Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia

Departamento de Engenharia Electrotécnica

An Ontology to Support Evolvable Production Systems

Por

António Correia de Campos Jordão Amado

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador: *Prof. Dr. José António Barata de Oliveira*

Lisboa

2009

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia

Departamento de Engenharia Electrotécnica

An Ontology to Support Evolvable Production Systems

Por

António Correia de Campos Jordão Amado

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador: *Prof. Dr. José António Barata de Oliveira*

Lisboa

2009

Agradecimentos

Quero deixar público os meus agradecimentos aos meus pais, pelo apoio que me deram durante a realização de todo este trabalho. À minha irmã quero agradecer pela ajuda que me deu ao ler este relatório e fazer comentários e correcções que fizeram aumentar a qualidade do mesmo.

Ao meu Professor e Orientador, Prof. Dr. José Barata, também um obrigado lhe dirijo por todo o esforço e ajuda que, não só a melhorou a qualidade deste trabalho, mas também fez com que a sua realização se concretizasse.

Ao meu colega e amigo Bruno Ferreira um obrigado por todo o tempo despendido durante toda a realização deste trabalho e às muitas horas passadas no Departamento de Engenharia Electrotécnica.

A minha namorada, Dina Macedo, também vou agradecer pela ajuda disponibilizada na correcção e melhoria de todo este relatório, que seria entregue com muitos erros se não tivesse a ajuda e paciência dela.

A todos os meus amigos que me apoiaram em todos os momentos, em especial, nos mais difíceis sem eles teria sido bastante mais difícil a realização de todo este trabalho.

Faculdade de Ciências e Tecnologia, Setembro de 2009

_____

António Amado

Resumo

Ontologias são cada vez mais um conceito fundamental no suporte à interoperabilidade. Além disso, elas também são fundamentais no suporte aos sistemas evolutivos de produção por duas razões principais. A primeira está relacionada com o facto de a clara identificação e formalização dos processos ser importante para a criação de módulos inteligentes. A segunda razão está relacionada com o facto de os sistemas evolutivos de produção (SEP) serem baseados em sistemas multi-agente que depende em muito, da construção das ontologias de modo a permitir a comunicação entre os agentes pertencentes ao sistema.

Os principais conceitos por detrás da ontologia aqui desenvolvida serão os conceitos de processos, tarefas, produto e componentes de manufactura.

Esta tese pretende mostrar não só a criação de uma ontologia, mas também de um agente de modo a ser possível a integração da ontologia num sistema multi-agente, no âmbito da manufactura inteligente respondendo às questões envolventes ao paradigma dos sistemas evolutivos de produção. Sabendo que os SEP são baseados em sistemas multi-agente, será também mostrado um agente que irá ter todo o controlo da ontologia e irá pertencer ao sistema de manufactura.

Palavras-chave:

Ontologia, Processo, Tarefa, Produto, Inteligência Artificial, Sistemas de Manufactura Inteligente, Sistemas Inteligentes baseados no Conhecimento, Sistemas Adaptativos

Abstract

Ontologies are becoming more and more a fundamental issue to support interoperability. In addition to that they are a fundamental issue to support evolvable production systems for two main reasons. The first one is related to the fact that a clear identification and formalization of processes are fundamental to support the creation of intelligent modules. The second reason is related to the fact that Evolvable Production Systems (EPS) are essentially based on the Multi-agent paradigm that depends very much of well constructed ontologies for the communication among the agents.

The main aspects behind the ontology here developed will be the concepts of processes, skills, product and manufacturing components.

This thesis will show the development of an ontology and a specific agent responsible for integrating this ontology in a multi-agent based manufacturing system that responds to the evolvable production systems paradigm. Because EPS are based on multi-agent systems, it also will be shown an agent responsible for issues with the ontology and also it will belong to the manufacturing system.

