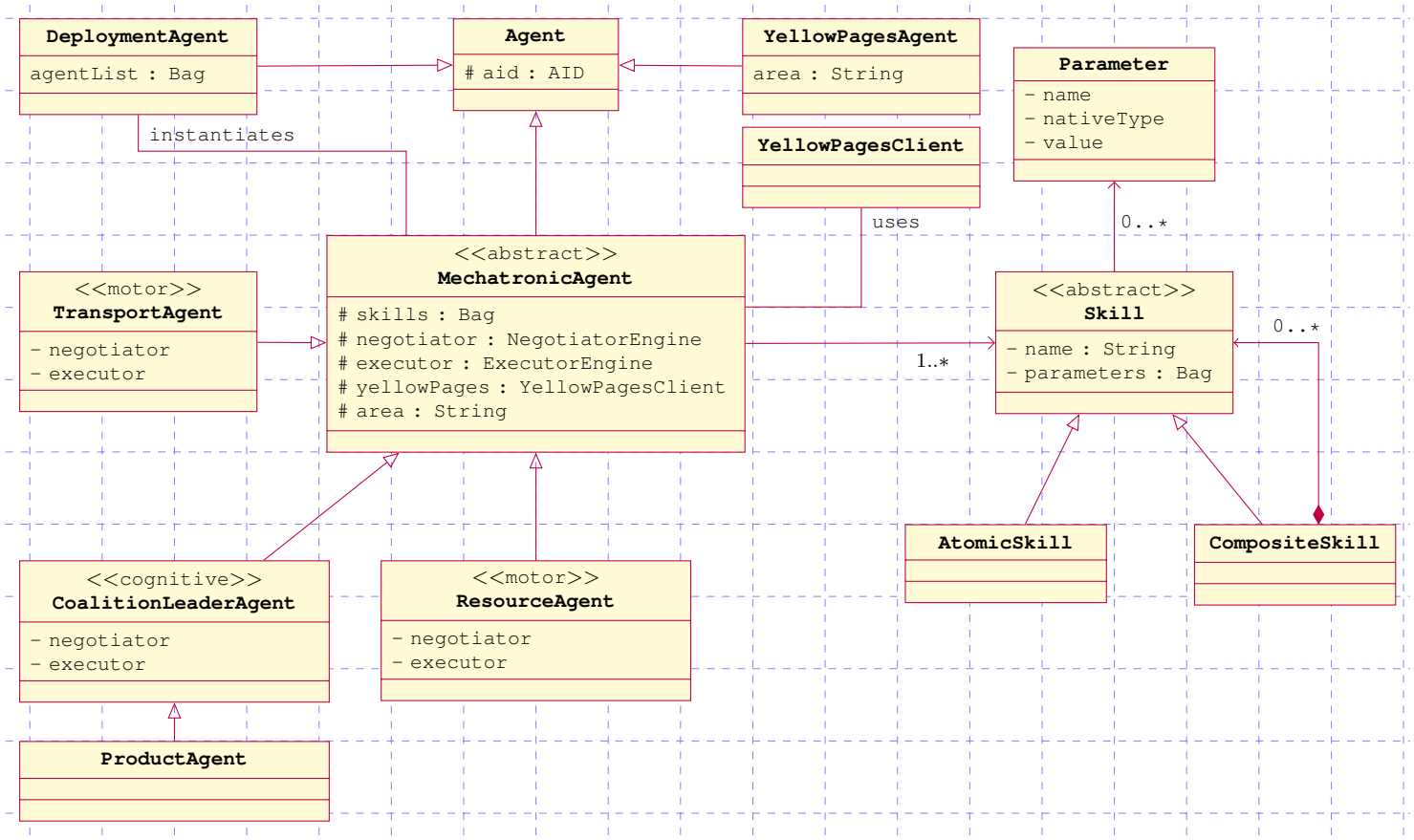

Figura 32: Definições conceituais dos agentes para a proposta.

4.8.1 A Definição dos Agentes Mecatrônicos

Os principais agentes mecatrônicos, são:

- `Resource Agent` (RA) - que representa um agente mecatrônico propriamente falando, isto é, externa os serviços que são executados pelo módulo mecatrônico que o executa, eventualmente realizando uma ação física no ambiente (uma montagem). Portanto, o comportamento global de um RA inclui os aspectos de controle do equipamento.

- `Coalition Leader Agent` (CLA) - responsável pela agregação e orquestração de agentes e funcionalidades dos outros agentes do sistema, em particular RAs e outros CLAs. Um CLA suporta a execução lógica de um processo complexo a ser executado no sistema. É o CLA que executa negociação com outros agentes para a execução das funcionalidades disponíveis e expõe, por si, uma funcionalidade ao sistema.

- `Product Agent` (PA) - é uma espécie de CLA, mas não requer externar uma funcionalidade, e, porque contém todo o conhecimento para a montagem de um produto inteiro, é um CLA de mais alto nível.

- `Transportation System Agent` (TSA) - é um agente que abstrai os sistemas de transporte. Do ponto de vista arquitetural, pode tanto ser encarado como uma forma de recurso (cuja funcionalidade é a movimentação das partes do sistema), como pode formar camadas hierárquicas.

O `MechatronicAgent` é o agente base da plataforma. Um agente mecatrônico possui um identificador único e um conjunto de funcionalidades (`Skills`). O tipo `Bag` no diagrama da Figura 32 representa uma lista de elementos. Necessariamente abstrata, nenhuma instância de `MechatronicAgent` pode ser criada, necessitando-se a criação de alguns descendentes, os quais de fato serão os agentes do sistema.

Como a diferença entre um agente cognitivo e um agente motor reside no motor de execução de suas funcionalidades, então um `CoalitionLeaderAgent` é um tipo de `MechatronicAgent` que provê uma implementação do motor de execução de um plano de processos complexos descrito como como grafo de funcionalidades. Já um `ResourceAgent` é um `MechatronicAgent` que provê a implementação da execução das funcionalidades como chamadas de método nativos.

Um produto é modelado através de um objeto `ProductAgent`, o qual é um agente `CoalitionLeaderAgent`. Os dois agentes aparecem no diagrama separadamente para mostrar as suas definições conceituais.

O `TransportAgent` é um agente mecatrônico que é basicamente um agente motor, entretanto o seu motor de execução permite algo a mais que o acesso ao hardware, como qualquer outro recurso; um agente de transporte é capaz de se comunicar com outros agentes.

Cabe às classes descendentes de `MechatronicAgent` a definição das formas serializadas dos respectivos agentes, as quais serão usadas pelo `DeploymentAgent` para a instanciação sob demanda daquele agente.

4.8.2 A Definição de *Skill*

As funcionalidades, por sua vez, são representadas no diagrama da Figura 32 por uma classe `Skill`, que define um nome e os parâmetros da funcionalidade. Tal classe também é abstrata pois define apenas as propriedades gerais para funcionalidades. Cabe aos descendentes as definições específicas.

Então, `AtomicSkill` define uma chamada de procedimento nativo. Um processo é definido por `CompositeSkill`, que permite o agrupamento de outras funcionalidades. Cabe também aos descendentes de `Skill` a definição das formas serializáveis das funcionalidades.

Um *skill* é, pois, uma funcionalidade específica dentro da arquitetura, a qual é disponibilizada por um CLA ou RA. Todas as funcionalidades externadas por agentes mecatrônicos tomam esta forma. São distinguíveis dois tipos de *skills*:

Atômico (ASk) o qual é integrado a um RA, possui os detalhes técnicos de interação e integração com o equipamento concreto sendo controlado. Em particular ele encapsula uma instância da biblioteca de baixo nível (o nível do controle de E/S) e mapeia funções de baixo nível com as funções de ordem mais alta ao nível do agente.

Composto (CSk) o qual é integrado a um CLA, suporta o projeto do processo de coalizão de *skills* na forma de um grafo de *skills* (um *workflow*). É chamado de composto, porque admite a inserção de *sub-skills*, incluindo outros CSk, o que permite o desenho de *workflows* de qualquer profundidade.

Um *skill* especial, somente executado localmente quando da execução de um plano de processo em um CLA, é um *Skill* de Decisão (DSk) usado para resolver, em tempo de execução, expressões booleanas e executar outros *skills* do plano de forma condicional (VER Seção 5.3.5 para mais detalhes).

Associar *skills* aos aspectos funcionais do sistema é uma abordagem divergente do CoBASA, o qual associa aos aspectos de conhecimento do sistema. Para o CoBASA, um *skill* é uma habilidade de um agente, o qual é exposto na forma de um contrato. Esta noção é a noção encontrada em SOA. A visão funcional permite uma abordagem direta para a implementação.

4.8.3 Definição dos Demais Agentes

Os outros agentes que não são mecatrônicos são:

- `YellowPageAgent` (YPA) - é um agente que externa serviços especializados de páginas amarelas para as funcionalidades e demais agentes na plataforma. . Cada plataforma deve possuir ao menos um agente de páginas amarelas, instanciado dentro do *container* principal. Cada YPA extra delimita um escopo de busca para agentes e um contexo específico para CLAs e RAs.

- `DeploymentAgent` (DA) - é um agente especializado em receber e instanciar uma versão serializada dos agentes principais da plataforma. É interessante notar que este é, de fato, o único agente que deve ser diretamente instanciado no módulo mecatrônico onde a plataforma será executada, pois, a partir dos DAs, os CLAs e RAs são instanciados.

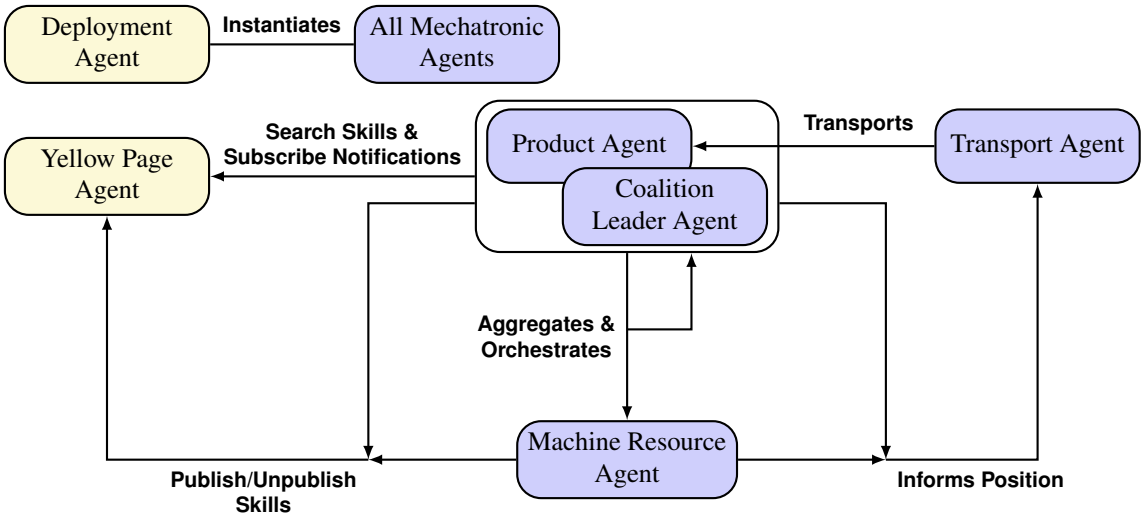

Figura 33: Os agentes da arquitetura e seus relacionamentos.

O diagrama da Figura 32 também consta o `DeploymentAgent` que representa o agente de distribuição e o `YellowPageAgent` que representa o agente de serviços de páginas amarelas. Note que os agentes mecatrônicos utilizam uma classe auxiliar, que equivale à parte cliente de acesso aos serviços de páginas amarelas, chamada `YellowPagesClient` para fazer acesso ao agente.

Este modelo é o conjunto mínimo de agentes e classes que define a plataforma. As implementações irão definir mais campos e classes de apoio, bem como ferramentas para criação e visualização das formas serializadas.

Note-se que, por causa das formas serializadas dos agentes, para um sistema iniciar, somente é preciso um único agente de páginas amarelas executando em alguma CPU, um agente de distribuição executando em cada CPU que se queira instanciar agentes, e um agente gerador das formas serializadas dos demais tipos de agentes, para se ter um sistema pronto para operar.

Uma visão das relações entre os agentes até agora discutidos, pode ser visualizada no diagrama da Figura 33.

Tal modelo possui as condições suficientes para prover tanto auto-organização quanto auto-otimização, isto é, o sistema multiagente é capaz de prover o verdadeiro *plug-and-produce*, através da entrada e saída de agentes mecatrônicos no sistema, reagindo de imediato a essas ocorrências e, ao mesmo tempo, ser capaz de autonomamente escolher os melhores recursos disponíveis na produção para a realização de suas atividades, em especial as melhores rotas no sistema de transporte a fim de se evitar gargalos.

Como acentuado anteriormente, a resposta de auto-organização do sistema toma lugar quando da negociação entre agentes. É no processo de negociação que as características de emergência aparecem no sistema. Dito de outra forma, o sistema somente apresenta comportamento emergente, com a consequente auto-organização de seus componentes quando realiza negociação.

4.9 O sistema de transporte

As tecnologias de transporte são bastante variadas. Por exemplo, algumas características de um sistema de esteiras são:

- a velocidade de transporte é, geralmente, constante;
- pode haver acúmulo de produtos sobre o sistema;
- não há colisões;
- há rotas bem estabelecidas;
- cada esteira tem ao seu lado vizinhos bem estabelecidos, com a consequente formação de uma rede;

Por outro lado, um AGV possui como características:

- a velocidade, em geral, não é constante;
- número de elementos transportados é bem definido;
- pode haver colisões;
- não há rotas estabelecidas;
- cada AGVs tem ao seu lado vizinhos varáveis, isto é, o tráfego é feito no plano.

Como as características são diferentes, na verdade excludentes, requerem softwares diferentes para lidar com cada caso, AGV ou esteira.

Além disso, as tecnologias de transporte não se limitam somente a estes tipos considerados, variando, virtualmente, de acordo com a criatividade humana[1].

Imagine-se um sistema que conta com esteiras, *pick&place* e AGVs, interligados. Neste caso, não há possibilidade de um mesmo tipo de agente, ou um único agente de transporte ser modelado para executar sua função apropriadamente.

Poder-se-ia pensar em criar um agente cognitivo que encapsulasse toda a complexidade do sistema através de um processo, o que pode ser uma solução menos flexível e menos robusta. Apesar de tal solução centralizada ser viável para o problema, essa não é a abordagem proposta, a qual tenta alcançar a diversidade dos equipamentos da produção, com a noção de agentes e sistemas distribuídos.

Então, como se criar um sistema de transporte multiagente capaz de lidar com essa situação? Cria-se não um agente de transporte único, mas apenas uma casca (uma interface) sob a qual é desenvolvido o sistema de transporte consoante a tecnologia. Essa interface tem a finalidade de ajustar as tecnologias de transporte ao que está anteriormente estabelecido para os outros elementos do sistemas, isto é, a negociação e execução de funcionalidades.

Para tal, o agente de transporte é modelado como um agente mecatrônico, o qual define e expõe um conjunto de funcionalidades de transporte. Considera-se o mínimo de três:

[1] Veja-se por exemplo, carros autônomos de transporte e gerenciamento de armazenagem em `http://www.kivasystems.com/resources/demo/`.

(i) ajuste (*setup*) da posição de um agente (módulo ou outro agente de transporte), a fim de o agente de transporte ter informações sobre os seus vizinhos que lhe são diretamente alcançáveis;

(ii) movimentação de um agente (módulo), geralmente um produto, para uma coordenada especificada do sistema de transporte, isto é, uma posição absoluta no sistema; e

(iii) movimentação de um agente (módulo), geralmente um produto, para uma posição onde está um outro agente (módulo), geralmente um recurso, isto é, uma posição relativa a um outro agente, cujas coordenadas podem variar no tempo.

Há vários tipos de agentes de transporte, no que tange a sua tecnologia subjacente, entretanto, todos são modelados como um agente motor no sistema. Todavia, se um sistema de transporte for formado por um agrupamento de módulos (por exemplo, um conjunto de esteiras), nada impede que um agente cognitivo faça a coalizão dos agentes motores associados a cada módulo. O restante do sistema encara aquele agente de transporte como se fosse um outro recurso do sistema, pois ele expõe as mesmas funcionalidades.

Uma característica importante é que um agente de transporte executa um procedimento local, portanto, é um agente motor. Entretanto, possui uma diferença marcante em relação aos recursos normais, ele possui a capacidade de realizar uma decisão: se o ponto alvo é diretamente acessível, ele entrega sua carga diretamente ao destino, por um acionamento do hardware; se o ponto não está acessível, um processo de comunicação é realizado, e o módulo mecatrônico que solicitou a movimentação é (re)passado para um seu vizinho (novamente por um acionamento do hardware). Portanto, um agente de transporte é um recurso com a capacidade de trocar informações com seus vizinhos, formando, então, uma rede.

Tal dinâmica é mostrado na Figura 34, a qual também mostra uma maneira de se utilizar os protocolos FIPA para a definição dessa dinâmica do sistema.

Observe que `Product` é um agente cognitivo, enquanto que os agentes `Transport 1`, `Transport 2` e `Recurso` são agentes motores. Cabe à implementação, no caso dos agentes transportes, fazer a distinção entre a execução direta do acesso ao hardware da chamada à camada de comunicação para a passagem para outro elemento do transporte. Note que, neste último caso, ainda assim o acesso ao hardware é realizado.

4.10 Processo de auto-otimização

O processo de auto-otimização inclui, de uma forma resumida, (i) análise da situação corrente, (ii) definição dos objetivos, e (iii) adaptação do sistema para execução dos objetivos escolhidos. Para a inserção de um processo de auto-otimização na proposta, um sistema de montagem deve ser construído de tal forma que os agentes (e.g. produtos) possam escolher os outros agentes (e.g. recursos) mais aptos a serem utilizados, minimizando tempo, deslocamento, custos, ou alguma outra métrica do sistema.

A auto-otimização toma lugar no agente cognitivo de duas formas: a primeira através de uma funcionalidade executada automaticamente (uma funcionalidade executada sempre que o agente é inicializado), na qual um processo de otimização pode ser implementado; ou (a segunda forma) interativamente, através do processo de negociação.

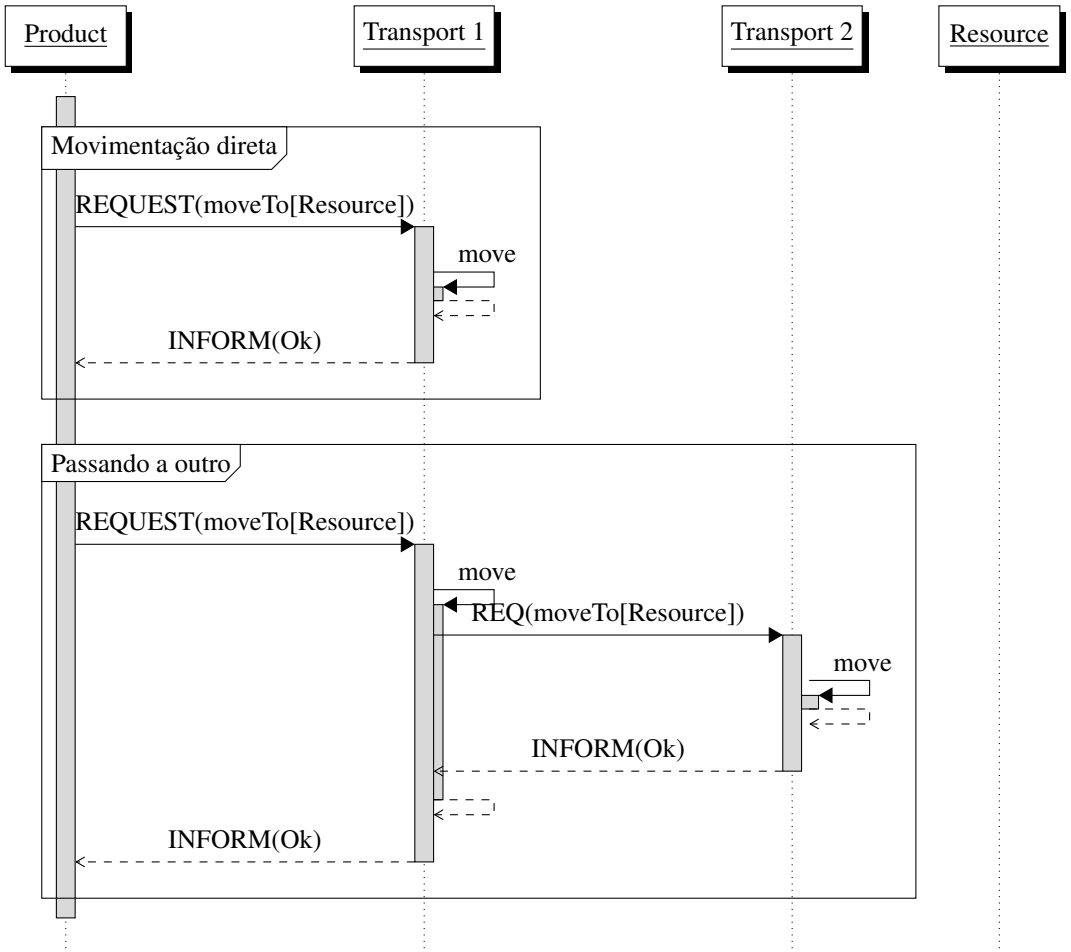

Figura 34: Mensagens de movimentação pelo sistema de transporte.

A primeira é a forma de transformar o agente cognitivo em agente de raciocínio tradicional. Nele pode-se implementar o que se queira, como por exemplo, o cálculo dos parâmetros para um determinado controlador. Após o cálculo, uma chamada a uma funcionalidade de atualização no controlador (um agente motor), faz com que os seus parâmetros sejam alterados. Enquanto o agente cognitivo calcula os novos parâmetros, o agente motor continua o seu funcionamento autonomamente, com os valores anteriores.

Por exemplo, em um sistema de controle de movimentos, o agente cognitivo pode chegar à conclusão que precisa de uma curva de aceleração mais agressiva, então ajusta os valores dos parâmetros do controlador, implementado como um agente motor, com os os novos valores. Neste caso, o processo de auto-otimização é realizado inteiramente pelo agente cognitivo, e a sua camada de comunicação é responsável em conectá-lo a(os) agente(s) motor(es) do sistema.

A segunda forma faz com que a otimização ocorra interativamente durante o processo de negociação. Essa é a forma que é descrita agora em mais detalhes.

A primeira etapa do processo de auto-otimização, a análise da situação corrente, em geral acontece quando o agente cognitivo recebe as respostas dos agentes motores em uma chamada *Call for Proposal* (CFP), durante o processo de negociação. Além disso, caso necessário, mais informações sobre o ambiente podem ser obtidas por outras mensagens em outros momentos, bem como o agente pode armazenar um histórico de ações e estados.

A segunda etapa do processo, a definição de objetivos, pode ser feita tanto em tempo de projeto como em tempo de execução. Esses objetivos devem estar escritos de tal forma que os dados recebidos via CFP possam ser comparados em função desses objetivos. Em tempo de execução, os objetivos podem ser selecionados a partir de uma lista de objetivos definidos em tempo de projeto. Se há um só objetivo, definido em tempo de projeto, de fato não há a realização dessa segunda etapa, em tempo de execução, pois o objetivo nunca irá mudar.

A última etapa do processo, de ajuste do sistema, é realizada automaticamente por causa da auto-organização, na qual o conjunto de todas as ações dos agentes influenciam.

Como um exemplo, a seguir será analisado como ocorrem essas iterações no caso que se queira otimizar, no caso minimizar, o tempo de transporte, dado que este é o objetivo a ser perseguido.

Primeiramente, em tempo de projeto, define-se que, durante a negociação, os recursos informem o tempo de processamento da funcionalidade sob negociação, enquanto o sistema de transporte informe o tempo de trânsito entre o ponto atual e o ponto onde estão os respectivos recursos participantes da negociação.

Em tempo de execução, o produto realiza um CFP, iniciando a negociação. Os recursos podem requisitar o tempo de transporte ao sistema de transporte e então responde às propostas com o tempo agregado (tempo de processamento e tempo de transporte). Ao receber as propostas, a camada de comunicação do produto, então, chama um método definido por uma interface da camada superior, passando as propostas.

O agente, então, escolhe o melhor recurso pelo menor tempo (o menor custo) dentre todos os agentes participantes da negociação, levando em consideração tanto o tempo de processamento no agente motor quanto o tempo de deslocamento pelo transporte.

Uma vez escolhido o agente, a última etapa, modificação do sistema, é consequente ao processo de negociação: ao seu final o agente escolhido já estará "alocado" e o agente cognitivo solicita ao sistema de transporte que lhe mova para a nova posição, isto é, a posição do agente motor escolhido. Dessa forma, o sistema se ajusta fisicamente para o uso da melhor rota (ver Figura 35).

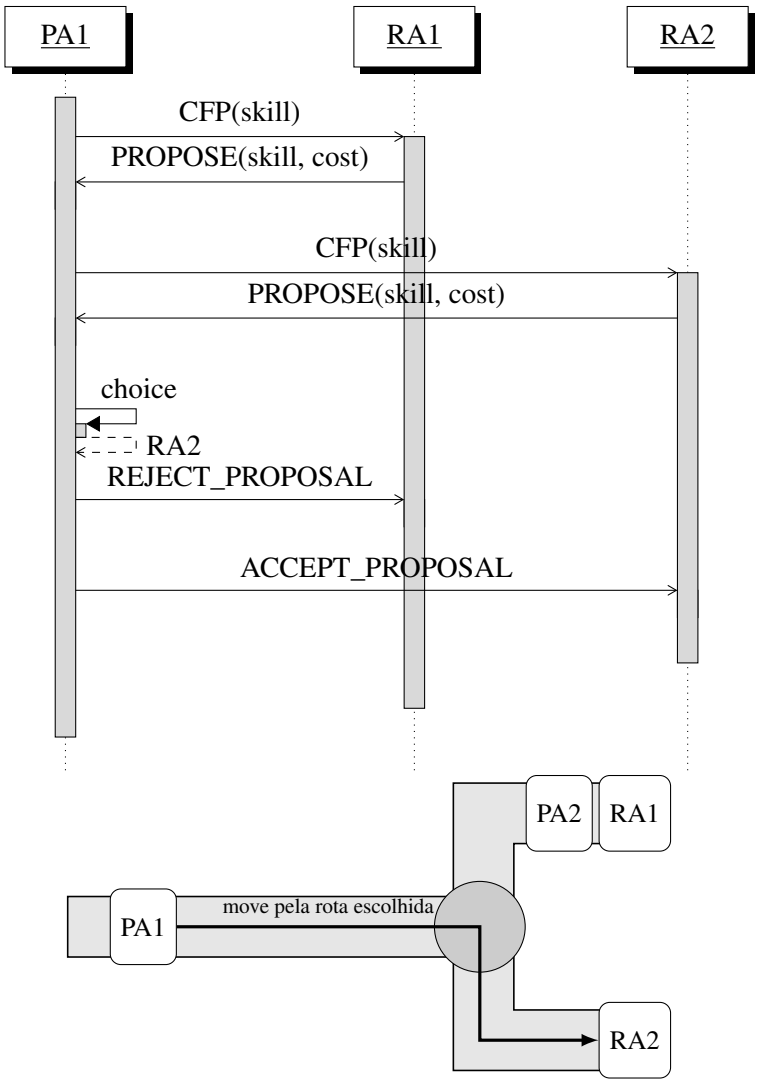

Figura 35: Escolha da melhor rota por um produto.

A interface entre a camada inferior (de comunicação) e a camada superior (aplicação) mais simples e eficiente é através de uma chamada de método onde as funcionalidades sendo negociadas são passadas como uma lista.

Como todos os agentes cognitivos (produtos ou líderes de coalizão) estarão executando estas etapas continuamente, o sistema como um todo estará realizando um uso mais racional dos recursos disponíveis.

A proposta apresenta, então, tanto um mecanismo de auto-organização, através da negociação de funcionalidades, quanto de auto-otimização, através de um processo interativo de escolha direcionada dos recursos do sistema que executam aquelas funcionalidades.

4.11 Mapeando agentes mecatrônicos aos elementos de um sistema de controle distribuído

Uma outra aplicação na qual a proposta pode vir a ser utilizada é em um sistema de controle distribuído (Distributed Control System - DCS).

Um DCS pode ser visualizado como na Figura 36, isto é, formada por um ou mais controladores, um ou mais sensores, um ou mais atuadores e ao menos um supervisor, que é o bloco responsável pela monitoração (e eventualmente configuração) das demais entidades. Cada uma delas é um nodo em uma rede, através da qual se comunicam.

Há diversas abordagens de como realizar um DCS. Tradicionalmente, cada elemento do sistema torna-se um processo em uma máquina, dentro do qual diversas *threads* realizam as suas atividades (blocos funcionais). A tecnologia de comunicação na rede tende a ser específica para a aplicação, entretanto, diversos fatores, tais como custo e facilidade de acesso aos equipamentos, têm levado os projetistas a usarem TCP/IP sobre Ethernet®.

Um DCS implementado com um sistema baseado em agentes adotaria uma abordagem semelhante, onde cada elemento da rede seria uma agente e cada tarefa (um bloco funcional) dentro dos nodos seriam implementados como comportamentos. A diferença aqui é a não necessidade de lidar com a comunicação inter-processo, porque os comportamentos seriam executados em compartilhamento do tempo por cooperação.

Todavia, pode-se querer uma completa separação dos blocos funcionais, mesmo dentro de um nodo. Neste caso, cada bloco funcional seria um agente, com poucos comportamentos associados. Se o elemento, por exemplo o sensor, apresentasse somente uma atividade, seria formado somente de um agente. Se apresentasse vários blocos funcio-

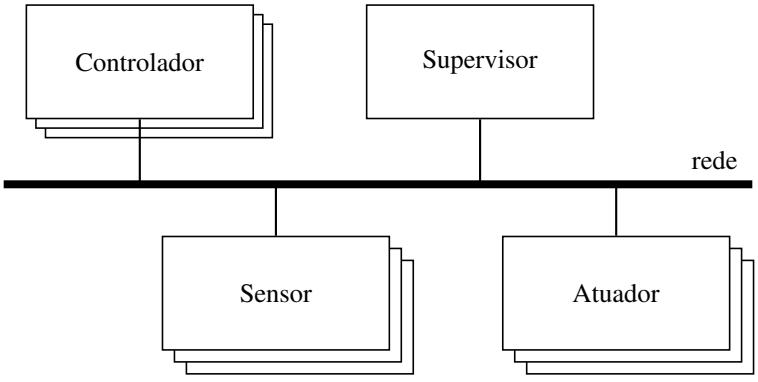

Figura 36: Visão geral de um sistema de controle distribuído (DCS).

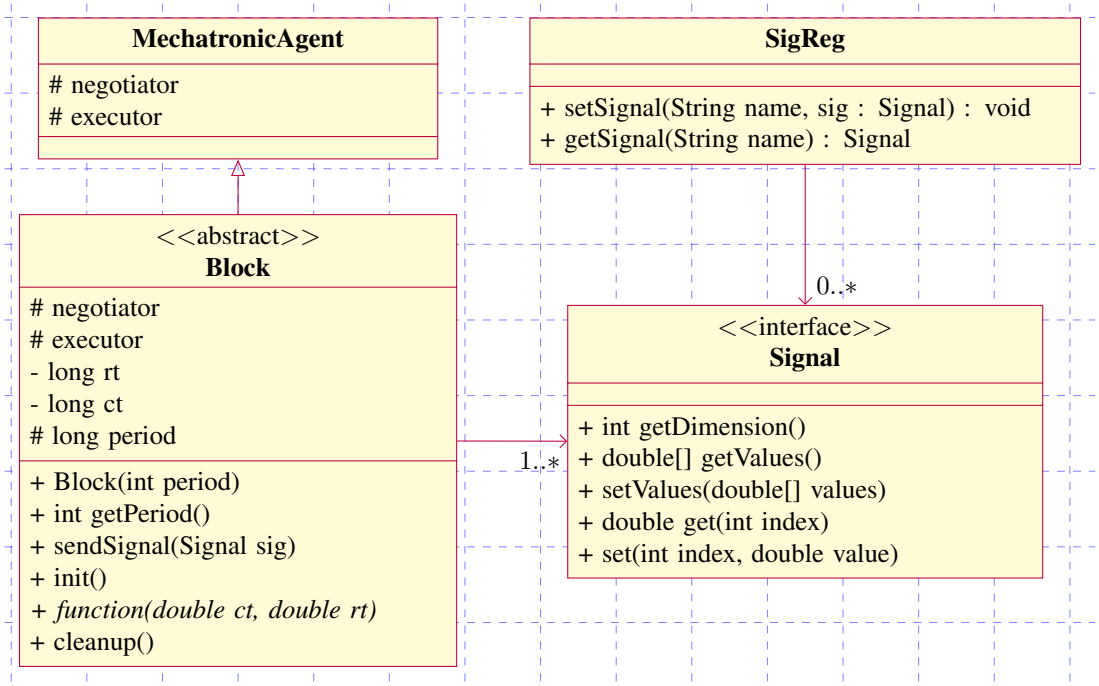

Figura 37: Classes para a plataforma DCS multiagente.

nais, seria formado por vários agentes e, portanto seria, por si mesmo, um sub-sistema multiagente.

Utilizando a abordagem proposta em duas camadas, o controlador torna-se um agente cognitivo e os demais elementos (sensores e atuadores) seriam agentes motores. Se o sistema for mais complexo, pode ser criado um conjunto de agentes cognitivos que, cooperando entre si, resolveriam o problema.

Os agentes motores funcionam praticamente da mesma forma que os agentes de recurso do IADE, respondendo aos comandos do controlador, por meio do protocolo *FIPA Request*, exceto que são modelados para possuírem um *skill* chamado `function()` que é executado automática e periodicamente.

A plataforma em si é formada a partir dos conceitos de `Block` e de `Signal`, (um bloco e um sinal) como mostrado esquematicamente no diagrama de classes da Figura 37.

Cada bloco é um agente que pode enviar e receber sinais. Ele é sempre associado a um período (`period`), de tal forma que, a cada período, a função `function(ct, rt)` é executada. Os parâmetros de `function()` indicam um tempo calculado e um tempo real. O tempo calculado é simplesmente o período vezes o número da iteração. Assim, na primeira é zero, na segunda é o próprio período, na terceira é duas vezes o período e assim sucessivamente. O tempo real é o tempo efetivamente medido no momento da chamada, o qual sofre alguma variação em relação ao tempo calculado. Um bloco é abstrato porque não tem nenhuma função implementada. Cabe aos descendentes de `Block` implementarem a função apropriada.

A classe `SigReg` é uma classe auxiliar que implementa um registro para sinais, facilitando o trabalho de quem programa os blocos funcionais para encontrar os fluxos de sinais de entrada ou de saídas a ele associado.

5 IMPLEMENTAÇÃO

Antes da descrição da implementação, propriamente dita, da arquitetura proposta, é necessário a descrição da plataforma de desenvolvimento de agentes genérico, no caso a plataforma JADE, base para a criação da plataforma de desenvolvimento de agentes mecatrônicos.

5.1 Java Agent DEvelopment – JADE

JADE é um conjunto de bibliotecas de classes que implementam *containers*[1] para agentes e FIPA para a comunicação entre estes agentes. Implementações de JADE para plataformas embarcadas e sistemas mínimos são um diferencial da plataforma. JADE implementa uma plataforma para programação multiagente e é distribuído na forma de um pacote Java, de forma que sua distribuição e uso em aplicações específicas de variados contextos é simplificada.

Uma plataforma JADE é um conjunto de *containers* os quais propiciam os serviços necessários para a execução e comunicação de agentes. A comunicação é otimizada dentro de cada *container*. Para a comunicação entre *containers* e entre plataformas é usado *Remote Method Invocation* (RMI), o que também otimiza a comunicação entre máquinas virtuais Java. O primeiro *container* criado é chamado de *Main Container* e é o que define a plataforma. Aplicações em que há redundância de *main containers* podem ser criadas, para questões de espelhamento, balanceamento de carga e robustez. Cada *container* é uma máquina virtual Java independente, ou seja, é um nodo na rede.

Para os objetivos deste trabalho, abaixo são descritas, em linhas gerais, o ciclo de vida de um agente no JADE e como os principais protocolos FIPA são ali implementados.

5.1.1 Ciclo de Vida de um Agente

Dentro de um *container* JADE, um agente é modelado através de uma classe Java, chamada `Agent`, a qual implementa uma *thread* Java que executa comportamentos continuamente. A Figura 38 mostra esquematicamente o ciclo de vida de um agente, definido como uma máquina de estados: após ser iniciado, o agente vai para o estado ativo e ali permanece até ser explicitamente colocado em espera ou suspenso (bloqueado). Um estado especial é o estado de trânsito, no qual o agente se encontra em trânsito entre dois nodos (entre um *container* e outro).

[1] Um *container* é tecnicamente uma máquina virtual Java. Uma plataforma é uma abstração de um conjunto de *containers* associados a um *main container*.

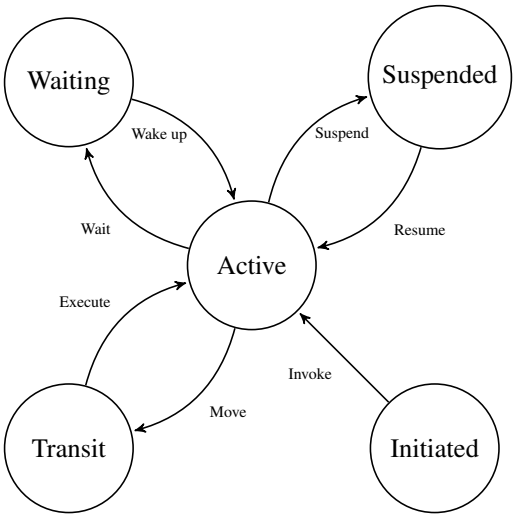

Figura 38: Estados de um agente em uma plataforma JADE. Fonte: Documentação JADE (JADE Board, 2010).

Para que o agente faça algo de útil, o programador deve sobrepor ao menos dois métodos da classe `Agent`: `setup()` e `takeDown()`. O primeiro é chamado quando da iniciação do agente, o último é chamado quando da finalização do agente.

Durante o tempo de vida do agente, o JADE irá executar os comportamentos associados àquele agente. Um comportamento é uma entidade de execução compartilhada dentro do agente, modelada através da classe `Behaviour`. Um agente pode ter vários comportamentos os quais são executados na forma de tempo compartilhado por cooperação. Os comportamentos são normalmente adicionados ao agente no método `setup()`, mas podem ser adicionados de qualquer lugar do agente, inclusive a partir de outros comportamentos. Para isso o JADE conta com a variável `myAgent` em cada comportamento, que aponta para o agente dono daquele comportamento.

A ideia por detrás do modelo de comportamentos em classes `Behaviour` é ter-se uma ferramenta capaz de definir o aspecto funcional e dinâmico do agente. Em um `Behaviour` é possível definir-se uma função específica, como a captura de dados externos, análise da situação corrente, definição da ação no meio ou a ação propriamente falando. Note-se ainda, que o comportamento global (conceitual) será o resultado das ações de todos os comportamentos (*JADE Behaviours*) do agente.

O ciclo de execução de um agente é mostrado na Figura 39. Ali pode-se verificar que os comportamentos são executados continuamente, pela execução do método `action()` do comportamento tomado da fila de ativos. Quando um comportamento bloqueia, ele sai da fila de ativos e vai para a fila de bloqueados. Assim, o agente otimiza o uso do processamento somente para os comportamentos que realmente são necessários, entretanto isso coloca um aspecto de não determinismo no tempo de execução de um comportamento.

A política de varredura das filas no JADE é *round robin*. Esta política dita que um comportamento é retirado do início da fila, é executado e colocado ao final da fila, fazendo com que o JADE execute todos os comportamentos ativos ciclicamente. A adição de um comportamento acontece sempre ao final da fila circular.

O término da execução de um comportamento é definido pelo retorno do método `done()`: enquanto retorna `false`, o comportamento é mantido em uma das listas de comportamentos, e quando retorna `true` o comportamento é removido das listas de comportamentos do agente.

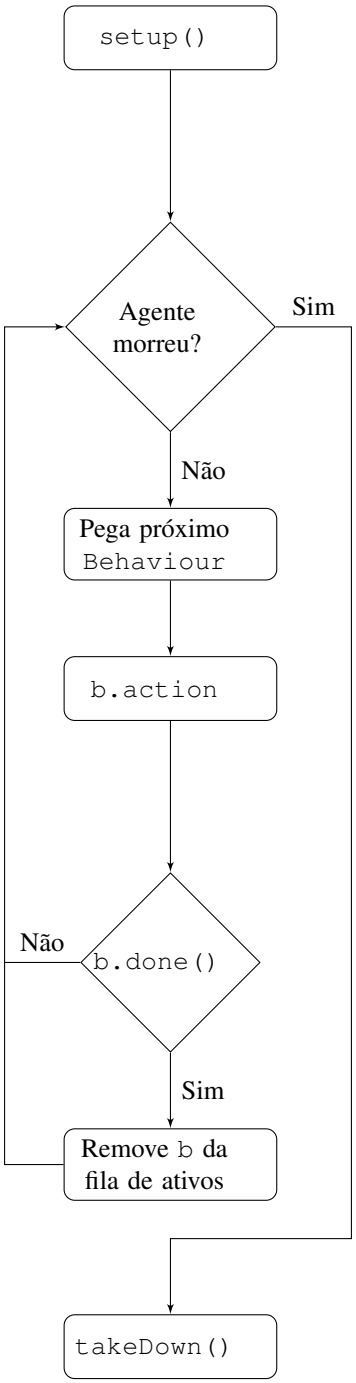

Figura 39: Execução de um agente JADE. Fonte: Documentação JADE (JADE Board, 2010).

Um comportamento pode ser colocado em bloqueio explicitamente pela chamada do método `block()` ou implicitamente através da entrada em espera por comunicação (e.g. `myAgent.receive()`). Um comportamento é colocado em ativo quando for o primeiro da fila de bloqueados, mas está pronto, isto é, não está a espera de nenhum evento externo, e.g., eventos de comunicação. Então diz-se que um agente sempre "acorda" ao final de um ciclo de execução de todos os comportamentos quando recebe uma mensagem de comunicação. Uma situação potencialmente perigosa é quando todos os comportamentos são bloqueados e não há uma mensagem de comunicação: o agente aparenta "congelar".

JADE define alguns comportamentos padrões:

- `SimpleBehaviour` - é a implementação da classe abstrata `Behaviour`. O programador deve sobrepor tanto o método `action()` quando o `done()`. É um comportamento básico propriamente falando.

- `CyclicBehaviour` - é a implementação de um `SimpleBehaviour` com o método `done()` sempre retornando `false`, dessa forma, o comportamento nunca termina (mas pode ser bloqueado implícita ou explicitamente).

- `OneShotBehaviour` - é a implementação de um `SimpleBehaviour` com o método `done()` sempre retornando `true`, isto é, o método `action()` do comportamento é executado apenas uma vez.

- `WakerBehaviour` - é um comportamento que executa uma ação (`onWake()`) após uma quantidade de tempo definida na criação.

- `TickerBehaviour` - é um comportamento que executa uma ação (`onTick()`) periodicamente, isto é, sempre que um *timer*, definido na criação, estoura. O *timer* pode ser parado ou zerado e o período pode ser variado durante a execução da ação.

- `SequentialBehaviour` - executa sub-comportamentos de modo sequencial, isto é, a cada ciclo do *scheduler* somente executa o próximo sub-comportamento quando o atual termina (seu método `done()` retorna false). Termina quando o último sub-comportamento termina.

- `ParallelBehaviour` - executa sub-comportamentos de modo paralelo, isto é, a cada ciclo do *scheduler* executa todos os sub-comportamentos que estiverem ativos. O modo de término é definido na criação e pode ser: quando o primeiro sub-comportamento termina, quando N sub-comportamentos terminam ou quando todos terminam.

- `FSMBehaviour` - executa sub-comportamentos como uma máquina de estados em que os eventos são os valores de retorno do método `onEnd()` e cada sub-comportamento é um estado da máquina. É a base para os comportamentos que implementam os protocolos FIPA.

Os comportamentos `SequentialBehaviour`, `ParallelBehaviour` e ainda `FSMBehaviour` são como uma espécie de *sub-scheduler* especializado dentro do *scheduler* padrão do JADE.

O JADE implementa os protocolos FIPA e, em especial, para o *Contract Net protocol*, JADE define dois comportamentos que implementam, cada um, uma máquina de estados

(FSM), chamados de `ContractNetInitiator`, para o iniciador e, para os demais participantes, `ContractNetResponder`. Todos os agentes mecatrônicos da plataforma devem, portanto, implementar ambos os protocolos, porque ou serão iniciadores, ou serão participantes, ou ambos, de negociação e/ou de execução de *skills*.

JADE também define dois comportamentos que implementam, através de uma FSM, o protocolo *FIPA Request*, chamados `AchieveREInitiator` para o iniciador e, para o participante, `AchieveREResponder`.

5.1.2 *Agent Managment System* (AMS)

JADE inclui vários agentes de suporte, os quais auxiliam a gerência e localização dos agentes na plataforma. Em especial, o AMS é iniciado no *Main Container* e é o responsável pela consistência da plataforma. Informações sobre cada agente que entra ou sai da plataforma é mantidas no AMS, em especial a sua identificação única (AID), evitando-se que agentes diferentes com um mesmo nome possam operar na mesma plataforma, o que geraria problemas de escopo e comunicação. Além disso, o AMS externa serviços que permitem criar e matar agentes, criar e matar *containers*, e mesmo a plataforma como um todo.

Na Listagem 5.1, é mostrado um pedaço do código de uma possível ferramenta (um agente de configuração) que faz uso do AMS a fim de monitorar a entrada e a saída de agentes na plataforma.

Listagem 5.1: Uso do AMS por uma possível ferramenta de configuração

```
1  AMSSubscriber myAMSSubscriber = new AMSSubscriber() {
2      protected void installHandlers(Map handlers) {
3          //Associate an handler to born-agent events
4          EventHandler creationsHandler = new EventHandler() {
5
6              public void handle(Event ev) {
7                  BornAgent ba = (BornAgent) ev;
8                  System.out.println(Utils.nowAsStr() + " Born agent " +
9                          ba.getAgent().getName());
10                 guiForm.bornAgent(ba.getAgent(), ba.getClassName());
11             }
12         };
13         handlers.put(IntrospectionVocabulary.BORNAGENT,
14                 creationsHandler);
15         //Associate an handler to dead-agent events
16         EventHandler terminationsHandler = new EventHandler() {
17
18             public void handle(Event ev) {
19                 DeadAgent da = (DeadAgent) ev;
20                 System.out.println(Utils.nowAsStr() + " Dead agent " +
21                         da.getAgent().getName());
22             }
23         };
24         handlers.put(IntrospectionVocabulary.DEADAGENT,
25                 terminationsHandler);
26     }
27 };
28 addBehaviour(myAMSSubscriber);
```

5.1.3 *Directory Facilitator* (DF)

O DF é o responsável por serviços de páginas amarelas entre agentes, isto é, expõe serviços de registro, cancelamento, e busca por serviços e agentes. Em geral, o DF irá fornecer informações sobre agentes (e.g. o AID) que realizam determinados serviços. Um **serviço** é definido como uma estrutura que possui um **nome** e um **tipo** e, opcionalmente, **propriedades**.