Keyword:

Ontology, Process, Skill, Product, Artificial Intelligence, Intelligent Manufacturing Systems, Intelligent Knowledge-based systems, Adaptative Systems

Índice

Índice de Figuras

Lista de Siglas

OWL – Ontology Web Language

DL – Description Logic

SWRL – Semantic Web Rule Language

SQWRL – Semantic Query Enhanced Web Rule Language

JESS – Java Expert System Shell

JADE – Java Agent Development framework

FIPA – Foundation for Intelligent Physical Agents

XML – Extensible Markup Language

XSD – XML Schema Definition

EAS – Evolvable Assembly System

EPS – Evolvable Production System

RMS – Reconfigurable Manufacturing System

FMS – Flexible Manufacturing System

HMS – Holonic Manufacturing System

OntA – Ontology Agent

GUI – Graphic Unit Interface

AMI – Agent Machine Interface

MRA – Manufacturing Resource Agent

TA – Transport Agent

OA – Order Agent

PA – Palette Agent

AI – Artificial Intelligence

API – Application Programmable Interface

Capítulo 1. Introdução

1.1 Descrição do problema

Até aos anos 80 a indústria de manufactura era baseada numa produção funcional, pouco flexível, com gamas de produtos bastante limitadas.

A partir desta década, o crescimento da sociedade económica e o aumento do nível de vida, levaram a um aumento do consumo de produtos com ciclo de vida curto e descartável.

Estas novas circunstâncias levaram a uma grande necessidade de inovação na indústria para esta conseguir gerar produtos exclusivos a baixo preço e de qualidade que conseguissem satisfazer os novos padrões de exigência dos consumidores. Os objectivos de produção passaram então a ser orientados ao produto, virado para o dinamismo e para a inovação, com unidades fabris mais pequenas, mais flexíveis. Isto levou a uma nova filosofia de manufactura, a que se deu o termo de *mass customization*, familiarizado e expandido por Joseph Pine II em 1992 (Pine II 1993).

As soluções para sistemas de controlo de *shopfloors* devem ir ao encontro dos seguintes objectivos, de forma a preencher os requisitos do mercado:

- Tem de ter uma interface amigável e visualmente apelativa – um produto de software para ser usado a nível do *shopfloor*, ou a níveis próximos do *shopfloor*, tem de permitir uma interacção quase intuitiva.
- Tem de ser altamente flexível, de modo a ser personalizado facilmente para diferentes tipos de estrutura de produção – o software para controlo tem de se adaptar à organização do ambiente de manufactura. Com o objectivo de uma optimização contínua, este ambiente é reestruturado e reorganizado frequentemente. Deve ser possível a actualização das ferramentas de modo a suportar tais mudanças.

- Tem de ser possível a integração do sistema controlo do *shopfloor* com os sistemas de agendamento, de produção de planos e sistemas de controlo já existentes – soluções isoladas não são mais aceitáveis na indústria de manufactura moderna.
- Tem de permitir o uso de interfaces de outros sistemas – A integração de informação e sistemas já existentes torna-se cada vez mais importante.
- Tem de facilitar a cooperação e a comunicação entre diferentes células de manufactura dentro de uma mesma planta de manufactura – A redução de níveis na estrutura organizacional cria a necessidade de uma comunicação e cooperação horizontal. A hierarquia de comando vertical é substituída por uma interacção directa no nível do shopfloor.
- Tem de permitir a comunicação entre fornecedores, manufactores e consumidores, permitindo um maior grau de integração ao longo da cadeia de fornecimento externa.

Por reconfigurabilidade entende-se a possibilidade de um sistema permitir a adição e/ou remoção de componentes sem perda de eficiência. A computação reconfigurável pode ser definida, portanto, pela habilidade de se configurar continuamente uma máquina de forma a esta poder realizar variadas funções.

Em sistemas reconfiguráveis há que se ter, no entanto, certos cuidados relativos aos tempos em que se faz essa reconfiguração uma vez que existe o problema dos custos envolvidos, ou seja, a aplicação da lógica reconfigurável implica parar a computação para inicializar a nova configuração. Contudo, algumas aplicações permitem reconfiguração dinâmica, isto é, enquanto se reconfigura uma parte da aplicação, as outras partes continuam a execução de tarefas.

Os sistemas de manufactura reconfiguráveis (RMS) têm a sua origem na ciência computacional onde sistemas de computação tentam lidar com a ineficiência dos sistemas convencionais devido a estruturas de hardware fixas e à lógica computacional.

Os sistemas de manufactura reconfiguráveis são desenhados de modo a permitir, de forma rápida e eficaz, mudanças tanto na estrutura já existente, como nos componentes de hardware e software, seja adicionando, removendo ou modificando capacidades específicas de processamento, controlos, software ou estruturas com o objectivo de ajustar a capacidade de produção em resposta às necessidades do mercado ou novas tecnologias. Este tipo de sistemas oferece uma flexibilidade personalizada para uma dada família de produtos permitindo melhorar, actualizar e reconfigurar em vez de substituir.

A parte principal do paradigma dos RMS traduz-se numa aproximação à reconfiguração baseada no desenho e integração de módulos e controladores reconfiguráveis com uma arquitectura aberta. Os RMS's podem, ainda, ser caracterizados por:

- Modularidade: Uso de várias unidades comuns para criar variantes ao produto. Em RMS todos os componentes principais são modulares;
- Integrabilidade: As interfaces usadas devem permitir as ligações e comunicação entre os vários módulos do sistema de forma coerente e eficaz;
- Personalização: Os módulos podem ser definidos em termos de controlo e flexibilidade;
- Convertibilidade: Os tempos de conversão dos diferentes grupos de produção devem ser baixos;
- Diagnosticabilidade: Uma vez que se emprega a modularidade nestes sistemas, a detecção e identificação de causas de comportamentos anormais deve ser relativamente fácil.

Cada vez mais os sistemas de manufactura são compostos por módulos criados de forma independente para facilitar a integração heterogénea de componentes com diferentes níveis de modularidade de modo a ser possível atingir o produto final pretendido com o menor custo possível no menor espaço de tempo. Esta heterogeneidade leva a que cada componente comunique de uma forma específica. Para que haja comunicação entre módulos diferentes é necessário cada um ter conhecimento dos outros.

Esta diferença de modularidade faz com que a comunicação entre os diversos módulos seja um processo complexo. Esta complexidade reside no facto de cada módulo comunicar de forma diferente. Para processar a comunicação é necessário que cada módulo saiba previamente com quem vai comunicar e qual a forma de comunicar. Tal como, para que duas pessoas de nacionalidades distintas consigam comunicar, é necessário a existência de "dicionário" e que esse dicionário seja conhecido por quem necessitar dele para a comunicação ser feita com sucesso. Também aqui, existe a necessidade de cada módulo ter conhecimento desse dicionário. Para evitar colocar todo o conhecimento em todos os módulos, cria-se um módulo específico nesta matéria, chamado Ontologia. A ontologia tem como função reter toda a informação existente num dado sistema de modo a que quando se faça a integração de um novo módulo, esse novo módulo só necessite de comunicar com a ontologia para ter a informação que precisa de modo a interagir com os outros módulos já existentes para se processar a manufactura pretendida. Será na ontologia que será guardado o "dicionário".

No contexto dos sistemas reconfiguráveis de manufactura (RMS), os sistemas evolutivos de produção (EPS) (Frei, Barata et al. 2006; Semere, Barata et al. 2007) focam a

reconfigurabilidade através de adaptação com inspiração biológica, tanto a nível mecânico como a nível de controlo.

O objectivo deste trabalho é desenvolver uma Ontologia que consiga dar suporte a sistemas evolutivos de produção num sistema multi-agente. Para conseguir integrar a ontologia no sistema multi-agente foi também desenvolvido um agente que irá servir de ponte de ligação entre todos os outros agentes presentes no sistema e a ontologia.

1.2 Organização da tese

O presente relatório é composto por seis capítulos. Nos próximos parágrafos será feita uma breve descrição de cada um dos capítulos.

No primeiro capítulo (capítulo corrente), é feita uma introdução ao trabalho efectuado como uma motivação, com o objectivo de cativar o leitor para o resto da leitura.