Em geral, os agentes registram-se no DF no `setup()` e fazem pesquisa quando da necessidade da descoberta de um serviço, isto é, quando um agente necessita saber qual ou quais agente executam um determinado serviço.

Note-se que o conceito de serviço é genérico e pode estar associado ou não com alguma ação física realizada pelo agente.

A Listagem 5.2 mostra um fragmento de código para registro de serviços no DF.

Listagem 5.2: Registro de serviços no DF

```
1  static public void RegisterInDF(Agent agentToBeRegistered,
2              String serviceName, String serviceType) {
3      DFAgentDescription dfd = new DFAgentDescription();
4      dfd.setName(agentToBeRegistered.getAID());
5      ServiceDescription sd = new ServiceDescription();
6      sd.setType(serviceType);
7      sd.setName(serviceName);
8      dfd.addServices(sd);
9      try {
10         DFService.register(agentToBeRegistered, dfd);
11     } catch (FIPAException fe) {
12         fe.printStackTrace();
13     }}
```

A opção de se criar um serviço de páginas amarelas próprio permite deixar a plataforma mais flexível, por não depender exclusivamente de uma opção tecnológica (no caso o JADE) e, principalmente, permitir uma maior performance, com menos trabalho para o programador dos agentes. Note que o uso do DF para busca de funcionalidades significa a decodificação do formato serializado das funcionalidades (doravante apenas *skills*), e toda manipulação dos campos internos para o registro ou busca de funcionalidades e agentes.

No momento é possível o projeto de execução de *sub-skills* paralelamente, sequencialmente ou em fluxos condicionais. Em versões futuras, pretende-se que seja possível a execução de quaisquer máquinas de estados mapeadas como *skills* compostos.

5.2 Implementação da Proposta

Como JADE é uma plataforma genérica para desenvolvimento de sistemas multiagente, vários sistemas para manufatura, que a utilizam, têm sido propostos[2], entretanto, uma plataforma específica para agentes mecatrônicos ainda se faz necessária. Daí o surgimento do *IDEAS Agent Development Environment* (IADE), uma plataforma para o desenvolvimento de sistemas mecatrônicos baseados em agentes.

O objetivo do IADE é, especificamente, disponibilizar uma infra-estrutura de desenvolvimento de aplicações multiagente para a manufatura, isto é, que seja capaz de suportar agentes mecatrônicos.

A proposta apresentada no Capítulo 4 serviu de base para a criação do IADE. Tal desenvolvimento realimentou a proposta, de tal forma que ambos, a proposta e o IADE evoluíram com o tempo. Entretanto, além do IADE, desenvolveu-se também uma pequena aplicação para controle distribuído baseado em agentes, a fim de se averiguar a possibilidade de uso da proposta em aplicações outras que não somente para um sistema de montagem evolutivo.

Esta seção e as seguintes, contudo, se deterão na aplicação da proposta a um sistema de montagem evolutivo. Em especial, se faz aqui a descrição do IADE e se discute as decisões de implementação e seus impactos. Portanto, quando for referida a implementação

[2](SHEN et al., 2006) e (MONOSTORI; VANCZA; KUMARA, 2006) mostram *surveys* sobre a aplicação de agentes na manufatura, incluindo o uso de JADE e outras plataformas.

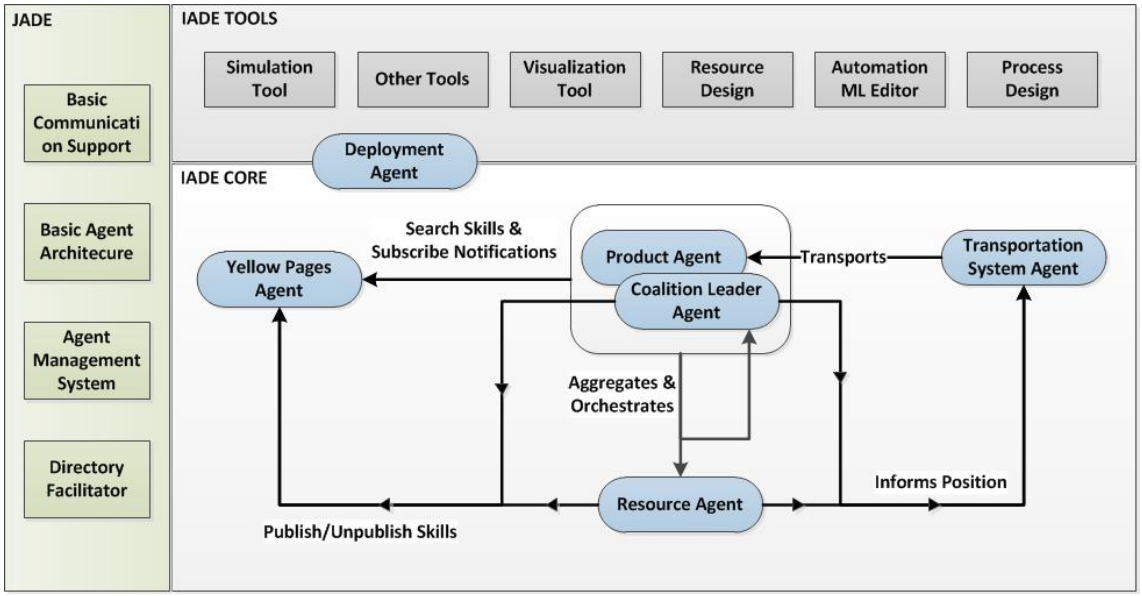

Figura 40: IADE: Arquitetura.

da proposta, está-se referindo ao IADE e vice-versa. As diferenças entre um e outro são anotadas quando aparecerem.

O IADE foi construído sobre o JADE, pois este é o padrão *de facto* para a construção de sistemas baseados em agentes. Ele já disponibiliza vários serviços e funcionalidades, incluindo o suporte da comunicação provida pelos protocolos FIPA, arquitetura básica para desenvolvimento de agentes, gerenciamento de alguns aspectos tecnologia da informação (plataforma, *containers*, etc.), e funcionalidades de publicação e subscrição de serviços e agentes.

A Figura 40 mostra a arquitetura do IADE, isto é, quais os principais agentes no *core* do *framework* e quais os serviços disponibilizados. É o desenvolvimento da proposta apresentada no Capítulo 4, acrescida dos serviços JADE que lhe servem de base.

Para aplicações mais simples, os serviços básicos do JADE são suficientes, o que elimina muita programação na aplicação. Assim, em uma primeira versão do IADE, o agente de páginas amarelas da arquitetura foi implementado usando o agente DF do JADE. Entretanto, a complexidade de aplicações como o IADE requerem serviços mais especializados, então decidiu-se pelo fornecimento do próprio serviço de páginas amarelas. Na plataforma IADE, o DF é somente utilizado pelos agentes outros que não os mecatrônicos, isto porque os agentes mecatrônicos usam os serviços especializados do `YellowPageAgent`.

RAs são os agentes que externam funcionalidades ao sistema de montagem. CLAs são agregadores e a base da reutilização de funcionalidades em qualquer nível. PAs são os CLAs de última instância que executam o processo de montagem do produto como um todo. Do ponto de vista de um agente produto, todos os demais agentes são recursos, mesmo outros CLAs. TSAs são abstrações do sistema de transporte, assim, apesar dos vários tipos de transporte existentes, estarão todos sob uma mesma interface. Fazendo um paralelo com a arquitetura mostrada no Capítulo 4, CLAs e PAs são o agente cognitivo e RAs e TSAs são agentes motores, com a exceção que TSAs devem ser capazes também de iniciar nova conversação com outros agentes (e.g. outros TSAs).

RAs e CLAs informam as suas posições ao sistema de transporte, o qual faz a movimentação dos produtos (PAs) entre os recursos disponíveis. Uma vez nas posições

adequadas os CLAs e PAs realizam chamadas aos serviços oferecidos pelos RAS e outros CLAs. Um serviço de páginas amarelas específico para a plataforma mecatrônica foi então concebido para permitir a busca por funcionalidades específicas.

Como mencionado no Capítulo 4, a nomenclatura do IADE é claramente baseada na arquitetura CoBASA (BARATA, 2005), da qual herda também as ideais das principais funcionalidade, entretanto, a abordagem agora proposta é mais simples, pois todos os serviços necessários estão melhor definidos. Igualmente melhor delineadas estão as relações entre os agentes, em especial as relações entre PAs e RAs e entre PAs e TSAs, criando-se uma abordagem mais integrada.

Todos estes agentes e classes auxiliares estão mostrados no diagrama da Figura 41.

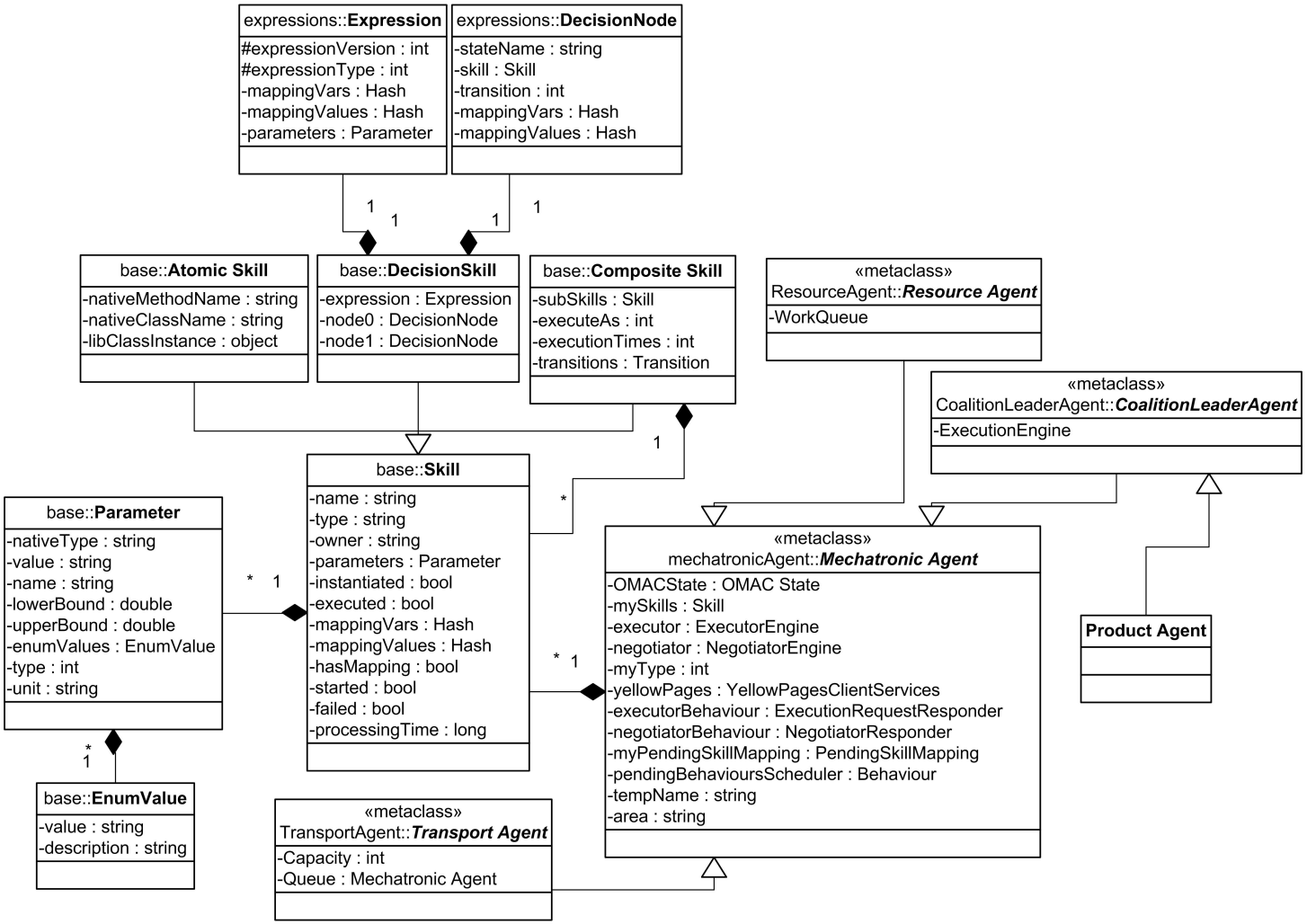

Figura 41: IADE: Modelo de classes (somente dados).

5.3 IADE: Modelo de Classes

Algumas decisões de implementação, impactam na estrutura geral do sistema. Nesta seção é colocado o modelo de dados para o *core* do IADE, contendo tanto a descrição dos agentes mecatrônicos quanto dos agentes de suporte, bem como a descrição dos *skills* que harmonizam a interface.

A Figura 41 mostra este modelo de dados em um diagrama de classes. Por simplicidade, somente os campos mais importantes são mostrados e também não constam os métodos das classes. Contudo, o modelo de dados conjuntamente com as descrições aqui colocadas devem ser suficientes para a compreensão geral do sistema e mesmo sua implementação.

5.3.1 Modelo para Agente Mecatrônico no IADE

De um ponto de vista da implementação, o `MechatronicAgent` é a classe base, a partir do qual todos os tipos de agentes mecatrônicos são gerados. É uma classe abstrata que encapsula o comportamento padrão de todos os agentes mecatrônicos no sistema. Dentre os campos existente, os seguintes merecem destaque:

`mySkills` é uma lista dos *skills* que são regenerados durante a fase de inicialização do agente. Os *skills* são lidos de um formato serializado e carregados com sua descrição completa para fins de execução local, seja ele um CSk ou ASk.

`negotiator` é um campo abstrato que guarda a referência ao objeto responsável pela negociação com outros agentes para a execução de um *skill*. Cada sub-classe deve prover a sua própria implementação.

`executor` é um campo abstrato que guarda a referência ao objeto responsável pela execução de um *skill*.

`yellowPages` é um campo que guarda a referência para a biblioteca de acesso aos serviços de páginas amarelas do ambiente.

`negotiatorBehaviour` é o lado servidor do negociador. Permite que todos os agentes mecatrônicos reajam à requisição de negociação. Ele suporta o FIPA Contract Net protocol.

`executorBehaviour` é o lado servidor do executor. Permite a todos os agentes mecatrônicos reajam à requisição de execução de um *skill*. Ele suporta o FIPA Request protocol.

`area` é um conceito inserido no IADE para se definir delimitações físicas e, eventualmente, restringir a busca por agentes no sistema. Isto restringe a resposta de auto-organização do sistema. É uma característica da implementação, mas com profundos impactos na arquitetura. Esta variável simplesmente guarda a identificação da área associada com o agente de páginas amarelas, conforme discutido no Capítulo 4.

`myType` armazena o subtipo do agente mecatrônico (RA, CLA, TSA ou PA) como um inteiro. É uma variável específica para implementação e usada no processo de serialização do agente durante o *deployment*.

`OMACState` variável que guarda o estado do agente contra um modelo OMAC do controlador associado a este agente (se aplicável ou usado). É um conceito convencional de automação.

`myPendingSkillMapping` é uma fila para a execução local de *skills*. Agentes mecatrônicos executam seus *skills* concorrentemente. Neste contexto toda requisição de execução é colocada aqui e executada quando possível. Na atual implementação, uma política de primeiro a chegar, primeiro a ser executado é considerada, contudo, execuções em lote ou por prioridade também podem ser consideradas em implementações à frente.

`pendingBehavioursScheduler` é o comportamento que controla a ordem de execução local dos *skills* contidos na lista `myPendingSkillMapping`.

`tempName` é o nome amigável do agente, que facilita a sua identificação durante as fases de projeto e distribuição. É específica da implementação, porque o JADE não permite se defina nomes amigáveis antes da criação efetiva do agente na plataforma, ou seja, antes da fase de execução do agente.

A fim de padronizar ferramentas de monitoração e controle de máquinas, *The Open Modular Architecture Control User's Group* foi criado em 1994. Em 2008, ela trocou de nome para *The Organization for Machine Automation and Control* (OMAC) para melhor refletir a sua missão[3]. Dentre seus grupos de trabalho, definiu uma máquina de estados conceitual mínima a que todo equipamento mecatrônico, compatível com seu padrão, deveria seguir, facilitando a interoperabilidade e as melhores práticas da indústria. A adequação do sistema aos estados OMAC é feito através de uma classe de adaptação (*Adapter Class* `OMACState`). O uso dos estados OMAC foi uma exigência dos parceiros industriais do projeto IDEAS e não influi nos objetivos desse trabalho.

5.3.2 Modelo para *Skill*

Agentes mecatrônicos expõem as suas funcionalidades como *skills* (`Skill`). Todos os agentes usam o mesmo modelo de *skill*, o qual, associado aos protocolos FIPA formam a linguagem em que os agentes se comunicam, isto é, negociam e realizam a execução de funcionalidades. Entretanto, os agentes, internamente e dependendo do seu tipo, distinguem dois tipos de *skills*: *skills* atômicos (ASk) e *skills* compostos (CSk), os quais são sub-classes da classe abstrata `Skill`, que tem os seguintes campos:

`name` é o nome que identifica a funcionalidade específica do domínio da aplicação, portanto tem um significado no sistema. O nome é utilizado no procedimento de busca de funcionalidades por CLAs e PAs quando estes estão tentando alocar recursos. Na aplicação IADE, além do nome, utilizou-se do número e tipo dos parâmetros para identificar univocamente em uma área um determinado *skill*, a fim de se ter uma relação em como linguagens de programação como Java ou Pascal, definem funções. Conceitualmente, pode-se optar somente pelo nome ou quaisquer outros conjuntos de campos para realizar essa identificação unívoca.

`type` identifica o tipo do *skill*: atômico, composto ou de decisão. É uma variável necessária no processo de serialização do *skill*.

[3] Veja-se o site `http://www.omac.org` para maiores detalhes.

owner identifica o nome do agente que executa o *skill*. É uma variável que foi solicitada pelos parceiros industriais para limitar (diria excluir) a resposta de auto-organização da arquitetura.

processingTime mantém o tempo médio de execução de um *skill* para fins da métrica de auto-otimização. Ao término da execução do *skill*, essa variável é atualizada.

instantiated, executed, started e failed esses campos definem o ciclo de vida de um *skill* quando da sua execução e são usados pelo *scheduler* do IADE, bem como pelos próprios *skills*.

parameters um *skill* pode ter diferentes parâmetros, que comportam-se como linhas de E/S. Cada parâmetro é definido por seu nome, seu tipo nativo (inteiro, ponto flutuante, string, etc.), e seu tipo (se é uma variável interna, um parâmetro de entrada ou de saída).

mappingVars tabela que mapeia o nome da variáveis locais (as quais são escritas na forma `skill.var_name`) com as variáveis globais em um *workflow*.

mappingValues tabela que mapeia o nome da variáveis globais com seus respectivos valores.

A existência das tabelas de mapeamento de variáveis se deve a uma decisão de implementação, mas que impacta na arquitetura geral, pois influi em como o sistema lida com a passagem de parâmetros entre *skills* no contexto de um *workflow*, isto é, dentro de um CSk (Veja-se detalhes em Seção 5.6).

Os parâmetros possuem uma estrutura toda particular, pois abstraem variáveis Java. Essa também foi uma decisão de implementação e que impacta na arquitetura: pode-se-ia escolher qualquer sintaxe e semântica para os parâmetros, contudo, como o sistema foi construído com a linguagem Java, optou-se por fazer os *skills*, notadamente os atômicos, como abstrações de métodos Java, definindo-se os parâmetros como na linguagem. Lembrar que *skills* atômicos o são porque não possuem sub-*skills*, tal como um *skill* composto.

Em especial, todos os identificadores usados na plataforma, tanto de *skills* quanto de agentes ou parâmetros, são definidos como identificadores Java e, como tal, possuem várias limitações na sua nomeação, como, por exemplo, não aceitam espaços em branco. Está sendo estudada, para uma versão futura baseada em um modelo XML, a superação dessa limitação.

A existência da variável `owner`, que define qual o agente mecatrônico que executará um certo *skill* definido dentro de um CSk, destiva o mecanismo de negociação, isto é, na execução daquele *skill*, o CLA simplesmente realiza a movimentação até o recurso definido no `owner` e, ao chegar lá, envia a requisição de execução diretamente.

Este comportamento faz com que a resposta de auto-organização seja reduzida ou mesmo completamente inexistente, quando o campo `owner` for ajustado para todos os PAs e CLAs do sistema. Apensar de ser um campo útil em um ambiente prático industrial, notadamente para testes, ou para se evitar negociação em um CLA quando se sabe, de antemão, quais os recursos disponíveis e quando executar suas funcionalidades, uma análise mais profunda mostra que este campo induz o projetista dos planos de processo dos produtos a criar um sistema sem nenhuma resposta de auto-organização. Ao se fazer isso, o objetivo maior da criação de sistemas do tipo aqui considerado (os quais requerem

auto-organização) torna-se reduzido ou simplesmente desaparece. Neste caso, propõe-se o uso de sistemas tradicionais, baseados em PLC (ou PCs industriais), que tem, de longe, mais desempenho.

5.3.3 Modelo para *Skill* Atômico

Um *skill* atômico (`AtomicSkill`) abstrai um método de baixo nível de uma biblioteca Java que faz uma ação no sistema. Como um *skill* ele possui todos os campos da classe base `Skill` e mais os seguintes:

`nativeClassName` é o nome da classe Java que irá ser instanciada e que contém o método abstraído por este *skill*.

`nativeMethodName` é o nome do método que este *skill* abstrai. Quando da sua execução, este método é chamado com os parâmetros atuais.

`libClassInstance` é uma variável que armazena uma referência para a classe Java nomeada em `nativeClassName`. É específica da implementação, usada para se evitar a re-instanciação do objeto da classe, a não ser quando requerido, pois é sempre uma operação muito custosa.

Para permitir a monitoração e ação contínua do agente sobre o hardware, quando da execução do *skill*, a chamada ao método é realizada continuamente até que que o campo `AtomicSkill.executed` seja colocado em `true`. O *skill* também deve ajustar o campo `AtomicSkill.failed` adequadamente. Note-se que o primeiro argumento de todo método que implementa um *skill* atômico deve ser justamente a referência para a definição do *skill*, a fim de o método da biblioteca possa acessar os respectivos campos.

A Listagem 5.3, mostra um pedaço de uma biblioteca Java que realiza a operação *pick and place*. Note que os *skills* `pick()` e `place()` são uma espécie de coalizão de outros *skills*, que também estão definidos na biblioteca, mas não mostrados na listagem.

Listagem 5.3: Pedaço de uma biblioteca de E/S

```
1  public void pick(AtomicSkill ask, int delayms) {
2      ask.setFailed(!pick(delayms));
3      ask.setExecuted(true);
4  }
5
6  private boolean pick(int delayms) {
7      boolean b;
8      b = gripperOpen(delayms);
9      if(!b) return false;
10     b = axisDown(delayms);
11     if(!b) return false;
12     b = gripperClose(delayms);
13     if(!b) return false;
14     b = axisUp(delayms);
15     if(!b) return false;
16     return true;
17 }
18
19 public void place(AtomicSkill ask, int delayms) {
20     ask.setFailed(!place(delayms));
21     ask.setExecuted(true);
22 }
23
24 private boolean place(int delayms) {
25     boolean b;
26     b = axisDown(delayms);
27     if(!b) return false;
```

```
28      b = gripperOpen(delayms);
29      if(!b) return false;
30      b = axisUp(delayms);
31      if(!b) return false;
32      b = gripperClose(delayms);
33      if(!b) return false;
34      return true;
35  }
```

5.3.4 Modelo para *Skill* Composto

Um CSk (`CompositeSkill`) agrega *sub-skills* de qualquer tipo, permitindo a execução de funcionalidades compostas. De fato, CSk são a base da coalizão de funcionalidades. Possui, além dos campos da classe base `Skill`, os seguintes campos:

`subSkills` uma lista de *sub-skills* que definem o *workflow* da coalizão.

`executeAs` define se os *sub-skills* são executados como uma sequência ou concorrentemente.

Como um `CompositeSkill` pode conter *skills* de todos os tipos, então pode conter ASk, DSk (ver abaixo) e outros CSk, criando-se uma estrutura hierárquica de qualquer profundidade.

A execução de um CSk não é trivial: inicia com uma análise da árvore de execução completa antes da execução propriamente falando. Os ASk são transladados para comportamentos JADE que realizam a negociação e posterior execução do *skill* atômico no agente remoto. Os CSk são convertidos em `SequentialBehaviour` ou `ParallelBehaviour` conforme o seu campo `executeAs`. O CSk de base (o mais externo) é sempre executado como sequencial. Um `DecisionSkill` é transladado para um `FSMBehaviour` que possui um ou dois ramos de execução. Uma vez o *workflow* seja transladado para comportamentos JADE, a execução propriamente falando segue as regras de execução dos demais comportamentos: o *scheduler* do IADE torna-se o *scheduler* do JADE, evitando-se assim, sub-agendamentos que pioram a performance do sistema.

Entretanto, fica claro que a quantidade de processamento envolvido para a execução de um processo (*skills* compostos) pode impactar severamente a performance do sistema quando de sua execução em sistemas mínimos (módulos mecatrônicos no chão-de-fábrica).

5.3.5 Modelo para *Skill* de Decisão

Um *skill* de decisão é modelado através da sub-classe `DecisionSkill` (DSk) e é responsável, em tempo de execução, pela análise de uma expressão booleana e a execução de nenhum, um ou outro ramo no *workflow*. Além dos campos da classe base (`Skill`), contém:

`expression` armazena uma expressão booleana a ser avaliada em tempo de execução.

`node0` o ramo que é executado quando a expressão é avaliada como verdadeira.

`node1` o ramo que é executado quando a expressão é avaliada como falsa.

Um `DecisionSkill` é uma espécie de *skill* interno (*built-in*) e está intimamente ligado aos parâmetros do *skill* composto que o contém, não tendo existência fora de um

workflow. Dessa forma, diferente de um *CompositeSkill* que está associado a um CLA (a execução de um *skill* em um CLA é a execução de um CSk) e um `AtomicSkill` que está associado a um RA (a execução de um *skill* em um RA é a execução de um ASk), o DSk não tem agentes mecatrônicos associados a ele.

5.3.6 Modelo para Recursos

Recursos são modelados através da classe `ResourceAgent`, a qual não expõe nenhum outro método além dos já existentes na classe base, isto é, `MechatronicAgent`. Entretanto, a classe de agente de recurso implementa todos os campos abstratos e comportamentos específicos. Sua função primordial é a execução de *skills* atômicos, então, RAs externam `AtomicSkills`.

Há uma relação unívoca e direta entre um ASk e um método a ser chamado (a execução do *skill* propriamente falando). Por hora, os parâmetros somente assumem os tipos primitivos e o tipo `String` de Java.

5.3.7 Modelo para Líder de Coalizão

Agentes de coalizão são modelados através da classe `CoalitionLeaderAgent`, a qual não expõe nenhum outro método além dos já existentes na classe base, isto é, `MechatronicAgent`. Entretanto, implementa todos os campos abstratos e comportamentos específicos. Sua função primordial é a execução de *skills* compostos, então, CLAs externam `CompositeSkills`.

O programador deve estar atento ao fato que toda negociação e execução de ASk em um *workflow* é feito através de mensagens FIPA, o que requer um tempo relativamente longo de execução (o tempo de troca de mensagens, em uma rede não determinística, alcança facilmente entre 100ms a 1s). Então deve-se balancear adequadamente a quantidade de CLAs no sistema. Note que, para um CLA, qualquer requisição de execução é vista como uma execução de um *skill* atômico, muito embora o agente mecatrônico que execute de fato aquele *skill* possa ser um outro CLA. Portanto a profundidade da hierarquia de CLAs no sistema também impacta diretamente em sua performance.

5.3.8 Modelo para Produtos

Um produto nada mais é que um líder de coalizão de nível mais alto na hierarquia de CLAs. Ele é modelado através da classe `ProductAgent`, que diferencia levemente da classe `CoalitionLeaderAgent`. Também pode-se simular um produto diretamente usando a classe `CoalitionLeaderAgent`.

A diferença entre um PA e um CLA é sutil: um PA não necessita externar *skills*, embora possa ser uma propriedade interessante para futuras expansões do sistema; também um PA não pode estar restrito ao conceito de área. Uma **área** é uma região fisicamente limitada do sistema produtivo na qual os CLAs tem atuação. Por exemplo: um robô de três eixos pode ser criado a partir da integração de cada módulo mecatrônico que forma um eixo simples. Assim, haverá um agente mecatrônico (RA) associado a cada eixo (p.ex. R_x, R_y, R_z) e um agente mecatrônico (um CLA) para o robô cartesiano como um todo (p.ex. $Robo$). Então, quando o CLA que implementa o robô fizer buscas por recursos, faz sentido em se restringir essa busca. Note-se que a auto-organização vai permitir a substituição de qualquer eixo (por causa de uma falha, por exemplo) sem que o sistema necessite ser reprogramado, o que fatalmente aconteceria em soluções não auto-organizadas. En-

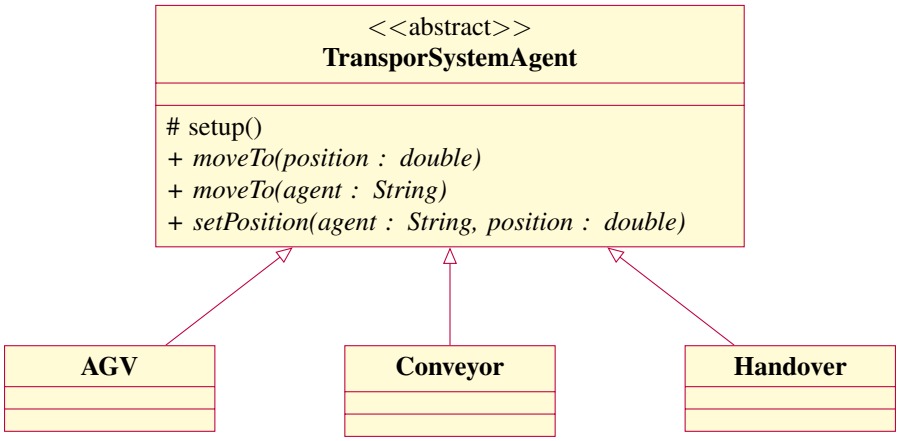

Figura 42: Classes do sistema de transporte.

tretanto, para produtos, a área não tem sentido, pois a busca por agentes deve ocorrer no sistema produtivo como um todo.

Os primeiros demonstradores para o projeto IDEAS simplesmente usam CLAs para implementar PAs. Esses CLAs possuem um único *skill*, chamado `process()`, a partir do qual é construído (e executado) o plano de processo para o produto e o campo área é deixado com o valor padrão `NONE` (que significa que a área não foi definida e então a busca será feita por todas as áreas). Contudo versões posteriores irão ser realizadas com uma classe específica: `ProductAgent`, a qual além desses detalhes de implementação contará também com um comportamento específico para lhe dar com o sistema de transporte.

5.3.9 Modelo para o Sistema de Transporte

O sistema de transporte é modelado por um conjunto de agentes de transporte, cada qual específico do dispositivo a que está associado. O diagrama da Figura 42 mostra os agentes mais comuns.

O agente `AGV` é usado para modelar um veículo autônomo, o agente `Conveyor` é usado para modelar esteiras e o agente `Handover` é usado para modelar elementos de transição do sistema, por exemplo, alimentadores ou nodos de uma rede de esteiras. Então, o `Handover` é a entidade de ligação: sempre que dois módulos se encontrarem, um agente deste tipo deve ser instanciado para servir de ponto de troca entre entidades distintas do sistema de transporte.

Os agente mecatrônico, modelados pela classe `TransporSystemAgent` que é uma classe abstrata, disponibiliza apenas dois *skills* para movimentação: `moveTo(`$agent$`)` e `moveTo(`$position$`)`. Um *skill* faz uma movimentação relativa, isto é, faz a movimentação de um módulo (agente) em relação a outro módulo (agente) a partir somente da identificação do agente alvo. O outro *skill* executa uma movimentação absoluta, isto é, faz a movimentação a partir de uma coordenada absoluta do sistema de transporte. Movimentações absolutas são interessantes para testes e protótipos, mas limitam a resposta de auto-organização do sistema.

As posições seguras para o sistema de transporte devem ser explicitamente configuradas no sistema de transporte (um parâmetro que é lido na inicialização da classe, e.g. no método `setup()`), em geral um posicionamento absoluto em um AGV (daí a real necessidade da movimentação absoluta) ou uma rota em um sistema de esteiras. Da mesma forma, todo agente de transporte "sabe" onde nasce: agentes de transporte associados à

recursos tomam o posicionamento do recurso e o de produtos (e.g. paletes) sempre entram no sistema por uma entrada bem definida.

Além dos métodos para movimentação, o sistema de transporte externa um método para permitir que os recursos ou outros agentes de transporte informem as suas posições (o que torna a movimentação relativa possível): `setPosition(`$agent$`,` $position$`)`. Tais *skills* podem ser visualizados, no diagrama da Figura 42.

5.3.10 Modelo para Páginas Amarelas

Agentes de páginas amarelas (YPA) são modelados pela classe `YellowPageAgent` a qual suporta a pesquisa, publicação e registro de serviços e agentes mecatrônicos (MAs). Então, cada MA, em sua iniciação, registra os serviços que oferece (os seus *skills*) e esse registro fica disponível para que os demais agentes do sistema possam pesquisar. Esse lado servidor possui dois comportamentos principais:

`queryResponder` o comportamento que implementa o protocolo FIPA *Request* que responde à requisições de serviços de busca, publicação e remoção de publicação de agentes.

`notificationsResponder` o comportamento que implementa FIPA Request protocol que responde pelas requisições de subscrição e remoção de subscrição de agentes.

Do lado do cliente, foi desenvolvida uma classe que faz acesso a estes serviços de forma mais facilitada dentro dos MAs.

5.4 Dinâmica do sistema

Esta seção mostra uma análise da dinâmica do sistema, a qual pode mostrar o aparecimento ou não de emergentes a partir das interações entre os elementos do sistema. Com isso, uma análise crítica pode ser feita sobre a questão da emergência em um sistema EAS/EPS.

Para esta análise será feita uma experiência mental de um EAS em funcionamento, o que servirá para mostrar as interações entre os elementos do sistema, os agentes mecatrônicos e os de suporte para se produzir novos produtos.

Imagine-se então um sistema que conta com alguns recursos (RAs), um sistema de transporte com *handovers*, AGVs e *esteiras* (TSAs), e um número arbitrário de produtos (PAs).

O sistema inicia, e os agentes RAs e TSAs realizam as suas rotinas de inicialização; note-se que um sistema real provavelmente contaria com alguma ferramentas de monitoração (*Visualization Tool*) e de criação de *workflows* (*Process Design Tool*), conforme foi mostrado na Figura 40. Então, em uma interface própria pode-se acompanhar a evolução dos agentes do sistema: início, negociação de *skill*, execução de *skill*, etc.

Após a criação de alguns *workflows*, a partir da organização dos *skills* que já existem no sistema ou, eventualmente, em um repositório, a ferramenta de desenho de processo começa a inserir PAs (produtos a serem montados com base nos *workflows* desenhados) no sistema. Associados a ele um TSA responsável pelo transporte de cada PA dentro do sistema também é criado.

Acompanhe-se, agora, o que ocorre a um produto específico (por exemplo, PA1): o PA inicia a execução do seu plano de processo e busca um recurso, via negociação, que

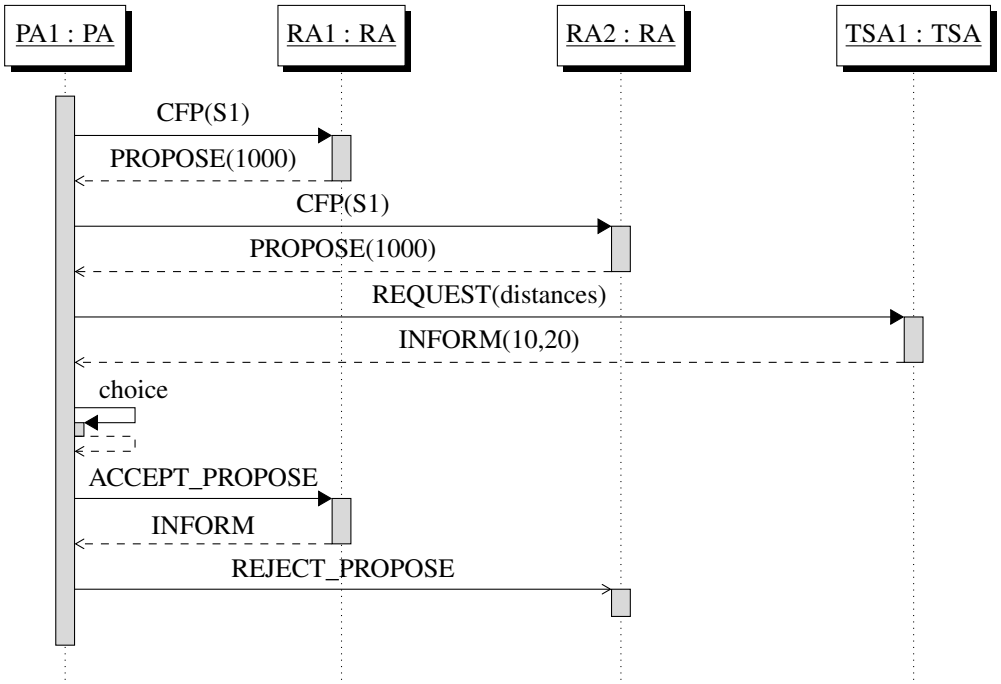

Figura 43: Escolha de um recurso por negociação.

possa executar o primeiro *skill* de sua lista (por exemplo, S1), após o que, solicita ao TSA associado a ele que o leve ao primeiro recurso conforme descoberto via negociação (por exemplo, R1).

O processo de negociação é dirigido pelo produto, mas usa dados do sistema de transporte e dos recursos para descobrir qual é o recurso mais vantajoso segundo uma métrica (VER Figura 43).

O produto, após solicitar ao sistema de transporte que o leve para o recurso alvo (note-se que o produto *não sabe onde* é a localização do recurso, mas o sistema de transporte o sabe), nada faz além de aguardar o resultado da sua solicitação. Neste ponto, ou o sistema de transporte conseguiu levar o produto ao local adequado ou, por uma falha qualquer (por exemplo, o recurso foi removido do sistema), o sistema de transporte não conseguiu cumprir com a tarefa, neste caso, levará o produto a uma posição segura no sistema, na qual ele poderá iniciar nova negociação. Apesar de simples, o algoritmo é bastante flexível.

Se uma falha catastrófica acontecer no transporte e nem ao menos ele puder levar o produto a um ponto seguro, então aquele agente de transporte irá negar todas as novas mensagens solicitando utilização (afinal ele está ocupado). Por exemplo, se um palete eventualmente bloqueia uma esteira, necessitando-se de uma ação manual, o agente nega quaisquer novas tentativas dos demais produtos de usar aquele trecho do sistema (a esteira bloqueada), de tal forma que o mecanismo de auto-organização automaticamente faz os desvios necessários.

Ao chegar na posição devida ao recurso solicitado (R1 no exemplo), o produto (PA1) pede a execução do *skill* (S1) àquele recurso (ver Figura 44). O recurso (R1) realiza a sua função (S1) e libera o produto para buscar outros recursos necessários. O produto repete o processo até completar o seu plano de processo.

Então, o sistema pode se ajustar a cada produto, autonomamente, porque não há nenhuma informação global associada com o processo de um determinado produto dentro dos recursos, pois eles simplesmente reagem a uma organização externa a eles. Essa or-

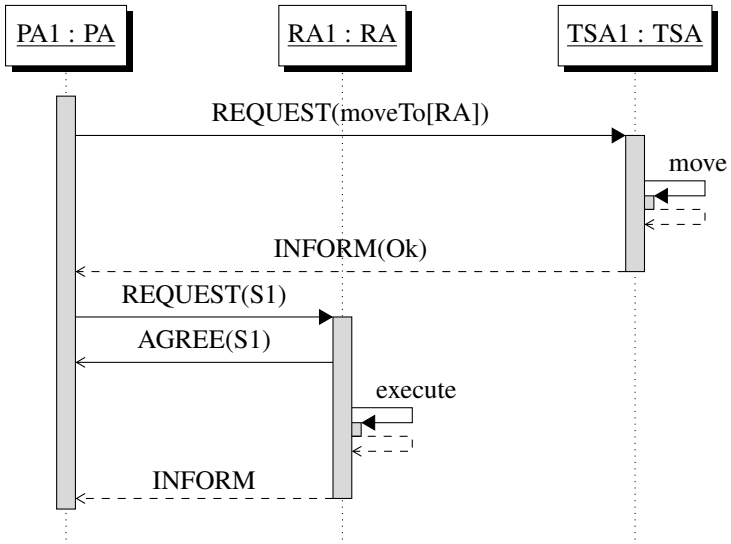

Figura 44: Execução de um *skill*.

ganização está contida dentro de outro agente, o produto e é uma inversão do processo produtivo tradicional, centrado no PLC, em que os produtos são agentes passivos e pouco inteligentes, enquanto os recursos da produção são os sujeitos ativos. A proposta EPS, ao contrário, é fazer com que os produtos possuam a inteligência do processo e os recursos saibam apenas executar o seu *skill*, o qual está intimamente ligado com o módulo mecatrônico associado.

Como o produto realiza uma negociação, ele também não sabe *exatamente* o processo que irá seguir, uma vez que os agentes de recursos que executarão as atividades do plano de processo são definidos em tempo de execução. Nenhum dos agentes contém, portanto, informação explícita sobre o comportamento global do sistema, nem mesmo informação de outros participantes.

Entretanto, à medida que o tempo passa, o sistema alcança um grau de organização maior (maior ordem) pois começa com peças separadas que depois são postas juntas e um produto é montado. Aqui parece então que EPS possui afinal emergência! Mas há uma outra visão que é necessário perceber: o produto não sabe *exatamente* o processo que irá seguir, porque não tem definido o recurso a ser usado, mas o processo *em si mesmo* está embutido dentro dele, portanto, seguir o processo seria igual a seguir uma programação no PLC (exceto pela escolha dos recursos), o que parece afinal não ser nenhuma novidade radical aparente no sistema e, portanto, não haveria emergência!

Agora, analisa-se o plano de processo: em geral ele é definido no setor de engenharia de processo da fábrica, e contém os passos necessários para a construção do produto a partir de *skills*. O plano em si é construído quebrando-se em partes o produto e definindo-se os *skills* necessários para a sua montagem utilizando-se os agentes de recursos existentes (disponíveis no sistema ou em um repositório) ou a serem construídos em algum módulo mecatrônico.

Para *skills* existentes, o sistema comporta-se como já analisado acima, então deve-se analisar agora o caso da necessidade de um novo *skill*.

Um produto que executa um plano que contém um novo *skill* ainda não existente no sistema (p.ex. o *skill* S2) pode iniciar o seu plano normalmente. O produto vai executar os *skills* já existentes até o ponto em que necessita executar esse novo *skill*. Se ao finalizar uma negociação o produto não encontra um recurso que possa executar o *skill*, conceitu-

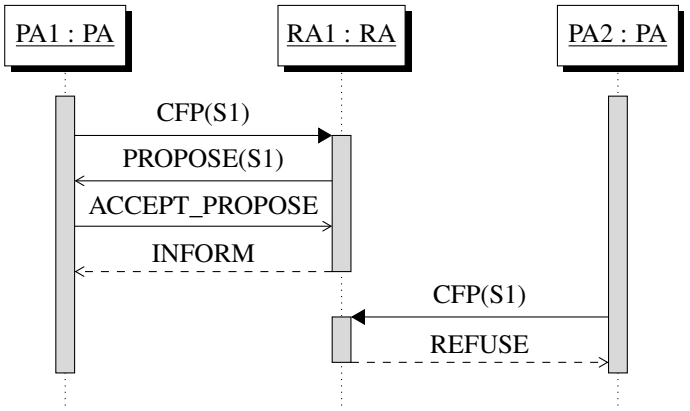

Figura 45: Requisição de um recurso por dois produtos.

almente há duas soluções: a primeira, o produto vai a uma posição segura no sistema e tenta, repetidamente, negociar aquele *skill*; a segunda, o produto faz um re-planejamento do plano de processo para executar aquele *skill* utilizando os *skills* existentes atualmente, isto é, recriar uma coalizão para aquele *skill* (S2), ou baixar de um servidor central a nova coalizão. Isto somente pode ser feito se a definição do *skill* em um formato de plano de processo estiver acessível ao agente.

A implementação atual do demonstrador para o projeto IDEAS utiliza a abordagem mais simples, que é a primeira. Ela é simples, mas igualmente flexível: se, a qualquer momento um novo recurso, que saiba executar aquele *skill* (S2) entrar no sistema, quase imediatamente o produto consegue alocar o novo recurso e prosseguir a execução de seu plano de processo.

Portanto, um produto pode entrar em execução mesmo antes que um novo módulo que execute um dos *skills* necessários para a montagem daquele produto seja efetivamente *plugado* no sistema de montagem. Isso significa que o sistema possui verdadeira *plugabilidade* e o sistema permite que um módulo mecatrônico seja adicionado e em seguida utilizado conforme a oportunidade, não sendo necessário parar ou reiniciar todo o sistema produtivo, nem tampouco há qualquer tempo de reconfiguração[4]. Isto mostra que o sistema tem capacidade de organizar a si mesmo para executar uma ação definida em alto nível (o plano de processo), isto é, possui auto-organização.

Ao se inserir um segundo produto, se tem dois produtos (PA1, PA2) negociando com os recursos a execução de *skills* (VER Figura 45). Se os dois produtos realizam planos de processos absolutamente diferentes, então nenhum conflito ocorre, entretanto, se, em um dado momento, os dois produtos necessitem do mesmo *skill*, um determinado produto (p.ex. PA2) pode conseguir acesso antes do outro (PA1), de tal sorte que o tempo relativo a execução do *skill* passa a ser maior devido a espera pela execução daquele *skill* em outro produto, o que, novamente, também não está descrito no plano de processo. Logo, a montagem do produto aparece no sistema como uma propriedade emergente deste, a partir de interações do nível micro, como uma novidade radical em relação às informações originalmente disponíveis nas partes que o formam.

Analisa-se agora o ponto de vista do recurso (p.ex. R1): neste caso, chega um produto (p.ex. PA1) que e o pede para realizar uma atividade (*skill* S1), a qual é realizada. Depois outro produto (PA2) faz outra solicitação e depois outro e assim sucessivamente. Note

[4]De fato o único tempo de espera entre a colocação do módulo no sistema e o seu uso é apenas a inicialização do agente mecatrônico.

que, entre uma ação e outra, o recurso terá um tempo não determinado de ociosidade. Mais uma vez, não há nenhum controle explícito do que está sendo feito, no ponto de vista do recurso.