No segundo capítulo é feita uma descrição teórica do que realmente são ontologias, como se constrói uma ontologia, das linguagens que envolve o tema das ontologias, e alguns projectos importantes já realizados em diversas áreas de investigação. Projectos como ontoMAS (Lohse, Ratchev et al. 2006) (*Ontology for the design of Modular Assembly Systems*), coBASA (Barata 2003) (*Coalition based Approach for Shop Floor Agility – A MultiAgent Approach*), entre outros são aqui referidos. O paradigma dos sistemas evolutivos de produção (SEP) também é referido neste capítulo como uma importante base para a ontologia aqui apresenta visto ser um paradigma actualmente presente no mundo da investigação a nível da manufactura e também como um paradigma de suporte à ontologia aqui desenvolvida.

No terceiro capítulo são descritas todas as tecnologias que deram suporte ao desenvolvimento do trabalho aqui apresentado. È feita uma descrição detalhada de cada tecnologia bem como o seu papel neste trabalho. A iniciar este capítulo é apresentado um programa de computador designado por Protégé que serviu como base à construção da ontologia. Depois a linguagem que permitiu a construção desta mesma ontologia, *Ontology Web Language* (OWL) é descrita também neste capítulo. A implementação de regras nesta ontologia foi realizada através de uma outra linguagem existente, também detalhada neste capítulo, designada por *Semantic Web Rule Language* (SWRL). De modo a desenvolver o agente que permite a integração e o acesso à ontologia num sistema multi-agente foram utilizadas outras duas tecnologias: JADE que representa uma biblioteca de funções para permitir o desenvolvimento de agentes e a linguagem Java (Arnold, Gosling et al. 1998) que foi a linguagem de programação que permitiu dar vida a este sistema.

No quarto capítulo é descrita com detalhe toda a arquitectura implementada neste sistema que permite o suporte da ontologia a sistemas evolutivos de produção bem como a sua integração num sistema multi-agente. Todas as classes existentes na ontologia são detalhadas, bem como a sua função. A descrição do agente aqui criado também é feita e qual a função deste agente no sistema.

No quinto capítulo é apresentada toda a implementação prática efectuada com base na arquitectura previamente definida. São descritos todos os detalhes práticos do agente desenvolvido, bem como as suas funções e importância no sistema. De suporte a esta implementação prática é apresentado um estudo de caso que permitiu comprovar os conceitos criados na ontologia. Este estudo de caso baseia-se num sistema didáctico presente no Departamento de Engenharia Electrotécnica da Universidade Nova de Lisboa, designado por MOFA France. De modo a compreender este sistema é feita uma descrição da sua composição com a ajuda de imagens.

No sexto capítulo são apresentadas as conclusões deste trabalho.

Por fim, no sétimo e último capítulo é apresentada toda a bibliografia consultada que permitiu o desenvolvimento deste trabalho.

Capítulo 2. Estado da arte

2.1 Introdução

A investigação efectuada no âmbito desta tese teve como objectivo responder a uma vasta gama de perguntas relacionadas com o conceito Ontologia. Perguntas como "o que é uma ontologia?", "o que se faz com uma ontologia?", "como desenvolver uma ontologia?", "porquê desenvolver uma ontologia?", entre outras. O conceito de ontologia está presente numa vasta gama de áreas. Neste contexto, as respostas encontradas tiveram por base os paradigmas da manufactura existentes (referidos neste capítulo) nos dias de hoje para que o seu desenvolvimento fosse enquadrado da melhor forma com os sistemas evolutivos de produção (EPS).

O desenvolvimento da World Wide Web permitiu que milhões de pessoas tivessem acesso à Internet e acesso a documentos (bem como as suas publicações) de uma maneira simples. Contudo o crescimento explosivo do número de documentos publicados na Internet levou a um problema de excesso de informação (Simon 2006). De modo a solucionar este problema, está ser investigada a possibilidade de ser criada a Web semântica (*Semantic Web*), onde o conhecimento e a informação são explicitamente postas de forma a permitir que computadores e máquinas processem e integrem a Web de forma inteligente. A utilização de ontologias, além de permitir pesquisas rápidas e precisas, permite também o desenvolvimento de agentes inteligentes na Internet e facilitar a comunicação entre dispositivos heterogéneos baseados na Internet (Breitman and Leite 8-12 Sept. 2003). As ontologias irão dar significado ao conteúdo da Web permitindo a agentes da internet compreender a informação nos contextos devidos.