Aumentando a complexidade, imagine-se dois recursos (R1 e R2) que consigam realizar o mesmo *skill* (S1). Neste caso, haverá competição entre os produtos e irá depender da negociação qual deles será escolhido por um determinado produto. Por exemplo, um produto (PA1) pode alocar um certo recurso (R2) enquanto o outro (PA2) vai para o recurso remanescente (R1) ou vice-versa. Não se sabe, *a priori* qual será executado e em que momento.

Ampliando ainda mais a análise para n produtos e m recursos, o problema do agendamento, que numa abordagem tradicional é resolvido com um algoritmo de execução exponencial, aqui é resolvido por negociação à medida e no momento em que seja necessário.

As análises até o momento foram do ponto de vista micro, isto é, apenas das relações entre os elementos do sistema. É inegável que existe uma relação hierárquica entre produtos e recursos (assim como entre CLAs e recursos), entretanto, não há nenhum algoritmo pré-definido na execução das atividades neste nível, não há programação explícita, num RA, de como montar um produto, nem há conhecimento do estado global do sistema[5], bem como não há restrições na execução de um dado *skill*, isto é, ele pode ser executado um número arbitrário de vezes sobre um mesmo produto e um número não especificado sobre produtos diferentes. Todas as decisões são tomadas apenas com informações locais ou recebidas via mensagens internas ao sistema, em geral da negociação. Também o comportamento dos produtos (e eventuais CLAs) parece ser caótico, pois não há regra explícita de qual recurso será tomado em determinado momento. Claro que isto está implícito na negociação, mas não pode ser pré-determinado a partir de uma análise reducionista do sistema.

Como nada disso estava previsto no plano de processo, aquela visão de que não há novidade no plano de processo é falsa e o sistema parece, de fato, apresentar emergência. Em uma visão global, há uma ordem crescente à medida que o produto é montado, graças às interações no nível micro. Poder-se-ia dizer que o produto "emerge" a partir das iterações no nível micro. Por outro lado, na visão local há um crescente aumento na aleatoriedade aparente das ações realizadas pelos agentes, à medida que o sistema torna-se mais e mais organizado no nível macro. Diz-se então que há um aumento da entropia (aumento na complexidade estatística e/ou computacional) no nível micro e uma diminuição da entropia (diminuição da complexidade estatística e/ou computacional) no nível macro.

O sistema assim considerado parece possuir emergência porque os produtos são fabricados (que seria o seu comportamento emergente) apesar da aparente aleatoriedade no nível micro.

Como o sistema é hierárquico em 2 camadas: a camada cognitiva, formada por CLAs e PAs, e a camada motora (executora), formada de RAs e TSAs, a arquitetura proposta possui auto-organização e emergência.

Entretanto, há uma outra experiência mental que é possível se fazer: incluir a definição do `owner` (o agente que deve executar um determinado *skill*) no plano de processo. Neste caso, o plano de processo definiria, de fato, todo o algoritmo de execução da montagem do produto. Se os produtos (PAs) fossem cuidadosamente instanciados no sistema, até mesmo o tempo de execução de cada qual poderia ser pré-determinado. De fato, um

[5]Note-se que não há essa informação nem mesmo no agente de páginas amarelas, que poderia centralizar todos os agentes e seus *skills*.

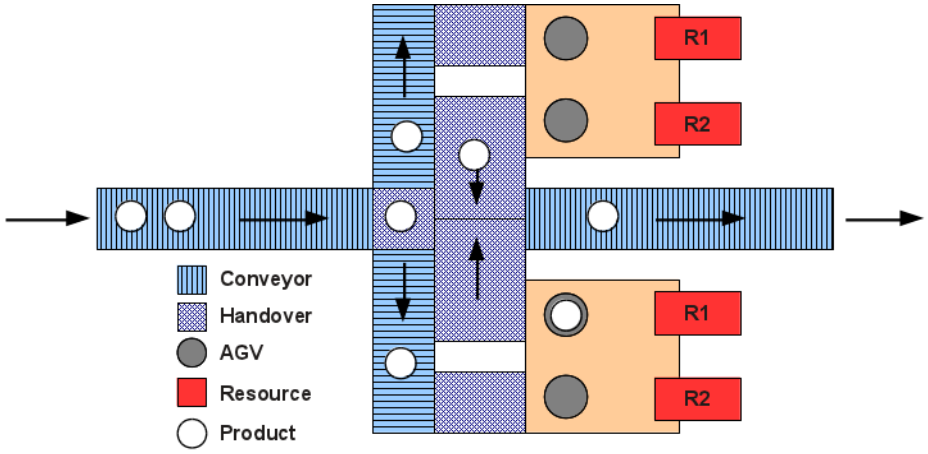

Figura 46: Sistema de montagem.

sistema assim não teria nenhum dos benefícios da auto-organização (em especial não teria *plugabilidade*), nem tampouco apresentaria emergência (não há efeito micro-macro, o sistema não é dinâmico etc.)

Por fim, uma última experiência mental seria fazer o sistema com um número de módulos mecatrônicos igual ao número de *skills* e uma equivalência um-a-um entre eles. Neste caso, mesmo com negociação, os produtos (PAs) somente poderão obter um único recurso por vez e, se os produtos forem instanciados escrupulosamente, até mesmo os tempos podem ser definidos. De fato um sistema assim teria uma performance melhor se fosse utilizada uma abordagem de programação tradicional. A única vantagem de um sistema auto-organizado seria a possibilidade de substituição de módulos em tempo de execução e nada mais (o que também se consegue via abordagem tradicional com algum esforço).

Conclusões: consoante a definição de trabalho de emergência, um sistema do tipo EAS/EPS somente apresenta propriedades emergentes se possuir livre possibilidade de negociação e redundância dos módulos disponível no sistema de montagem, sejam eles recursos, unidades do sistema de transporte, agentes de coalizão e produtos; entretanto, se o sistema não apresenta essas condições não pode ser chamado de emergente.

5.5 Especificidades do transporte

Como colocado no Capítulo 4, não há nenhuma abordagem formal já definida para o sistema transporte, então os implementadores estão livres para propor as soluções que acharem mais satisfatórias, desde que respeitadas as interfaces. Descreve-se aqui a proposta levada a efeito pelo autor dentro do IADE, como parte do projeto IDEAS.

Será utilizado, como exemplo, o sistema mostrado na Figura 46, replicada aqui por conveniência.

Na implementação padrão do IADE a associação entre os elementos do transporte e os agentes no sistema pode ser feita conforme o esquema mostrado na Figura 47. Nota-se que todos os elementos vizinhos podem comunicar entre si, isto é os agentes mecatrônicos: RA, CLA, TSA e PA.

Cabe ao próprio agente produto realizar a solicitação ao sistema de transporte para levá-lo até o recurso objetivado, sendo que as etapas intermediárias estão todas sob a sua visibilidade, se necessário. Contudo, pode deixar a carga do sistema de transporte os

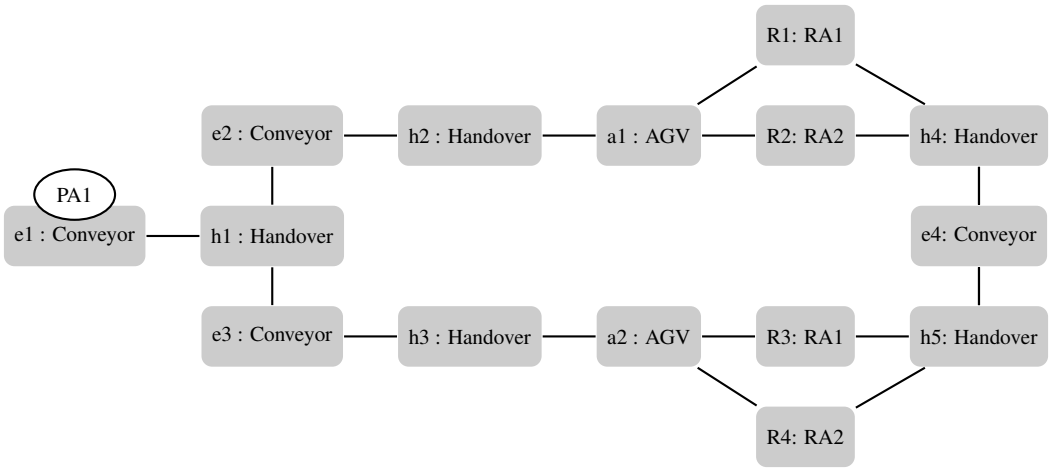

Figura 47: Um possível modelo em agentes para o exemplo da Figura 46

pontos intermediários. Entretanto, para que o produto possa acessar o custo do sistema de transporte, é necessário mensagens extras (ver Figura 48)

Em geral a preocupação com o agendamento vem da repercussão das ações do produto sobre o sistema de transporte. Por exemplo, se um produto puder verificar que uma certa esteira está sobrecarregada, pode escolher buscar outro recurso que lhe preste um serviço, em detrimento daquele primeiro caminho. Mas não há, formalmente, um algoritmo de agendamento, ou antes, ele é efetuado conforme a demanda, de acordo com as escolhas que cada agente no sistema faz em relação ao seu funcionamento.

Assim, para que um agente produto possa escolher, dentre várias alternativas, qual o recurso seja mais viável a ser utilizado, deve usar de alguma estratégia de custo associado a cada tarefa de cada recurso (*skill*), que irá executar, e ainda um custo associado ao sistema de transporte, isto é, um custo em termos de tempo que será utilizado para trafegar do ponto em que está para se chegar ao recurso alvo.

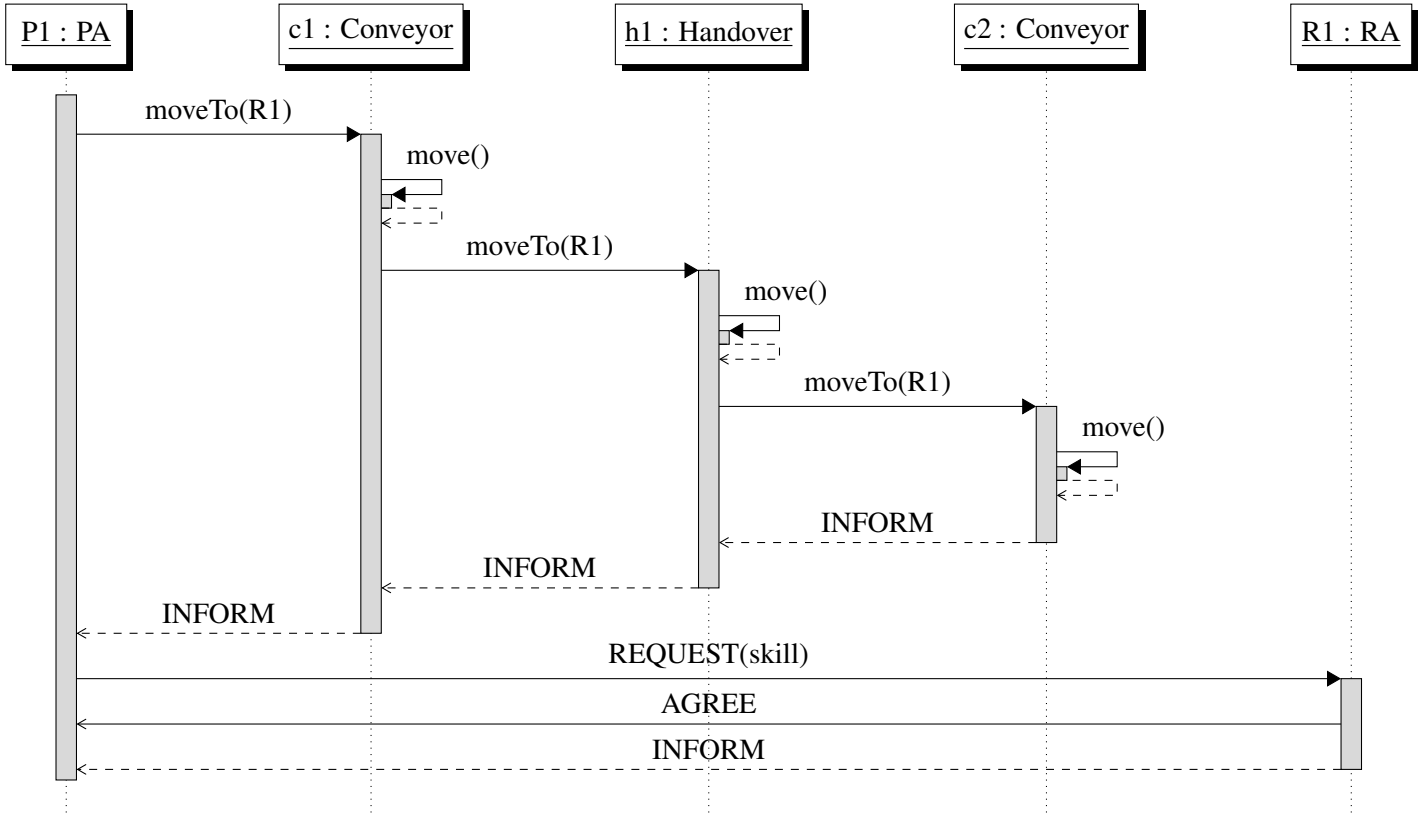

Figura 48: Mensagens pelo sistema de transporte para levar um produto até um recurso por meio de duas esteiras e um entroncamento.

Os custos relativos às tarefas dos recursos são, em geral, os relacionados ao tempo de execução da tarefa a ser solicitada, tais como o tempo médio ou o último tempo de processamento. O custo do recurso aparece quando este expõe mais de um *skill* que poderá ser usado num processo; se outro recurso não oferece os mesmos *skills*, esta é uma informação relevante, que pode influenciar a decisão por parte do agente produto, visto que pode não ser necessário uma movimentação extra.

Para os custos de transporte, em geral, são considerados o tempo de espera em linha (devido a um acúmulo de produtos em uma esteira, por exemplo) e a distância entre o ponto atual e o ponto de destino. Em AGVs, em geral, as rotas mais curtas são as preferidas, enquanto em esteiras são preferidos os tempos em fila.

A interação entre os agentes mecatrônicos e o sistema de transporte é feita através da negociação e execução de *skills*. Assim, para um produto "entrar" no sistema, deve negociar com o sistema de transporte que lhe está associado, no caso um TSA que representa a palete, a sua disponibilidade. Em geral será aceito, pois deverá haver uma palete para cada produto, isto é, a palete sempre colocará um custo zero a uma solicitação quando não ocupada, e emitirá uma mensagem FIPA *Refuse* para o caso de outro produto iniciar negociação com ela quando ocupada.

Uma vez dentro do sistema, o produto irá começar a executar o seu plano de processo e, então, irá negociar com os recursos a execução de um *skill*. A abordagem aqui é não realizar o agendamento previamente, mas decidir qual o melhor recurso a ser utilizado conforme a demanda, o que é levada a efeito durante a fase de negociação.

Além das informações dos recursos, o produto deve solicitar ao sistema de transporte informações adicionais sobre o custo de se movimentar da posição atual até cada um recurso participante da negociação. Armazenar o histórico das decisões tomadas pode melhorar a capacidade de tomada de decisão no sistema, pois esse histórico pode ser repassado aos produtos entrantes, tornando-os mais "espertos". Por outro lado, uma métrica e um decisão simples pode ser mais rapidamente conseguida, o que pode melhorar o desempenho do sistema.

A arquitetura em si admite qualquer formulação, entretanto, optou-se pela seguinte:

Para o recurso, ele mantém o tempo médio de processamento ($processingTime$) para cada *skill*. Esse valor é passado para o produto.

Para um sistema de esteiras, assume-se que cada nodo possui as informações relativas à sua capacidade, tamanho e velocidade, que são obtidas por configuração na inicialização do agente, e pode adquirir quantidade de peças na fila em um dado instante (através de um sensor externo, por exemplo). Em geral o que determina o início de uma esteira é a saída de algum outro elemento de transporte a ela associado, e o que determina o seu fim é, em geral, um ponto de parada *stoppers* antes da entrada no próximo elemento do sistema. Para os casos em que as esteiras não possuem pontos de parada, sensores que detectem a chegada de uma palete e a retiram da linha para a ação dos respectivos recursos podem ser empregados. De qualquer forma, a quantidade de elementos na fila é normalmente de conhecimento do agente da esteira. É claro que se deseja maximizar o uso da esteira, então ela deve possuir a maior quantidade possível de produtos em fila, mas sem chegar ao ponto de tornar-se um gargalo. É de notar-se a diferença de visão para um sistema tradicional, em que não há preocupação com o sistema de transporte, uma vez é o posto de trabalho à frente que deixa acumular peças na esteira. Entretanto, esta proposta usa uma abordagem multiagente, na qual cada agente é uma entidade autônoma e, portanto, responsável por si. Cada agente deve decidir, com informações locais ou adquiridas de seus vizinhos, como deve agir, então, faz sentido criar um ponto de corte, a partir do

qual, o custo relativo àquela esteira se torna proibitivo para o sistema de transporte e, abaixo do qual, a esteira ainda admite novos elementos. Todos esses parâmetros podem ser controlados remotamente.

Resumindo, para esteiras, tem-se:

Configuração:	$length$	tamanho da esteira (m)
Configuração:	$speed$	velocidade da esteira (m/s)
Configuração:	$capacity$	capacidade máxima da esteira (número de peças)
Configuração:	$fillFactorTh$	ponto de corte para o custo (% do número de peças)
Calculado:	$fillFactor$	% do número de peças presentes
Entrada:	$inventory$	quantidade efetiva na linha
Persistente:	$lastCost$	variável auxiliar
Constante:	$growing$	fator de crescimento após ponto de corte

$minTravelTime = \frac{length}{speed}$
$fillFactor = \frac{inventory}{capacity}$
se $fillFactor < fillFactorTh$
  então
    $C_t = minTravelTime * (1 + fillFactor)$
  senao
    $C_t = growing * C_{t_{ant}}$
fim-se
$lastCost = C_t$

onde C_t é o custo do transporte. Note que o custo do trasporte tem a dimensão de tempo. O $fillFactor$ é sempre um número entre zero e um. O custo do transporte deve ser proporcional ao $fillFactor$ e não pode ser menor que o tempo mínimo de tráfego pela esteira.

No produto, então, essas informações são carreadas através da mensagem de negociação e o produto pode calcular:

$$Custo = processingTime + C_T$$

Note que o custo, neste caso, é um tempo que serve de comparação para o produto. Minimizar este tempo é seu objetivo.

A fim de ilustrar o uso, mostra-se na Listagem 5.4, um fragmento do código da camada inferior de um CLA, a qual é formada por alguns comportamentos que visam a interação com recursos e outros CLAs. Neste fragmento, vê-se o momento em que todas as respostas da negociação são recebidas e uma avaliação é feita (observe a chamada `this.mA.getNegotiator().evaluateProposals(this.skToNegotiate,proposals)`). O agente mecatrônico possui em sua camada superior, uma entidade, o negociador, que é responsável pela avaliação das propostas. É nela em que os custos são comparados e as decisões tomadas.

Listagem 5.4: Negociação dentro de um CLA

```
1  public class NegotiatorInitiator extends ContractNetInitiator {
2
3      private MechatronicAgent mA;
4      private Skill skToNegotiate;
```

```
5       private boolean sucessfull = false;
6       private ACLMessage cfp;
7
8       public NegotiatorInitiator(Agent a, ACLMessage cfp, Skill
            skToNegotiate) {
9           super(a, cfp);
10          ...
11      }
12
13      ...
14
15      protected void handleAllResponses(Vector responses, Vector
            acceptances) {
16          Bag proposals = new Bag();
17          for (int i=0; i<responses.size(); i++){
18              ACLMessage temp = (ACLMessage) responses.get(i);
19              if (temp.getPerformative() == ACLMessage.PROPOSE){
20                  proposals.add(temp.getContent());
21                  ACLMessage reply = temp.createReply();
22                  reply.setPerformative(ACLMessage.REJECT_PROPOSAL);
23                  reply.setReplyByDate(new Date());
24                  acceptances.add(reply);
25              }
26          }
27
28          ...
29
30          int acceptedProposalIndex = this.mA.getNegotiator().
                evaluateProposals(this.skToNegotiate, proposals);
31          ACLMessage tempReply = (ACLMessage)acceptances.get(
                acceptedProposalIndex);
32          tempReply.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
33          Iterator receiversReply = tempReply.getAllReceiver();
34          if (receiversReply.hasNext()){
35              AID tempAID = (AID)receiversReply.next();
36              this.skToNegotiate.setOwner(tempAID.getLocalName());
37              this.sucessfull = true;
38          }
39
40          ...
41      }
42
43      ...
44  }
```

Para um AGV, o mais importante é a definição da rota. Uma forma de se fazer isso é através de um agente chamado `PathPlanning`, o qual não faz parte da arquitetura em si, mas do sistema de transporte e é encapsulado pelas interfaces de agente do TSA. Ele calcula o melhor caminho para os AGVs, e.g. a partir da distância euclidiana entre os pontos. Note que essa solução é uma solução centralizada para o sistema de transporte. Uma abordagem distribuída poderia ser conseguida através de algoritmo como o de Dijkstra (TANENBAUM, 2003), o qual permite a programação de restrições físicas, conforme o sistema em questão, para a movimentação dos carros.

5.6 Compartilhamento de informações entre *skills*

Quando da construção de um plano de processo (*workflow*), o operador frequentemente necessita compartilhar alguma informação entre as várias chamadas de *skill* dentro do *workflow*. Esta sessão mostra como é feito o compartilhamento de dados entre as execuções de *skills* num plano de processo.

Para entender isso, é preciso relembrar que os comportamentos do JADE, e por conseguinte do IADE, são executados em compartilhamento do tempo por cooperação. Uma consequência disso é que não há nenhuma necessidade da criação de semáforos ou outros

mecanismos de comunicação inter-processo (IPC) entre os comportamentos para fins de troca de dados entre eles.

Os *skills*, por sua vez, são mapeados como comportamentos para execução pelo *scheduler* do JADE. Logo, a execução de um plano processo é a execução, pelo JADE, de um conjunto de comportamentos. Cada tipo de *skill*, bem como o desenho do grafo onde este *skill* reside é transladado para um tipo de comportamento específico. Assim, para se ter a execução de *skills* em paralelo, usa-se um `ParallelBehaviour`, para execução sequencial, usa-se um `SequentialBehaviour`, para um *skill* atômico, usa-se um `FSMBehaviour` devidamente preparado para requisitar execução remota, e para um *skill* de decisão, usa-se um `FSMBehaviour` devidamente preparado para resolver uma expressão booleana.

Então, caso haja necessidade de trocar informações entre *skills* sendo executados em um plano de processo, basta acessar a variável dentro dos *skills* de maneira direta. Variáveis internas são definidas igualmente como os parâmetros, exceto que não são usadas para chamada do método nativo.

Por causa do compartilhamento por cooperação, em um dado instante, nenhum outro comportamento e, portanto, nenhum outro *skill* estará sendo executado em concorrência. Então, tem-se absoluta certeza do estado das variáveis na entrada e na saída dos comportamentos (e *skills*).

Entretanto, *skills* atômicos, os quais são executados continuamente até que o próprio método da biblioteca Java que aquele *skill* abstrai indique o término (o que é feito por meio de `AtomicSkill.executed=true`), precisam tomar cuidado em alterar variáveis, uma vez que o comportamento que controla a execução do *skill*, irá executar a chamada ao método nativo várias vezes e, entre as execuções, outros comportamentos também serão executados. Assim, é possível que uma variável possa ser alterada entre essas execuções sucessivas do método nativo.

No entanto, isso foi proposital, porque pode-se ter comportamentos que monitorem o hardware separadamente da execução dos *skills*. Assim, quando uma condição qualquer no hardware mudar, o comportamento monitor modifica o estado de uma variável na biblioteca, o que afeta o comportamento dos métodos e é bastante eficiente.

Como pode ser percebido, tais situações são específicas da implementação do IADE e não fazem parte da proposta original.

No IADE, foi criado um mecanismo de troca de dados durante a execução de *skills*, através de tabelas de mapeamento de variáveis e valores. Estas tabelas permitem o acesso, pelos *skills*, a todas as variáveis globalmente definidas durante o projeto do plano de processo. Ela é acessível durante a execução de *skills* graças às referências é passadas quando da execução de cada *skill* do plano de processo. Estas tabelas estão mostradas na Figura 49.

Em execuções de métodos nativos não há grande uso para tal funcionalidade, porque o compartilhamento de variáveis pode ser feito diretamente na biblioteca de E/S ou através do agente. Em execuções de processo, todavia, há inúmeros usos, servindo de base para tomada de decisão em um *workflow*.

Ao final de cada execução dos *skills*, as tabelas são mantidas consistentes, atualizando os valores globais a partir dos valores de retorno das variáveis locais. Dessa forma, posteriores usos dessa variável já contarão com valores atualizados, podendo o *skill* fazer uso dela e tomar decisões.

Essa questão do compartilhamento de variáveis ajuda na questão temporal, porque não há interrupção no processamento, não exigindo nada além do acesso direto às variáveis.

mappingVars

local	global	type
Sk1.x	x	IN
Sk1.y	y	IN
Sk1.z	z	IN
Sk1.result	out	OUT
Sk2.prior	out	IN
Sk2.x	x	IN
Sk2.y	y	IN

mappingValues

variable	values
x	100
y	221
z	25
out	false

Figura 49: Mapeamento de variáveis para um *skill* `Sk1` e *skill* `Sk2`.

6 RESULTADOS

Uma vez implementada a proposta através do IADE e os demonstradores do projeto IDEAS, os quais são soluções baseadas em agentes para a manufatura, alguns resultados são obtidos em experimentos em laboratório. Esta sessão descreve estes experimentos, os seus resultados e mostra que a arquitetura apresenta-se conceitualmente válida para os casos propostos.

6.1 Execução em uma célula Festo

A primeira execução do IADE foi feita em uma célula Festo, a qual é mostrada na Figura 50 e, esquematicamente na Figura 51.

Esta célula é formada por três eixos com *grippers* (`G1`, `G2` e `G3`), dois eixo rotacionais (`R1` e `R2`), dois eixos para translação (`T1` e `T2`). Estes dispositivos estão distribuídos em três partes bem distintas do sistema e trilhos levam o "produto" de um parte a outra. No diagrama da Figura 51, o elemento rotacional `R2` e o de translação `T2` estão acoplados ao eixo `G3`, o que permite que o mesmo se desloque em vários eixos. O produto é representado por uma bola de gude, de tal forma que o agente produto associado a uma determinada bola é executado em um PC ao lado da célula. Cada dispositivo conta com o seu próprio controlador, nos quais os agentes de recursos são executados. São módulos ARM7 de 200MHz, onde roda uma máquina virtual Java ME Personal Profile 1.0, JADE 4.0.1 e o IADE.

Em cada uma das três partes, os dispositivos são mais ou menos integrados para realizar uma atividade.

Na primeira parte, o `G1` que é um eixo vertical (pode realizar movimento para cima e para baixo) e, em sua ponta, há uma garra (que permite o movimento de abrir e fechar). A função deste arranjo é pegar uma bola do `R1` (que permite movimento rotacional no sentido horário e anti-horário) e o depositá-la num trilho que, por gravidade a leva para a segunda parte.

Na segunda parte, o `G2`, que é um eixo vertical (movimento para cima e para baixo), possui um sugador na ponta e está associado com outro eixo que permite um movimento para direita e esquerda na célula. A função destes dispositivos é pegar a bola do trilho de entrada e levá-la para o trilho de saída, o qual é um dispositivo de translação (`T1`) que permite dois movimentos, para dentro e para fora.

Por fim, na última parte o `G3`, que é um eixo com motor linear e uma garra na ponta, permite pegar a bola no `T1` e levá-la para até 4 posições distintas.

A Figura 52 mostra a primeira parte da célula real. Note as bolas de gude como produtos.

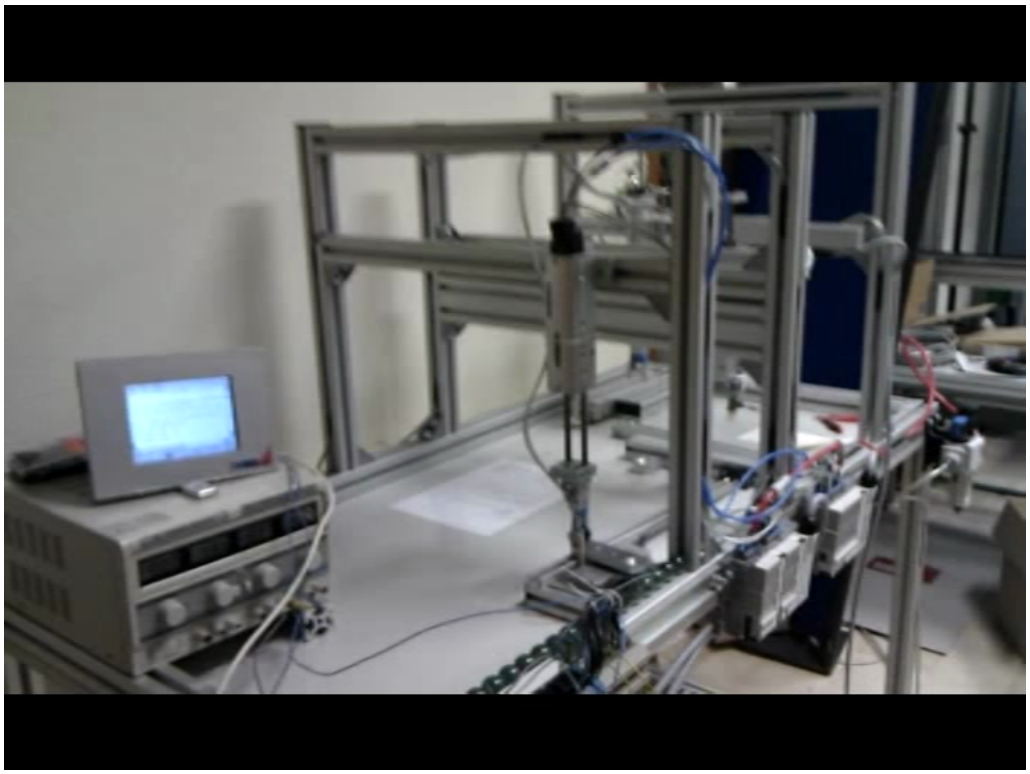

Figura 50: Instantâneo da célula Festo em funcionamento.

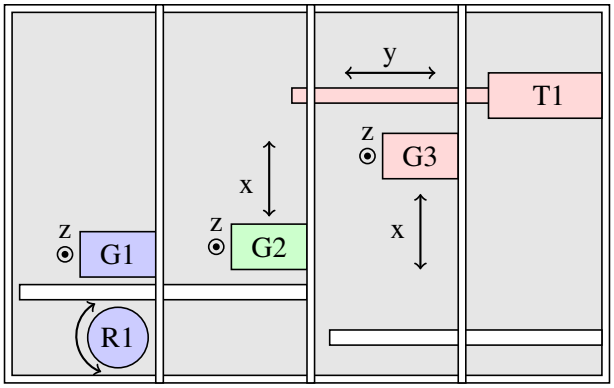

Figura 51: Vista superior esquemática de uma célula Festo.

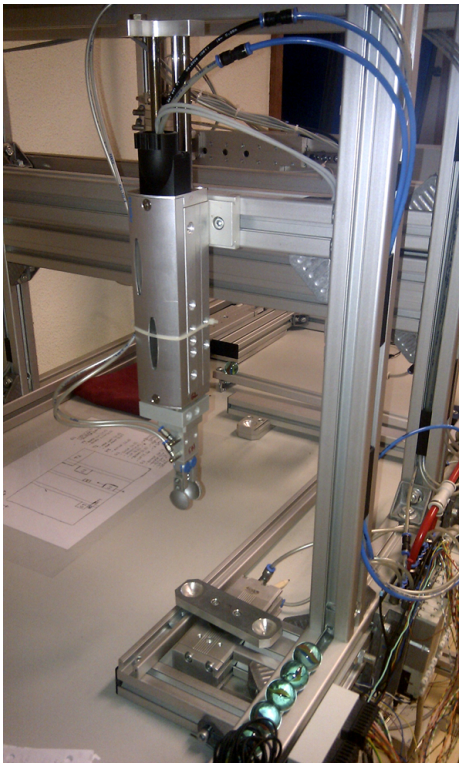

Figura 52: Primeira parte da célula Festo para o desenvolvimento do IADE.

O objetivo deste experimento é testar o desempenho do sistema em níveis diferentes de granularidade de *skills*. Para isso foram criados três agentes de recursos, um para cada área. Cada agente implementa um conjunto de *skills* com níveis de granularidade diferente.

Exemplo: para uma operação *pick & place* no primeiro eixo, pode-se descrever o plano de processo detalhadamente, como mostrado na Listagem 6.1.

Listagem 6.1: Pick & Place com granularidade fina

```
1       rotateccw(2000)
2       gripperOpen(1000)
3       axisDown(5000)
4       gripperClose(1000)
5       axisUp(5000)
6       rotatecw(2000)
7       axisDown(5000)
8       gripperOpen(1000)
9       axisUp(5000)
```

O parâmetro numérico dos *skills* na listagem é o seu *timeout*. Um processo deste tipo necessita realizar negociação para cada um dos *skills*.

Por outro lado, se a biblioteca possuir um *skill* `pick()` e um outro `place()`, um processo mais simples e mais eficiente pode ser feito, como o mostrado na Listagem 6.2.

Listagem 6.2: Pick & place com granularidade grossa

```
1       rotateccw(2000)
2       pick(5000)
3       rotatecw(2000)
4       place(5000)
```

Com isso é possível se estudar, neste sistema, a modificação da granularidade do *skills* e verificar o impacto no desempenho. Também é possível fazer observações das mensagens e interações entre os agentes do sistema, problemas relacionados à execução concorrente de produtos com ou sem controle das filas, etc.

Os resultados do experimento foram os seguintes:

- Não houve programação direta de agentes, apenas programação das bibliotecas de acesso ao hardware e a configurações dos *skills* dos agentes. O mesmo código foi usado tanto neste experimento, quanto no pré-demonstrador.
- Quanto mais fina é granularidade do sistema, isto é, quanto mais detalhes são descritos no plano de processo, menor o desempenho conseguido.
- O item anterior é facilmente explicado pela quantidade de mensagens necessárias para a realização das negociações e execuções. Quanto mais grossa a granularidade, menos detalhes do sistema são aparentes e menos mensagens são trocadas.
- O desempenho foi bem inferior ao de um sistema tradicional.
- O sistema é flexível e admite mudanças nos planos de processo como operação normal.

6.2 Pré-demonstrador do projeto IDEAS

O IADE foi criado, em sua primeira versão, para ser a base do desenvolvimento do primeiro demonstrador do projeto IDEAS, chamado *MiniProd*. O primeiro demonstrador é um sistema de montagem flexível, formado por uma mesa onde dois carros podem trafegar, uma unidade *stacker*, que funciona simultaneamente como alimentador e como receptor de produtos, em duas filas, uma de entrada e outra de saída, e cinco posições onde podem ser encaixados módulos padronizados. Um instantâneo do sistema em funcionamento pode ser visto na Figura 53, e um esquemático desse funcionamento visto na Figura 54.

Há uma limitação física para o movimento possível dos carros, devido aos seus cabos de alimentação que não podem se cruzar. A regra para se evitar este cruzamento é: o carro C_1 não pode cruzar a vertical que passa pelo centro do carro C_2 e o carro C_2 não pode cruzar a vertical que passa pelo centro do carro C_1, conforme a referência de vertical da Figura 54.

Outro problema com este tipo de sistema é a possibilidade de colisão, a qual deve ser evitada. Como os carros podem mover-se simultaneamente, todo o caminho a ser trafegado por um carro deve ser mantido ocupado, fazendo com que todos os pontos que lhe formam não fiquem mais disponíveis para um outro carro.

Como os agentes de transporte que modelam os carros somente possuem informação sobre a sua posição e não a de outros agentes, há duas abordagens que podem ser seguidas para se obter a informação faltante, a saber, centralizada ou distribuída.

Na abordagem centralizada, um agente especial (chamado `PathPlanner`) armazena todas as informações do sistema de transporte, nomeadamente a posição dos recursos, da `stacker` e a posição corrente dos carros, informações estas que são utilizadas na geração dos caminhos. A abordagem é simples, porém insere um único ponto de falha no sistema, o `PathPlanner`.

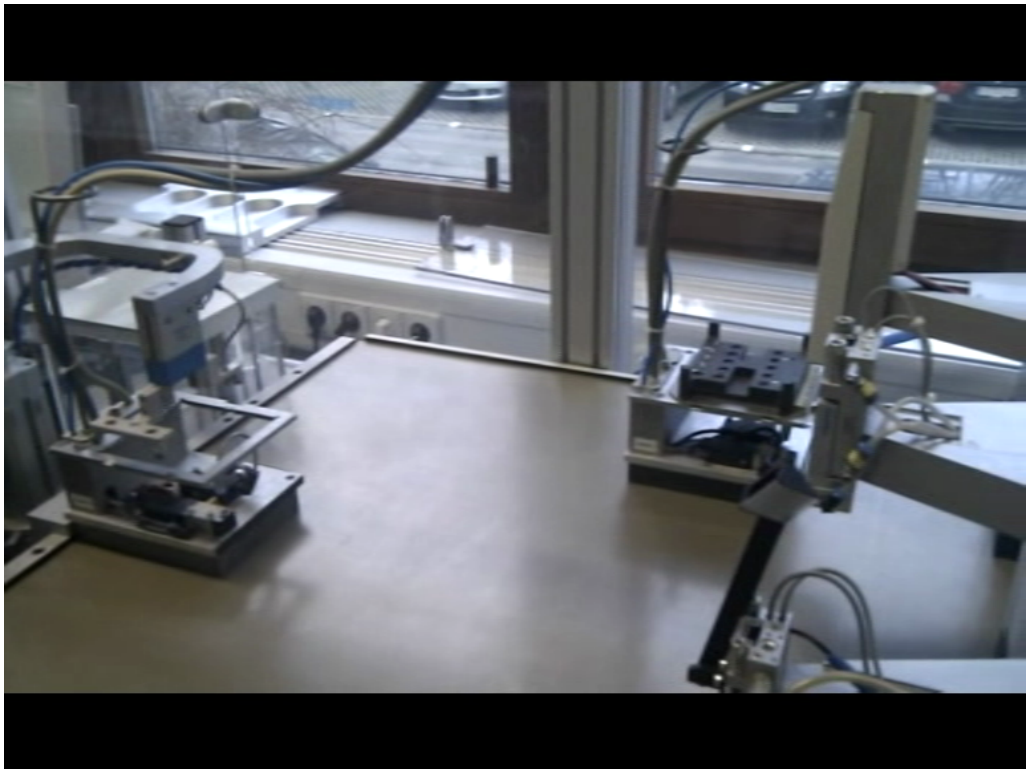

Figura 53: Pré-demonstrador para o IDEAS em ação. Note um carro pronto para receber um novo produto enquanto outro já está em plena execução (no canto superior direito).

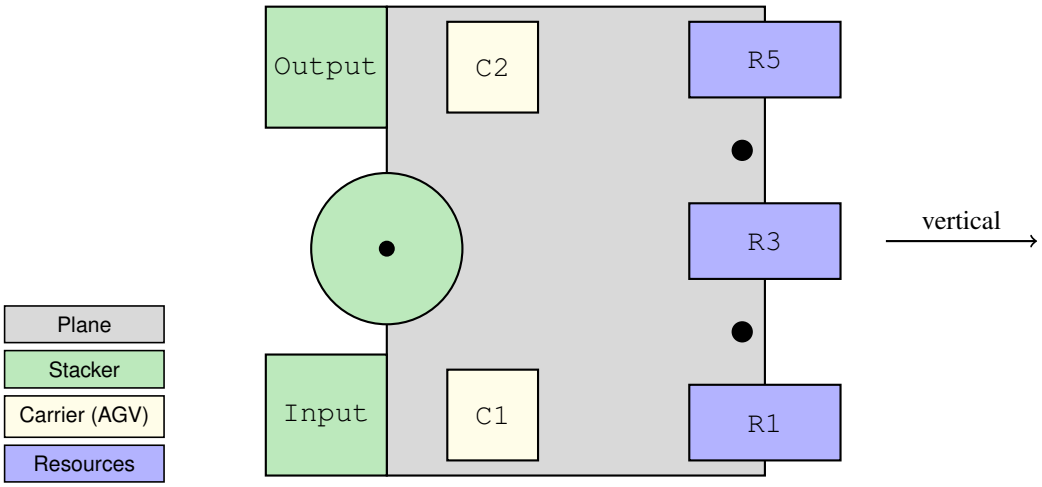

Figura 54: Diagrama esquemático do pre-demonstrador do projeto IDEAS.

O funcionamento é da seguinte forma: quando uma produto precisa de uma rota, ele solicita-a ao `PathPlanner`, o qual lhe fornece uma rota livre de colisões. O cálculo da regra para o cruzamento dos cabos de alimentação dos carros já será cumprida para a geração da rota. Esta foi a abordagem utilizada no pré-demonstrador.

Na abordagem distribuída, as informação são passadas agente a agente. Então o sistema de transporte implementa um algoritmo de melhor caminho, como o Dijkstra (TANENBAUM, 2003), no qual pode-se programar todas as restrições físicas necessárias de forma distribuída.

Esta é a abordagem para o demonstrador final, a qual é bem mais interessante em um sistema de esteiras, por naturalmente formar um rede.

O `PathPlanner` é parte do sistema de transporte, mas é completamente transparente para o restante do sistema. Do ponto de vista dos produtos em produção, não há diferença em se usar uma ou outra abordagem. O `PathPlanner`, portanto, não faz parte da plataforma, sendo um agente desenvolvido à parte, somente para este caso.

A Figura 54 também mostra os carros em suas posições originais, ou de referência. Essa posição inicial também é a posição segura, para onde o carro desloca-se caso haja qualquer problema na movimentação entre módulos, em especial *deadlocks*.

O funcionamento do sistema, percebido por um observado, é o seguinte:

- na pilha de entrada são colocados paletes com a base dos produtos a serem montados;
- um botão é acionado e a unidade *stacker* faz o seu procedimento de iniciação, bem como os carros também realizam um procedimento de autoreferenciação, para garantir que estão em suas posições originais;
- um dos carros (C_1 ou C_2) é, então, escolhido, e este se desloca para o ponto de retirada da *stacker*;
- a unidade *stacker* retira, então, da pilha de entrada uma palete e a coloca no carro;
- o carro, então, se desloca a um dos recursos (módulos mecatrônicos posicionados do lado oposto à unidade *stacker*, de um a cinco elementos);
- o módulo executa as suas atividades, e o carro move-se para o módulo seguinte;
- uma vez o produto montado, isto é, todas as atividades de montagem concluídas, o carro volta ao ponto de retirada da *stacker*;
- a unidade *stacker* é acionada, retirando-se o produto montado e colocando-o na pilha de saída;
- por fim, o carro volta ao seu ponto de origem.

Uma vez que um carro tenha sido selecionado, o outro torna-se disponível, de tal forma que o sistema pode produzir dois produtos (diferentes) simultaneamente.

Do ponto de vista dos agentes, o processo acima pode ser descrito como:

- os agentes de recursos, do sistema de transporte (dois para os carros e um para o `PathPlanner`) e da *stacker* são iniciados;

- o `PathPlanner`) recebe a localização dos carros (suas posições iniciais), da *stacker* e dos recursos;
- produtos começam a ser criados e a executar o seu plano de processo, realizando as seguintes etapas:
  - o produto aloca a *stacker*, o que somente é conseguido quando o produto estiver no topo da pilha de entrada;
  - uma vez alocada a *stacker*, segue a negociação de um carro livre: se os dois carros estiverem livres, o produto escolhe o primeiro, se não escolhe o único livre; se nenhum estiver livre, tenta negociar um carro até que um esteja livre[1];
  - uma vez obtido um carro, solicita a sua movimentação para o ponto de referência da *stacker*;
  - depois, solicita do agente *stacker* a retirada do produto da pilha de entrada e sua colocação no carro. A posição de partida de todo produto é, portanto, o ponto de retirada (ponto de referência) do *stacker*;
  - a partir daí o agente produto inicia a execução do processo de montagem propriamente dito:
    - realiza negociação de um *skill* com os recursos de interesse;
    - escolhe o melhor recurso (o agente produto escolhe o recurso que está na menor distância da sua posição atual, que nesta aplicação é a melhor rota; outras abordagens são possíveis);
    - solicita, do sistema de transporte (o carro, neste caso), a sua movimentação para o agente escolhido;
    - solicita a execução daquele *skill*;
  - repete o processo até acabar o seu processo de montagem;
  - o último item do processo é a retirada, pelo *stacker* da palete com o produto montado do carro e a sua colocação na pilha de saída.
- estas etapas são repetidas até que a pilha de entrada da *stacker* esteja vazia e não haja mais produtos sobre a mesa sendo processados, então a unidade *stacker* se abre e aguarda ação do operador que, em geral, remove os produtos montados da pilha de saída e recarrega a pilha de entrada com novas paletes, e o ciclo reinicia.

Devido a existência de vários produtos à espera na pilha de entrada, há uma disputa constante entre os agentes produto pelo agente *stacker*. Tendo um carro disponível o agente `stacker` libera o próximo produto, ou seja, o agente *stacker* contem uma pilha (em software) de agentes produto, a qual deve acompanhar o posicionamento físico destes produtos na pilha de entrada da *stacker*.

No primeiro demonstrador, esse gerenciamento é feito através de uma ferramenta visual em que os operadores, manualmente, gerenciam a fila de agentes produtos concomitantemente com o aspecto físico na pilha de entrada da unidade *stacker*. No demonstrador final, será utilizada uma tecnologia de identificação para automatizar este processo.

[1] Note-se que não há problema em "segurar" a *stacker* uma vez que é o único produto capaz de utilizá-la no momento.

6.2.1 Experimentos realizados no pré-demonstrador

Os objetivos do pré-demostrador são: a partir de um número arbitrário de produtos diferente a serem montados, colocados na pilha de entrada, o sistema deve garantir que o os mesmos sejam montados corretamente, a partir dos planos de processo de cada produto, e que os módulos dos recursos possa ser inseridos ou removidos do sistema como parte integrante do seu funcionamento.

A Figura 53 mostra um instantâneo da apresentação oficial do pré-demonstrador nas instalações da indústria Festo AG & Co.

Foram feitos dois experimentos: o primeiro é a execução de processos diferentes, com os carros funcionando concorrentemente; o segundo é o teste de plugar-e-produzir.

Com os carros em execução concorrente, o sistema, graças ao mecanismo de auto-organização, realiza as atividades dos respectivos processos produtivos, sendo a execução do processo de um produto independente do de outro.