Voltando a questão das ontologias no contexto dos paradigmas da manufactura, que de facto é onde a ontologia aqui desenvolvida se irá enquadrar, a sua existência num sistema de manufactura permite um maior grau de modularidade dentro do sistema. Também a existência de componentes heterogéneos torna o respectivo processo de integração mais simples devido ao facto de que para dois componentes comunicarem não é necessário especificar o modo de comunicação de cada um, bastando a interacção com a ontologia, onde estará registado o conhecimento de todos os módulos do sistema e as suas especificações a nível de comunicação.

2.2 Ontologias

A palavra ontologia deriva do Grego *ontos* (ser) + *logos* (palavra). Foi uma palavra introduzida na filosofia no sec. XIX, por filósofos Alemães. Sendo uma disciplina filósofa, a ontologia tem como objectivo catalogar as diferentes visões do que representa o mundo (Guarino 1998).

No mundo das tecnologias de informação, as ontologias foram desenvolvidas na área da Inteligência Artificial (IA) de modo a facilitar a reutilização e a partilha da informação. Nos dias de hoje, as ontologias são largamente utilizadas em áreas como a integração de informação inteligente, comércio electrónico, engenharia de software baseado em agentes, entre outras (Breitman and Leite 8-12 Sept. 2003).

Na literatura da Inteligência Artificial existem diversas definições de uma ontologia. Segundo (Noy and McGuinness 2001), uma ontologia é uma descrição explícita e formal de conceitos num determinado domínio. Esta descrição é então composta por classes (também designadas de conceitos), propriedades de cada classe descrevendo características e atributos dessa classe e também por restrições nas propriedades. Em cada classe é possível criar diversas instâncias (também conhecidas por indivíduos). Estas instâncias em conjunto com as classes formam uma base de conhecimento. As classes podem conter subclasses que representam conceitos mais específicos que as classes.

No mundo da manufactura inteligente, existe a necessidade de integrar componentes com diferentes níveis de modularidade de forma a facilitar todo o processo de reengenharia. Sempre que existe um processo de reconfiguração, este fica simplificado ao existirem diferentes níveis de modularidade. O mesmo componente pode ser reutilizado para realizar diferentes tarefas de acordo com os pré-requisitos do produto final que se pretende atingir. Todo este processo de

reconfiguração requer que todos os componentes tenham conhecimento de todos os outros de forma a conseguirem atingir a configuração que se pretende. A existência de uma ontologia nestes sistemas irá simplificar este processo. É possível tornar os componentes mais simples, no sentido em que não têm que se "preocupar" com os outros. Esta tarefa fica para a ontologia. O conhecimento do sistema e dos seus agentes deixa de estar localizado em todos os seus intervenientes, passando a estar a cargo da ontologia.

Sempre que existe a necessidade de reconfiguração, é à ontologia que se recorre para obter o conhecimento pretendido.

Uma ontologia produz uma linguagem comum de modo a ser possível a reutilização e a partilha de conhecimento de um domínio em particular.

As ontologias estão a ser cada vez mais utilizadas. Uma simples pesquisa na Internet irá devolver milhares de resultados de páginas. Algumas razões que provavelmente levam a que as ontologias estejam cada vez mais a ter um papel relevante são:

- Aumento dos agentes computacionais - Os sistemas de computadores estão cada vez a centrar-se numa orientação aos agentes ao invés do humano. Estes agentes estão cada vez mais a substituir o papel do Humano e a interacção entre diferentes agentes está a aumentar.
- Importância da *World Wide Web* – Com o aumento da *Web* como suporte não só cientifico, mas também a nível empresarial e pessoal, está a ser desenvolvida uma nova *Web* com o nome *Web Semântica*. A maioria da informação presente na *Web* destina-se a ser interpretada por Humanos. Os computadores são muito mais eficazes a interpretarem informação estruturada. Nos dias de hoje existe a necessidade de existir informação que não seja apenas apresentada para os Humanos, mas também que seja interpretada por computadores.