Uma observação que foi feita: quando da possibilidade de colisão, bem como a existência de um recurso já ocupado, é visível que o processo parece "congelar"; isso acontece porque o `PathPlanner` só consegue alocar um caminho livre para um carro e não consegue alocar um caminho livre para o segundo carro, fazendo com que este fica a espera do outro. Isto dá a sensação ao observador que um carro está a espera da passagem do outro carro, ou a liberação do tal recurso. É este último caso que pode gerar um *deadlock* no sistema. A Seção 6.2.2 discute este problema em particular.

No teste de plugar-e-produzir, um único produto é iniciado no sistema, contudo um dos módulos de recursos está desligado (simulando uma situação de falha). Quando o produto alcança, em seu plano de processo, o passo para a execução de um *skill* daquele recurso (e não há outro recurso que ofereça aquele mesmo *skill*), o processo "congela", aguardando a entrada do módulo em produção. Quando da inserção do módulo em uma nova posição, após a sua inicialização, o produto retoma a execução de seu processo e termina a sua execução normalmente.

Caberia aqui, em um parêntesis, a descrição da reação de espanto do engenheiro da Festo ao ver isto acontecendo. Apesar de ele fazer parte da equipe e esperar que tal acontecesse, para uma pessoa acostumada a descrever processos em PLCs, é contra-intuitivo ter o processo fora do módulo que efetivamente realiza a ação de montagem.

Note-se que o código da plataforma utilizado neste experimento foi o mesmo do laboratório, exceto pelas bibliotecas de acesso ao hardware. Outra personalização específica para esta execução foi o `PathPlanner`.

O módulo de execução nos recursos foi o mesmo ARM7, contudo, controlando um hardware completamente diferente. O controle dos carros, todavia, foi realizado através de um PLC virtual, executando em um PC.

6.2.2 Problema de *deadlock* no pré-demonstrador

Sistemas como o pré-demonstrador do projeto IDEAS, no qual as alocações são feitas um passo a frente, tem um problema inerente de *deadlock* (bloqueio). Este problema acontece quando os dois carros estão ocupando, cada um, um recurso que deve ser requerido pelo outro para a continuidade da execução do seu processo.

A Figura 55 mostra um possível bloqueio no pré-demonstrador. Se o `Produto 1` quiser alocar o `Recurso 2`, mas já está na posição do `Recurso 1`, e o `Produto 2`, que já está na posição do `Recurso 2`, quiser alocar o `Recurso 1`, ocorre uma situação de *deadlock*.

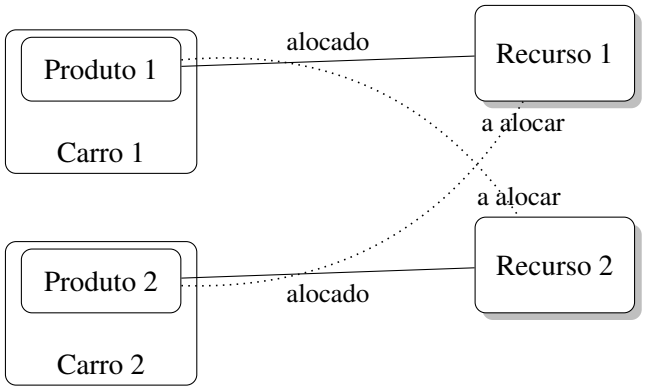

Figura 55: Possível *deadlock* no pré-demonstrador.

Entretanto tal problema é relativamente simples de ser resolvido: sempre que uma alocação for requerida e não for conseguida, antes de uma nova tentativa, o produto solicita sua ida para o ponto de referência do carro em questão. Como não é possível o cruzamento pelas verticais (conforme orientação da Figura 54), sempre haverá um caminho livre.

Uma vez na posição segura, o produto volta a solicitar o novo recurso, o que eventualmente irá conseguir, porque já não estando na posição do recurso previamente alocado, e agora livre, o outro produto consegue alocar um caminho para aquele recurso recém livre, liberando então o seu. Note que talvez nem seja necessário o deslocamento de ambos os carros para o ponto seguro, mas somente um já resolve o problema.

Apesar de o sistema ter sido executado muitas vezes, e mesmo uma situação de *deadlock* ter sido programada nos processos produtivos de dois agentes, a própria dinâmica temporal da negociação parece ter impedido tal ocorrência, pois apenas em uma situação os carros pareceram estar em bloqueio, antes de prosseguir os seus processos. Na ocasião não foi possível determinar se tal ocorreu devido especificamente a um possível bloqueio que foi evitado ou a qualquer outro erro no processo de alocação. Como o sistema apresenta-se inerentemente robusto, a recuperação da condição de erro (eventual) foi realizada sem nenhum transtorno aparente ao funcionamento do sistema.

6.2.3 Resumo dos experimentos no pré-demonstrador

Foram criados os seguintes agentes:

`C1` e `C2` - representam os carros (AGVs).

`R1`, `R3` e `R5` - representam os recursos. Foram desenvolvidas as bibliotecas de acesso ao hardware e geradas as configurações dos agentes (lista de *skills*).

`Stacker` - representa o módulo das filas de entrada e saída de produtos.

`P1`, `P3`, ... - diversos produtos.

`PathPlanner` - agente especialmente desenvolvido para esta aplicação; centraliza as informações do sistema de transporte e gerencia a criação de caminhos.

O objetivo do experimento é verificar a auto-organização, através do funcionamento concorrente dos dois carros, em função da entrada de produtos diferentes, portanto diferentes processos produtivos, e da plugabilidade, através da remoção e inserção de recursos como parte do funcionamento normal do sistema.

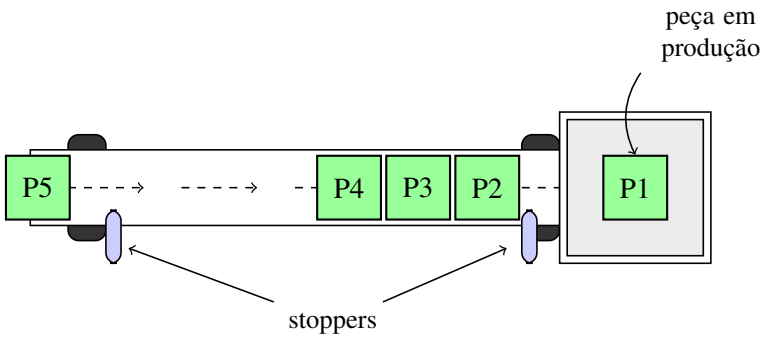

Figura 56: Sistema de esteira para a simulação.

Foram realizadas dois experimentos básicos: produtos com processos distintos foram colocados em produção, e foi realizada uma simulação de falha e recuperação.

Pode-se elencar alguns dos resultados:

- O sistema apresentou auto-organização: à medida que novos produtos, portanto novos processos, entravam em produção eles passaram a concorrer pelos recursos existentes.
- Não houve *deadlocks*.
- O sistema automaticamente recupera-se de falhas nos recursos, realizando o plugar e produzir.
- As especificidades do sistema de transporte foram escondidas através da interface proposta.
- Foi utilizado o mesmo código desenvolvido para a célula Festo, mostrada na seção anterior, exceto pelas bibliotecas de acesso ao hardware e as configurações dos agentes que são diferentes de aplicação a aplicação.

Apesar de ter sido criado apenas com o intuito de ser uma prova de conceito para a plataforma, o pré-demonstrador apresentou bom desempenho qualitativo, comparativamente com um sistema de deslocamento gerado diretamente no PLC controlador dos carros. Isso devido o fato de estar sendo usado apenas um percentual pequeno da velocidade de movimentação dos carros, de forma que não havia muita diferença entre uma movimentação executada pelo sistema de agentes ou diretamente programado no PLC. Entretanto, foram detectadas muitas repetições de mensagens quando da alocação dos recursos, o que mostra um ponto de possível melhoria para o sistema.

6.3 Simulação de uma esteira

O projeto IDEAS visa a construir um demonstrador final que utiliza tanto a abordagem de AGVs quanto de esteiras, no entanto, ainda não existe fisicamente tal sistema, o que nos deixou sem uma possibilidade de estudar o funcionamento das propostas em um sistema físico. Todavia, foram realizadas algumas simulações no computador para a formulação proposta de um sistema de esteira.

O sistema é mostrado esquematicamente na Figura 56. Note-se, a existência de 3 produtos em espera, enquanto um está em produção e outro está a adentrar o sistema.

Neste exemplo, formado por uma esteira e um recurso com um *skill*, são assumidos os seguintes valores para a simulação:

Esteira:
$length = 5$; (m)
$speed = 1$; (m/s)
$capacity = 10$; (peças)
$fillfactor_{Th} = 0.7$; (7 peças)
$inventory = 0..capacity$;

Recurso:
$processingTime = 5..10$ (s)

O custo do transporte é calculado de tal forma que aumenta quase linearmente a medida que vão chegando mais e mais produtos na esteira, até o ponto em que há um corte e o custo aumenta abruptamente, forçando os produtos a não mais escolherem aquela esteira, como destino.

O custo total é o custo de transporte mais o custo de processamento.

Para observar este efeito, foram elaborados dois gráficos: o primeiro é o do custo total em função da quantidade de peças em espera na esteira (VER Figura 57); o segundo é o custo total em função tempo de processamento de um *skill* (VER Figura 58). Os gráficos mostram várias ocorrências de um e outro caso para as variações da outra variável.

No primeiro caso é simples perceber que a partir de um determinado ponto o custo do transporte cresce muito rapidamente. No segundo caso, o crescimento do custo é sempre linear mas, a partir de uma certa quantidade de produtos na esteira, a declinação modifica-se, aumentando-se o custo total acentuadamente.

A plataforma em si permite diversos tipos de métricas e formulações para o custo do transporte. O experimento aqui descrito foi executado em um PC Intel© Core™ i5 CPU M 460 @ 2.53GHz $\times 4$, com Ubuntu Linux 11.10 - 64-bits O experimento é apenas uma mostra simples de como inserir uma dada formulação, em caso de um sistema de transporte por esteiras, no sistema.

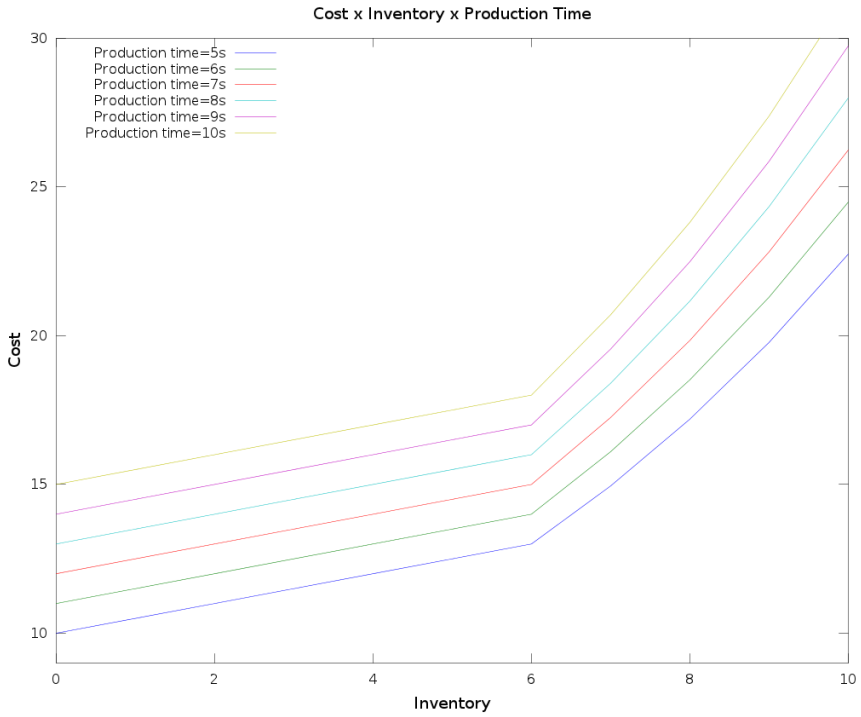

Figura 57: Custo de transporte em função da quantidade de peças na esteira.

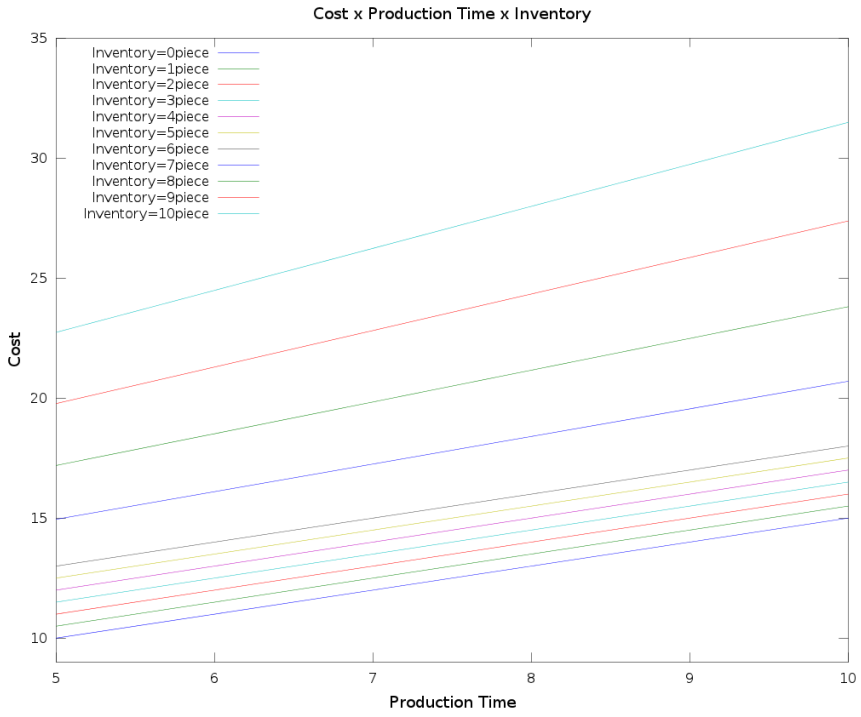

Figura 58: Custo de transporte em função do tempo de processamento.

7 CONCLUSÃO

A tendência da manufatura atual é para a personalização dos produtos, então, os cenários de automação estão cada vez mais dinâmicos, exigindo uma adequação contínua dos sistemas produtivos. Para alcançar os desafios que se apresentam, diversos paradigmas tem sido propostos. Recentemente, o surgimento de abordagens baseadas em agentes e o conceito de sistemas evolutivos tem permitido o desenvolvimento de sistemas que apresentam auto-organização, auto-otimização, autoadaptação, etc., isto é, a característica básica destes sistemas é a autonomia.

Este trabalho pretende contribuir neste degrau do desenvolvimento tecnológico, mostrando uma proposta em inteligência artificial distribuída (*Distributed Artificial Intelligence* DAI, em inglês), baseada em agentes, para permitir a customização em um sistema de montagem.

Faz uso dos conceitos de auto-otimização e auto-organização, resumidos como a capacidade de o sistema, simultaneamente, buscar uma ordem e que esta ordem seja considerada melhor que outra, segundo algum critério.

Foi proposta uma arquitetura baseada em dois tipos básicos de agentes: um cognitivo e um motor, de forma que o sistema passa a ter uma característica híbrida, isto é, possui tanto características reativas quanto deliberativas. Agentes reativos, quando postos de forma colaborativa, tornam o sistema apto a reagir a mudanças na demanda. Por outro lado, agentes deliberativos possuem a capacidade de análise do meio e uma maior capacidade de discernimento em quando e como agir. Enquanto um reage, o outro age. Um modelo híbrido tenta obter o melhor benefício destes dois mundos, sem contudo deter-se em suas respectivas fraquezas: em princípio reage, mas a forma como o faz varia conforme uma cognição que faz sobre o meio.

Uma plataforma, baseada nesta arquitetura, foi concebida especificamente para o desenvolvimento de sistemas de manufatura, ou seja, possui um foco em uma ação física.

A partir dessa plataforma foram realizados vários experimentos, no âmbito de um projeto europeu de sistema evolutivos, e foi demonstrado que um sistema de manufatura multiagente possui as características de auto-organização, mas também com capacidade de auto-otimização.

A auto-organização toma lugar quando os agentes reagem às mensagens externas, emitidas por outros agentes. Tais mensagens facultam ao sistema a negociação e a execução de funcionalidades.

Então, um sistema de manufatura assim desenvolvido possui efetivamente capacidade de plugar-e-produzir, isto é, o sistema é formado de uma miríade de módulos, os quais podem ser inseridos ou removidos como parte de seu funcionamento normal, pois que o processo de negociação é capaz de lidar com esta flutuação no sistema.

A auto-organização, contudo, não é totalmente espontânea, se tomado um ponto de vista particular do sistema: ela surge direcionada por um plano de processo. Entretanto, ela pode ser considerada espontânea sob o aspecto global, pois não há nenhuma previsibilidade em como a organização será efetuada. Assim considerado, o sistema apresenta emergência.

Da mesma forma, pode-se incluir elementos de otimização no sistema, de tal forma que ele se organize, não de uma maneira qualquer, mas de algumas maneiras que minimizem (ou maximizem) algum critério. De fato, na abordagem proposta, os desenvolvedores podem programar praticamente qualquer tipo de algoritmo de otimização, entretanto, os melhores serão aqueles de natureza distribuída e em que o processo de otimização se realiza interativamente.

Por exemplo, em um sistema de manufatura, ao se levar em conta a carga do sistema de transporte, pode interativamente escolher as melhores rotas (as que estejam menos sobrecarregadas), utilizando os mesmos mecanismos que engendram a resposta de auto-organização, isto é, os mecanismos de negociação, para a escolha dos melhores agentes a executar a tarefa negociada. A escolha é direcionado por uma métrica definida para o sistema de transporte. Dessa forma, evita-se também a execução repetida de algoritmos de agendamento, que sempre requerem uma certa capacidade de processamento e tomam tempo.

Tal exemplo de auto-otimização foi levado a efeito em um sistema real, que permitia a escolha da melhor rota no sistema de transporte baseado em AGVs. Também foi proposta uma solução para um sistema de transporte baseado em esteiras.

Como uma prova de conceito, o desenvolvimento da plataforma IADE foi realizado, e mostrou-se que a mesma plataforma pode ser utilizada em duas aplicações distintas de um ambiente industrial, sendo necessário somente a programação das bibliotecas específicas de acesso ao hardware de tais aplicações, bem como a criação das configurações dos agentes. Ou seja, trocou-se codificação por configuração, a qual é mais simples de ser realizada, a medida que novas ferramentas sejam criadas para isto.

7.1 Trabalhos futuros

A plataforma implementada apresenta vários tipos diferentes de agentes mecatrônicos para a realização de algumas funções especializadas. Assim tem-se: PAs para produtos, CLAs para módulos complexos, RAs para recursos e TSAs para o sistema de transporte.

Abstraindo-se o acesso ao hardware, o funcionamento de um CLA e de um RA parecem similares, exceto pelos motores de execução de suas funcionalidades.

Então, se um agente puder interpretar uma linguagem qualquer para o chão-de-fábrica, e possuir acesso ao hardware, este poderá ser usado tanto como um CLA quanto como um RA. Tal mecanismo poderia ser inserido na arquitetura, criando-se um mecanismo de diferenciação, na camada inferior de um agente mecatrônico, de uma chamada de procedimento nativo, da execução de um processo. O impacto disso em relação ao esforço de programação e configuração de agentes ainda precisa ser verificado.

Além disso, há várias questões de desempenho que também devem ser endereçadas, tais como quais os critérios que permitem a maximização da resposta de auto-organização do sistema, sem contudo penalizar a performance. Também pode-se questionar qual o impacto no desempenho da granularidade no sistema, e como reduzir o impacto da comunicação, a base para a negociação e auto-organização, no desempenho.

Do ponto de vista da implementação, uma alternativa é o uso de plataformas computacionais de melhor desempenho e a codificação em uma linguagem de programação que possa ser compilada diretamente em código nativo, reduzindo-se o custo da camada extra de software que uma máquina virtual Java possa inserir. Dados os avanços contínuos no hardware e na maximização do desempenho das máquinas virtuais, tal re-codificação teria que ser bem avaliada, se haveria ganho real ou não.

Por outro lado, uma re-codificação com vistas a permitir o desenvolvimento de aplicações de tempo real duro (*hard real time*) é de interesse dos parceiros. Uma avaliação das técnicas a serem usadas para alcançar tais requisitos são necessários.

7.2 Alcance da tecnologia

É muito difícil prever o desenvolvimento tecnológico no longo prazo, pois frequentemente tecnologias disruptivas, e movimentos sociais surgem, com a consequente alteração das necessidades e da paisagem tecnológica.

Contudo, pode-se perceber uma tendência para uma mudança no paradigma da criação das aplicações industriais: ao invés de o programador realizar a programação do processo produtivo diretamente nos módulos do sistema produtivo, o programador irá programar o processo produtivo, na forma de objetivos de alto nível, em módulos específicos do sistema, o qual se ajustará para a realização daqueles objetivos.

A forma como será feita esta programação (ou melhor, esta configuração do sistema) será determinada consoante o paradigma utilizado: se usar sistemas holônicos, será na forma de *holons*, se sistemas evolutivos, na forma de agentes. A proposta apresentada usa o conceito de sistemas evolutivos e de agentes.

Em resumo a proposta apresentada realiza o seguinte, mostrado nas figuras a seguir em sequência: os agentes de produtos `P1` e `P2` podem operar tranquilamente (VER Figura 59 (a) - (b)), realizando uma coalizão, baseada em *skills*, de outros agentes de recursos no sistema (módulos mecatrônicos). Isto é uma forma de auto-organização por flexibilidade nos produtos. Por outro lado, o agente de produto `P3` não pode completar o seu processo produtivo, porque falta um módulo que realize um *skill* específico (VER Figura 59 (c)). Tão logo o módulo é inserido, ele retoma o seu processo e pode completá-lo (VER Figura 59 (d)).

Esta forma de encarar o processo produtivo, baseado em módulos e agentes, é uma mudança no paradigma de criação de aplicações para manufatura.

Durante a criação da proposta apresentada neste trabalho, esta foi sendo melhorada interativamente, dado o contato com parceiros industriais. De fato, a plataforma IADE foi desenvolvida para resolver um ponto importante para os parceiros: a capacidade de um mesmo equipamento ser usado em situações diversas, sem que a engenharia gastasse muito tempo para programar tais aplicações.

Em particular, a proposta apresentada permite se visualize um novo mercado para os sistemas de manufatura: o uso de sistemas de montagem sob demanda. Assim, uma empresa, que possui a necessidade de montar uma certa quantidade de produtos diferentes, entretanto tais quantidades não justificam a criação de linhas de produção específicas, pode contratar os serviços de um integrador, que usando da tecnologia aqui proposta, pode realizar a configuração do sistema conforme a necessidade da empresa e alugar o equipamento, por um tempo determinado, na produção daquelas quantidades.

De fato, esta era uma das necessidades de um dos parceiros o qual queria poder oferecer serviços sob demanda para a indústria farmacêutica.

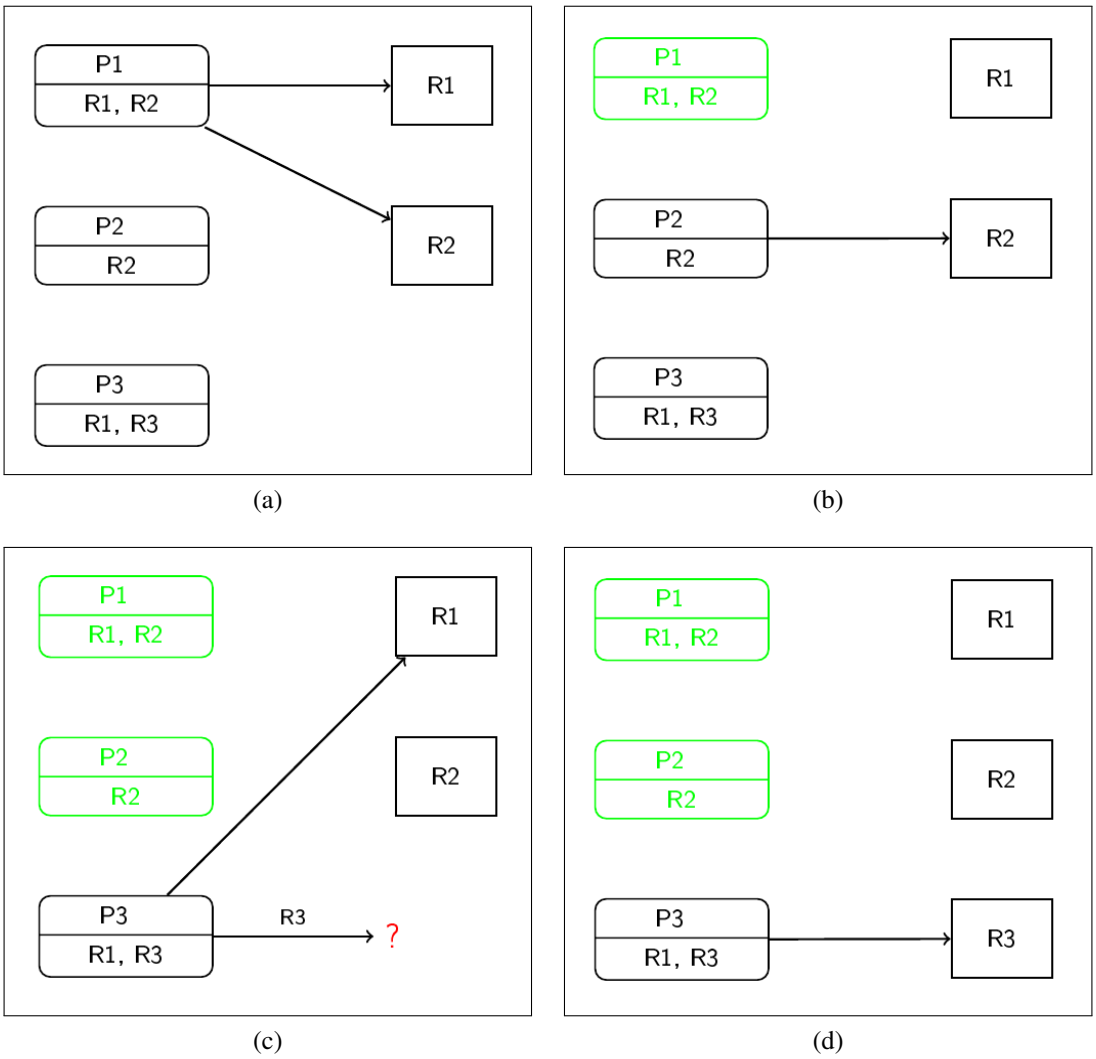

Figura 59: Mudança de paradigma na manufatura: inteligência do processo fora das máquinas que realizam o trabalho de montagem.

O uso de agentes no âmbito industrial está somente no seu inicio. Diversas ideias e aplicações são perceptíveis, desde a transformação do chão-de-fábrica em um grande sistema inteligente, integrado e geograficamente distribuído, as quais são necessidades atuais, quanto a maximização de sua capacidade de personalização, sem dúvida a tendência futura.

REFERÊNCIAS

ALLGAYER, R.; CAVALCANTE, A.; PEREIRA, C. E. Wireless Sensor Networks on Heterogeneous Platforms using RTSJ. In: REAL-TIME LINUX WORKSHOP, 10., 2008, Guadalajara. **Proceedings...** Guadalajara:Open Source Automation Development Lab, 2008.

ALSTERMAN, H.; ONORI, M. Definitions, Limitations and Approaches of Evolvable Assembly System Platforms. In: CAMARINHA-MATOS, L. (Ed.). **Emerging Solutions for Future Manufacturing Systems**. Vienna: Springer, 2005. v.159, p.367–377.

ALTERA. **Literature: White Papers**. Disponível em: <http://www.altera.com/literature/wp/lit-wp.jsp>. Acesso em: 06 maio 2009.

ARSHINOV, V.; FUCHS, C. (Ed.). **Causality, Emergence, Self-Organisation**. Moscow: NIA-Priroda, 2003. Supported by Austrian Ministry for Education, Science, and Culture.

ASHBY, W. Principles of the Self-organizing Systems. In: **E:CO Classical Papers**. London: Pergamon, 2004. v.6, n.1-2, p.102–126.

BABICEANU, R.; CHEN, F. Development and Applications of Holonic Manufacturing Systems: a survey. **Journal of Intelligent Manufacturing**, [S.l.], v.17, n.1, p.111–131, 2006.

BABICEANU, R. F.; CHEN, F. F.; STURGES, R. H. Real-time holonic scheduling of material handling operations in a dynamic manufacturing environment. **Robotics and Computer-Integrated Manufacturing**, Toronto, v.21, n.4-5, p.328–337, 2005.

BARATA, J. **Coalition Based Approach for Shop Floor Agility**. 2005. Tese — Universidade Nova de Lisboa, Lisboa. 329p.

BARATA, J.; CAMARINHA-MATOS, L. Coalitions of manufacturing components for shop floor agility - the CoBASA architecture. **International journal of networking and virtual organisations**, [S.l.], v.2, n.1, p.50–77, 2003.

BARATA, J.; CAMARINHA-MATOS, L.; ONORI, M. A Multiagent Based Control Approach for Evolvable Assembly Systems. In: IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL INFORMATICS (INDIN), 3., 2005, Perth. **Anais...** Perth:IEEE, 2005. p.478–483.

BARATA, J.; SANTANA, P.; ONORI, M. Evolvable Assembly Systems: a development roadmap. In: IFAC SYMPOSIUM ON INFORMATION CONTROL PROBLEMS IN MANUFACTURING, 12., 2006, St Etienne. **Anais...** St Etienne:IFAC, 2006.

BARBOSA, J.; LEITÃO, P.; PEREIRA, A. Combining adaptation and optimization in bio-inspired multi-agent manufacturing systems. In: IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS (ISIE), 2011, Gdansk. **Anais...** Gdansk:IEEE, 2011. p.1773 –1778.

BARONE, D. A. C. et al. **Sociedades artificiais** : a nova fronteira da inteligência nas máquinas. Porto Alegre: Bookman, 2003.

WOOLDRIDGE, M. (Ed.). **Developing Multi-Agent Systems with Jade**. [S.l.]: John Wiley & Sons, 2007.

BÖCKER, J. et al. Self-Optimization as a Framework for Advanced Control Systems. In: ANNUAL CONFERENCE OF THE IEEE INDUSTRIAL ELECTRONICS SOCIETY (IECON), 32., 2006, Paris. **Anais...** Paris:IEEE, 2006. p.4671–4675.

BUSSMANN, S.; MCFARLANE, D. Rationales for holonic manufacturing control. In: INTERNATIONAL WORKSHOP ON INTELLIGENT MANUFACTURING SYSTEMS, 2., 1999, Leuven. **Proceedings...** Leuven:IFAC, 1999. p.177–184.

CICIRELLO, V. **A game-theoretic analysis of multi-agent systems for shop floor routing**. Pittsburgh: Carnegie Mellon University, 2001.

DE WOLF, T.; HOLVOET, T. Emergence Versus Self-Organisation: different concepts but promising when combined. In: BRUECKNER, S. et al. (Ed.). **Engineering Self-Organising Systems**. Berlin: Springer-Verlag, 2005. v.3464, p.1–15.

DONOTH, J.; KLEINJOHANN, B.; ADELT, P. (Ed.). **Self-X in Engineering**. Karlsruhe: Karlsruher Institut für Technologie, 2010.

DORIGO, M.; BIRATTARI, M.; STUTZLE, T. Ant colony optimization. **Computational Intelligence Magazine, IEEE**, Bruxelles, v.1, n.4, p.28–39, 2006.

ELMARAGHY, H. Flexible and reconfigurable manufacturing systems paradigms. **International Journal of Flexible Manufacturing Systems**, [S.l.], v.17, p.261–276, 2005.

FERGUSON, S. et al. Flexible and Reconfigurable Systemas: nomemclature and review. In: INTERNATIONAL DESIGN ENGINEERING TECHNICAL CONFERENCES & COMPUTERS AND INFORMATION IN ENGINEERING CONFERENCE, 2007, Las Vegas. **Proceedings...** Las Vegas:IEEE, 2007. p.1–15.

FIPA. **FIPA Agent Communication Language Specifications**. Disponível em: <http://www.fipa.org/repository/aclspecs.html>. Acesso em: 20 Janeiro 2012.

FIPA. **FIPA Contract Net Interaction Protocol Specification**. Standard number 00029. Disponível em: <http://www.fipa.org/specs/fipa00029/>. Acesso em: 20 Janeiro 2012.

FIPA. **FIPA Request Interaction Protocol Specification**. Standard number 00026. Disponível em: <http://www.fipa.org/specs/fipa00026/>. Acesso em: 20 Janeiro 2012.

FREI, R. et al. An Architecture for Self-Managing Evolvable Assembly Systems. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETIC, 2009, San Antonio. **Anais...** San Antonio:IEEE, 2009. *pre-release*.

GAGLIARDI, M.; RAJKUMAR, R.; SHA, L. Designing for evolvability: building blocks for evolvable real-time systems. **Real-Time and Embedded Technology and Applications Symposium, IEEE**, Los Alamitos, v.1, p.100–109, 1996.

GAREY, M. R.; JOHNSON, D. S.; SETHI, R. The complexity of flowshop and job-shop scheduling. **Mathematics of Operations Research**, [S.l.], v.1, n.2, p.117–129, May 1976.

GAUSEMEIER, J.; FRANK, U.; STEFFEN, D. Intelligent Systems, Self-optimizing Concepts and Structures. In: RECONFIGURABLE MANUFACTURING SYSTEMS AND TRANSFORMABLE FACTORIES, 2006, Berlin. **Anais...** Berlin:Springer-Verlag, 2006. p.719–742.

GAUSEMEIER, J.; KAHL, S.; POOK, S. From Mechatronics to Self-Optimizing Systems. In: SELF-OPTIMIZING MECHATRONIC SYSTEMS: DESIGN THE FUTURE, 2008, Paderborn. **Anais...** Paderborn:Heinz Nixdorf Institut, 2008. p.3–32.

GÖTZ, M. **Run-time Reconfigurable RTOS for Reconfigurable Systems-on-Chip**. 2007. Tese — University of Paderborn and Federal Univerity of Rio Grande do Sul. 180p.

GOU, L.; LUH, P.; KYOYA, Y. Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation. **Computers in Industry**, [S.l.], v.37, n.3, p.213–231, 1998.

GU, W. et al. A neuroendocrine-inspired bionic manufacturing system. **Journal of Systems Science and Systems Engineering**, [S.l.], v.20, p.275–293, 2011.

GUNASEKARAN, A. Agile manufacturing: a framework for research and development. **International Journal of Production Economics**, [S.l.], v.62, n.1–2, p.87–105, 1999.

HORN, P. **Autonomic Computing: IBM's perspective on the state of information technology**. Armonk: IBM Coorporation, 2001.

HSIEH, F.-S. Design of evolvable manufacturing processes. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2002, Los Alamitos. **Proceedings...** Los Alamitos:IEEE Computer Society, 2002. v.1, p.339–344.

IDEAS Project. **D1.1 – IDEAS EPS Goals and Requirements**. Techreport. Unpublished.

IDEAS Project. **FP7 - Instantly Deployable Evolvable Assembly Systems (IDEAS) Project**. Stockholm: Royal Institute of Technology, 2010.

JADE Board. **JADE - Java Agent DEvelopment Framework**. Disponível em: <http://jade.tilab.com/>. Acesso em: 07 Abril de 2010.

JERALD, J. et al. Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm. **The International Journal of Advanced Manufacturing Technology**, London, v.25, p.964–971, 2005.

KACEM, I.; HAMMADI, S.; BORNE, P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. **IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews**, [S.l.], v.32, n.1, p.1–13, 2002.

KAKU, M. **Physics of the Future. How Science Will Shape Human Destiny And Our Daily Lives By The Year 2100.** New York: Doubleday, 2011.

KOCH, M.; OBERSCHELP, O. Simulation of self optimizing mechatronical systems with expert system knowledge. In: ASIAN CONTROL CONFERENCE, 5., 2004, Melbourne. **Anais...** Melbourne:IEEE, 2004. v.3, p.1437–445.

KOPETZ, H. The Real-Time Environment. In: **Real-Time Systems**. Virginia: Springer, 2011. p.1–28.

KOREN, Y. et al. Reconfigurable Manufacturing Systems. **CIRP Annals - Manufacturing Technology**, [S.l.], v.48, n.2, p.527–540, 1999.

LEITÃO, P. Agent-based distributed manufacturing control: A state-of-the-art survey. **Engineering Applications of Artificial Intelligence**, [S.l.], v.22, n.7, p.979–991, 2009.

LEITÃO, P.; RESTIVO, F. ADACOR: A holonic architecture for agile and adaptive manufacturing control. **Computers in Industry**, [S.l.], v.57, n.2, p.121–130, 2006.

LONGO, G. Communication, Technology, and the Planetary Creature. **tripleC - Cognition, Communication, Co-operation**, [S.l.], v.8, n.1, p.18–27, 2010.

LU, T.; YIH, Y. An agent-based production control framework for multiple-line collaborative manufacturing. **International Journal of Production Research**, [S.l.], v.39, n.10, p.2155–2176, 2001.

LUH, P. Scheduling of flexible manufacturing systems. In: SICILIANO, B.; VALAVANIS, K. (Ed.). **Control Problems in Robotics and Automation**. Berlin: Springer, 1998. v.230, p.227–243.

MATURANA, F. P.; NORRIE, D. H. Multi-agent Mediator architecture for distributed manufacturing. **Journal of Intelligent Manufacturing**, [S.l.], v.7, p.257–270, 1996.

MEHRABI, M.; ULSOY, A.; KOREN, Y. Reconfigurable manufacturing systems and their enabling technologies. **International Journal of Manufacturing Technology and Management**, [S.l.], v.1, n.1, p.114–131, 2000.

MEKID, S.; PRUSCHEK, P.; HERNANDEZ, J. Beyond intelligent manufacturing: a new generation of flexible intelligent nc machines. **Mechanism and Machine Theory**, [S.l.], v.44, n.2, p.466–476, 2009.

MEMORY ALPHA. **Earl Grey Tea**. Disponível em: <http://en.memory-alpha.org/wiki/Earl_Grey_tea>. Acesso em: 10 Novembro 2011.

MONOSTORI, L.; VANCZA, J.; KUMARA, S. Agent-based systems for manufacturing. **CIRP Annals-Manufacturing Technology**, [S.l.], v.55, n.2, p.697–720, 2006.

NAHM, Y.-E.; ISHIKAWA, H. A hybrid multi-agent system architecture for enterprise integration using computer networks. **Robotics and Computer-Integrated Manufacturing**, [S.l.], v.21, n.3, p.217 – 234, 2005.

ONORI, M. Evolvable assembly systems - a new paradigm? In: INTERNATIONAL SYMPOSIUM ON ROBOTICS (ISR), 33., 2002, Stockholm. **Anais...** Stockholm:International Federation of Robotics, 2002. p.617–621.

ONORI, M.; BARATA, J.; FREI, R. Evolvable Assembly Systems Basic Principles. **Information Technology for Balanced Manufacturing Systems**, New York, p.317–328, 2006.

PAPADIMITRIOU, C. H. Computational complexity. In: **Encyclopedia of Computer Science**. Chichester: John Wiley and Sons, 2003. p.260–265.

PARUNAK, H.; BAKER, A.; CLARK, S. The AARIA agent architecture: from manufacturing requirements to agent-based system design. **Integrated Computer Aided Engineering**, [S.l.], v.8, n.1, p.45–58, 2001.

PARUNAK, H. V. D.; BAKER, A.; CLARK, S. J. The AARIA agent architecture: an example of requirements-driven agent-based system design. In: AUTONOMOUS AGENTS, 1997, New York. **Proceedings...** New York:ACM, 1997. p.482–483.

PEETERS, P. et al. Pheromone based emergent shop floor control system for flexible flow shops. **Artificial Intelligence in Engineering**, [S.l.], v.15, n.4, p.343–352, 2001.

PEREIRA, C. E.; CARRO, L. Distributed real-time embedded systems: recent advances, future trends and their impact on manufacturing plant control. In: INFORMATION CONTROL PROBLEMS IN MANUFACTURING (INCOM), 2006, Oxford. **Anais...** Oxford:Elsevier, 2006. p.21 – 32.

QIU, L. et al. Scheduling and routing algorithms for AGVs: a survey. **International Journal of Production Research**, [S.l.], v.40, n.3, p.745–760, 2002.

RIBEIRO, L.; BARATA, J.; COLOMBO, A. MAS and SOA: a case study exploring principles and technologies to support self-properties in assembly systems. In: IEEE INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS WORKSHOPS, 2., 2008, Venice. **Anais...** Venice:IEEE, 2008. p.192–197.

RIBEIRO, L. et al. Evolvable Production Systems: an integrated view on recent developments. In: CIRP-SPONSORED INTERNATIONAL CONFERENCE ON DIGITAL ENTERPRISE TECHNOLOGY, 6., 2009, Hong Kong. **Proceedings...** Hong Kong:Springer, 2009. p.841–854.

RIBEIRO, L. et al. IT Support of Mechatronic Networks: a brief survey. In: IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS, 20., 2011, Gdansk. **Proceedings...** Gdansk:IEEE, 2011.

RIBEIRO, L. et al. IADE – Ideas Agent Development Environment: lessons learned and research directions. In: CIRP CONFERENCE ON ASSEMBLY TECHNOLOGIES AND SYSTEMS (CATS), 4., 2012, Michigan. **Proceedings...** Michigan:CIRP, 2012.

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**. São Paulo: Editora Campus, 2004.

SALLEZ, Y.; BERGER, T.; TRENTESAUX, D. A stigmergic approach for dynamic routing of active products in FMS. **Computers in Industry**, [S.l.], v.60, n.3, p.204–216, 2009.

SANCHEZ, L. M.; NAGI, R. A review of agile manufacturing systems. **International Journal of Production Research**, [S.l.], v.39, n.16, p.3561–3600, 2001.

SHALIZI, C. R. **Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata**. 2001. Tese — University of Wisconsin at Madison.

SHEN, W. Distributed manufacturing scheduling using intelligent agents. **IEEE Intelligent Systems**, [S.l.], v.17, n.1, p.88–94, 2002.

SHEN, W. et al. Applications of agent-based systems in intelligent manufacturing: an updated review. **Advanced Engineering Informatics**, [S.l.], v.20, n.4, p.415–431, 2006.

SHEN, W. et al. An agent-based service-oriented integration architecture for collaborative intelligent manufacturing. **Robotics and Computer-Integrated Manufacturing**, [S.l.], v.23, n.3, p.315–325, 2007.

SHEN, W.; WANG, L.; HAO, Q. Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. **Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on**, [S.l.], v.36, n.4, p.563–577, July 2006.

SINRIECH, D.; KOTLARSKI, J. A dynamic scheduling algorithm for a multiple-load multiple-carrier system. **International Journal of Production Research**, [S.l.], v.40, n.5, p.1065–1080, 2002.

SOUSA, P.; RAMOS, C.; NEVES, J. Scheduling in Holonic Manufacturing Systems. In: WANG, L.; SHEN, W. (Ed.). **Process Planning and Scheduling for Distributed Manufacturing**. London: Springer, 2007. p.167–190.

TANENBAUM, A. S. The Network Layer. In: **Computer networks**. 4th.ed. New Jersey: Prentice Hall, 2003.

THARUMARAJAH, A.; WELLS, A.; NEMES, L. Comparison of emerging manufacturing concepts. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS, 1998, San Diego. **Anais...** San Diego:IEEE, 1998. v.1, p.325–331.

UEDA, K. A concept for bionic manufacturing systems based on DNA-type information. In: IFIP TC5/WG5.3 EIGHT INTERNATIONAL PROLAMAT CONFERENCE ON HUMAN ASPECTS IN COMPUTER INTEGRATED MANUFACTURING, 1992, Amsterdam. **Proceedings...** Amsterdam:North-Holland Publishing, 1992. p.853–863.

UEDA, K. Emergent Synthesis Approaches To Biological Manufacturing Systems. In: CUNHA, P.; MAROPOULOS, P. (Ed.). **Digital Enterprise Technology**. New York: Springer, 2007. p.25–34.

USHER, J. M. Negotiation-based routing in job shops via collaborative agents. **Journal of Intelligent Manufacturing**, [S.l.], v.14, p.485–499, 2003.

VALCKENAERS, P.; BRUSSEL, H. V. Holonic Manufacturing Execution Systems. **CIRP Annals - Manufacturing Technology**, [S.l.], v.54, n.1, p.427 – 432, 2005.

VAN BRUSSEL, H. et al. Reference architecture for holonic manufacturing systems: PROSA. **Computers in industry**, [S.l.], v.37, n.3, p.255–274, 1998.

VAQUERO, T.; SILVA, J.; BECK, J. A Brief Review of Tools and Methods for Knowledge Engineering for Planning & Scheduling. In: WORKSHOP ON KNOWLEDGE ENGINEERING FOR PLANNING AND SCHEDULING, 2011, Freiburg. **Proceedings...** Freiburg:University of Freiburg, 2011. v.1, p.7–14.

VRBA, P.; HRDONKA, V. Material Handling Problem: FIPA Compliant Agent Implementation. In: MARÍK, V. et al. (Ed.). **Multi-Agent Systems and Applications II**. Berlin: Springer, 2002. v.2322, p.27–45.

WANG, L.; WU, S. Modeling with colored timed object-oriented Petri nets for automated manufacturing systems. **Computers & Industrial Engineering**, [S.l.], v.34, n.2, p.463–480, 1998.

WESTKAEMPER, E. New Trends in Production. In: DASHCHENKO, A. I. (Ed.). **Reconfigurable Manufacturing Systems and Transformable Factories**. Berlin: Springer, 2006. p.15–26.

WHITBY, B. **Artificial Intelligence**: A Beginner's Guide. New York: Oneworld Publications, 2003.

WONG, T. et al. Dynamic shopfloor scheduling in multi-agent manufacturing systems. **Expert Systems with Applications**, [S.l.], v.31, n.3, p.486–494, 2006.

WOOLDRIGE, M. **An introduction to multiagent systems**. London: John Wiley & Sons, Ltd, 2002. 966p.

XIA, W.; WU, Z. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. **Computers & Industrial Engineering**, [S.l.], v.48, n.2, p.409–425, 2005.

XIANG, W.; LEE, H. Ant colony intelligence in multi-agent dynamic manufacturing scheduling. **Engineering Applications of Artificial Intelligence**, [S.l.], v.21, n.1, p.73–85, 2008.

XILINX. **Xilinx Support Documentation**. Disponível em: <http://www.xilinx.com/support/documentation/index.htm>. Acesso em: 06 Maio 2009.

YUSUF, Y.; SARHADI, M.; GUNASEKARAN, A. Agile manufacturing:: the drivers, concepts and attributes. **International Journal of Production Economics**, [S.l.], v.62, n.1–2, p.33 – 43, 1999.

ZAMPIERI, S. Trends in networked control systems. In: IFAC WORLD CONGRESS, 17., 2008, Seoul. **Anais...** Seoul:IFAC, 2008. p.6–11.

Criado e editado usando JABREF, KILE, OKULAR e LaTeX 2_{ε}.