Em termos práticos, desenvolver uma ontologia inclui:

- Definir classes na ontologia;
- Organizar as classes hierarquicamente (definir classes e subclasses);
- Definir propriedades bem como as suas restrições;
- Instanciar (ou seja, criar indivíduos) dando valores às propriedades.

Exemplo de uma ontologia simples:

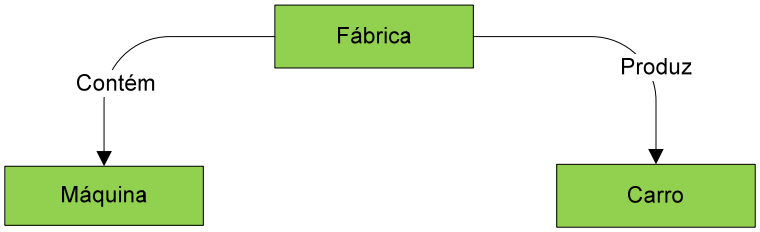

Figura 2.2.1 – Exemplo de uma ontologia

Nesta figura (Figura 2.2.1) podem-se observar três classes (“Fábrica”, “Carro” e “Máquina”). Cada classe de uma ontologia pretende ser uma representação abstracta do domínio que se pretende especificar. Por exemplo, a classe “Fábrica” pretende ser uma representação abstracta de uma fábrica real que produz um determinado produto (neste caso um carro). Qualquer fábrica real tem máquinas no seu interior que pertencem à planta fabril. Estas máquinas também são representadas na ontologia por uma classe. O “Carro”, sendo o produto final da fábrica, é também representado por uma classe. Qualquer classe pode ser caracterizada e instanciada. Para caracterizar uma classe criam-se propriedades. Para instanciar uma classe criam-se instâncias. Por exemplo se pretendermos ter duas fábricas, não vamos criar duas classes com o nome fábrica; criamos uma classe e em seguida instanciamos essa classe duas vezes. Deste modo obtemos uma representação abstracta do nosso mundo físico.

Na classe Fábrica foram criadas duas propriedades: “Produz” e “Contém”. A propriedade “Produz” tem uma restrição indicando que os seus valores têm que pertencer à classe Carro e a propriedade “Contém” tem também uma restrição que indica que os seus valores têm que pertencer à classe Máquina. Deste modo, sem ter qualquer outro conhecimento prévio sobre o sistema e só através da ontologia, conseguimos saber o que contém a fábrica (contém máquinas) e o que ela produz (carros). Para caracterizar os produtos desta fábrica só teríamos que adicionar propriedades à classe Carro. Da mesma forma, para caracterizarmos as máquinas pertencentes à fábrica só teríamos que adicionar propriedades à classe Máquina.

Por cada instância criada em cada classe, teríamos um objecto diferente. Ou seja, se instanciarmos a classe máquina duas vezes, estaríamos a dizer que a fábrica continha duas máquinas. De igual forma se processa com os produtos da fábrica. Por cada instância da classe Carro teríamos um carro diferente.

2.3 Paradigmas da manufactura

Actualmente, os mercados são altamente dinâmicos, obrigando os sistemas de manufactura a serem modulares, distribuídos e flexíveis (ou reconfiguráveis).

Neste subcapítulo serão descritos alguns paradigmas da manufactura, tais como, sistemas reconfiguráveis, sistemas flexíveis e sistemas evolutivos. Este último é referido com maior detalhe visto ser o paradigma baseado para o desenvolvimento deste trabalho. Para os outros, é feita uma pequena comparação para se perceber quais as principais diferenças existente entre eles.

2.3.1 Sistemas Evolutivos de Produção

Os sistemas Evolutivos de Produção (EPS/EAS) (Onori 2002; Frei, Barata et al. 2006; Frei, Barata et al. 2007) representam um paradigma da manufactura que tem como objectivo apresentar uma solução para o sistema de manufactura, que através de elementos básicos (módulos), reconfiguráveis com tarefas próprias, permitir uma contínua evolução no sistema. A reconfiguração destes módulos é feita através de pré-requisitos definidos em (Frei, Barata et al. 2006):

- **Módulo –** É uma unidade capaz de realizar uma tarefa e integrar uma interface específica. Níveis de granularidade devem ser definidos (o nível mínimo, e o maior grau de emergência);
- **Granularidade –** O menor nível de granularidade de um módulo dentro de uma arquitectura de referência será, por exemplo, uma *gripper* ou um suporte; o maior comportamento de emergência será, por exemplo, se uma *gripper* conseguir comunicar com um robô. Novas características operacionais deverão emergir (virar um produto, parte em movimento, ajustamento de posição, etc.), no entanto isto implica que um conjunto de definições e formas de gerir informação devem surgir, os níveis mínimos devem ainda ser clarificados;
- **Conectividade** – A habilidade para voltar a encontrar e integrar módulos do sistema com a plataforma de trabalho de uma dada arquitectura. O novo *"layout"*

não deverá excluir a performance, simplesmente se irá juntar e formar uma nova disposição;

- **Configurabilidade** (interoperabilidade) – A habilidade para voltar a encontrar módulos do sistema onde estejam disponíveis operações novas e pré-definidas (conectividade mais as características que asseguram a performance eficiente do novo *"layout"*);
- **Evolução** – Um sistema completamente reconfigurável que exibe comportamentos emergentes os quais introduzem níveis de funcionalidade novos ou redefinidos. Para tal requer-se uma arquitectura de referência rigorosamente bem definida de modo a permitir a correcta aplicação das características mais relevantes.

Uma aplicação onde é possível ver o sucesso do paradigma dos Sistemas Evolutivos de Produção, pode ser visto em (Barata 2003), onde os EPS são aplicados através de um sistema multi-agente.

O conceito de ontologia é um conceito largamente utilizado na área da inteligência artificial, na engenharia de conhecimento e na área da informática em aplicações que envolvem a gestão de informação e de conhecimento.

A arquitectura de referência definida para os EPS/EAS, descrita em (Semere, Barata et al. 2007) define a relação existente entre as entidades existentes no sistema e conceitos que podem ser caracterizados por: produto, processo e recurso.

Uma ontologia definida para os EPS/EAS deve ter determinadas características (Semere, Barata et al. 2007):

- Reutilização: A construção da ontologia deve ser feita tendo em vista que o seu conhecimento será reutilizado por diferentes aplicações através de diferentes vistas;
- Consensual: A ontologia é como uma visão partilhada de um determinado domínio;
- Formalidade: De modo a evitar ambiguidades, deverá ser utilizada uma formalidade na representação dos conceitos;
- Conceptualização: conceptualização é a ideia básica que uma pessoa/grupo tem sobre o mundo.

2.3.2 Sistemas Reconfiguráveis, Flexíveis e Holónicos

Sistemas Reconfiguráveis de Manufactura (RMS), Sistemas Flexíveis de Manufactura (FMS) e Sistemas Evolutivos de Produção (EPS/EAS) são muitas vezes confundidos. Uma comparação entre estes sistemas é feita em (Frei, Barata et al. 2006). Sistemas reconfiguráveis começam, na maioria dos casos, com o produto e os requisitos. Sistemas flexíveis partem de máquinas que oferecem várias funcionalidades com elevados custos que na maioria dos casos são relativamente úteis, mas óptimas em nenhuma delas. A reconfigurabilidade pode ser vista como uma evolução da flexibilidade.

Sistemas evolutivos diferem bastantes destes dois tendo em conta os seguintes aspectos:

- Focus principal: RMS estão focados na reconfigurabilidade dos componentes do sistema, não sendo necessariamente automática; EPS/EAS estão focados na adaptabilidade dos componentes do sistema através de propriedades emergentes;
- Início do desenvolvimento: RMS utilizam as características actuais do produto enquanto os EPS/EAS focam numa reengenharia do sistema;
- Nível de modularidade: os RMS aplicam uma divisão convencional em blocos "transporte – tratamento – montagem – finalização", levando a uma granularidade baixa. Os EPS/EAS focam-se no baixo nível baseado nas características do processo: O nível de operação fica dependente da modularidade, atingindo-se assim um nível elevado de granularidade.

Uma comparação detalhada entre os sistemas flexíveis (FMS) e os sistemas reconfiguráveis (RMS) pode ser encontrada em (Koren, Heisel et al. 1999; Hoda 2006; Mehrabi, Ulsoy et al. April 2002) e pode-se resumir na (Figura 2.3.1).

	RMS	FMS
Estrutura fisica	Ajustavel	Fixo
Focus do sistema	Familia dos componentes	Máquina
Escalabilidade	Sim	Sim
Flexibilidade	Personalizada	Geral
Operação em simultaneo	Sim	Não

Figura 2.3.1 – Comparação entre os sistemas FMS e RMS

As diferenças entre os HMS e os EPS são descritas em (Barata, Onori et al. 2007).

O conceito Holónico foi desenvolvido pelo filósofo Arthur Koestler para explicar a evolução dos sistemas sociais e biológicos (Koestler 1989). Ele propôs o termo "*Holon*" como o elemento básico dos Sistemas Holónicos de Manufactura (HMS). Este termo deriva do grego "holos" que significa "tudo" com o sufixo "-on" que significa parte. Este conceito reflecte as tendências dos agentes do mundo real que agem como entidades autónomas, mas com capacidade de interagir para formar hierarquias organizadas dentro de sistemas.

A arquitectura PROSA é uma das mais conhecidas arquitecturas como sendo uma arquitectura que descreve os tipos de *holons* que podem ser encontrados num sistema HMS (Brussel, Wyns et al. 1998). Esta arquitectura descreve os seguintes tipos de agentes:

- **Produto**: É o agente com o conhecimento sobre o produto e o processo para o construir. Contém o modelo do produto;
- **Recurso**: Este agente representa os recursos de produção. Ele é responsável pela execução dos processos;
- **Ordenador**: Este agente representa uma determinada tarefa no sistema de manufactura. Ele é responsável pela execução de uma determinada tarefa cumprindo todos os requisitos.

Capítulo 3. Tecnologias de suporte

Neste capítulo irão ser descritas as tecnologias que deram suporte ao desenvolvimento deste trabalho e de que forma esse suporte foi implementado.

Os seguintes conceitos são descritos neste capítulo: Sistemas multi-agente; JADE; Protégé; OWL; SWRL; JESS.

3.1 Agentes

3.1.1 Introdução

Os agentes autónomos e os sistemas multi-agente representam um novo modo de analisar, desenhar e implementar sistemas computacionais complexos. O conceito de agente oferece um reportório poderoso de ferramentas, técnicas e metáforas que têm o potencial de inovar o modo como as pessoas conceptualizam e implementam muitos tipos de software.

A definição de o que é um agente está longe de ser consensual, podendo encontrar-se na literatura da especialidade um grande número de definições distintas entre si. Uma das possíveis razões para tal situação prende-se com o facto do termo "agente" não ser um termo exclusivo desta área científica mas sim um termo de uso comum com inúmeras aplicações, levando a diversas confusões no seu uso que não acontecem com outros termos específicos das respectivas áreas.

É prática comum definir um agente como um sistema computacional, situado num determinado ambiente, que é capaz de executar acções autónomas e flexíveis com o intuito de

alcançar objectivos pré-definidos. Existem, portanto, três conceitos fundamentais para definir um agente: interacção, autonomia e flexibilidade.

Um agente é interactivo uma vez que recebe dados sensoriais do ambiente em que se situa, podendo efectuar mudanças no mesmo. É autónomo, pois tem a capacidade de agir independentemente de qualquer intervenção humana. E é flexível atendendo a que é reactivo (toma conhecimento do ambiente e reage prontamente a mudanças), é proactivo (não apenas reage a mudanças, como também toma iniciativas orientadas aos seus objectivos) e social (consegue interagir com outros agentes ou entidades por forma a resolver não só os seus próprios problemas, como ajudar nas actividades de outros agentes).

Um sistema multi-agente é, então, um sistema desenhado e implementado como vários agentes que interagem entre si. Os sistemas multi-agente ajustam-se idealmente a situações onde existem vários métodos para resolver um determinado problema, várias perspectivas e/ou várias entidades que poderão resolver esse problema.

A interacção entre agentes pode ser feita através de cooperação (trabalhando em conjunto para atingir um objectivo comum), coordenação (organizando actividades de modo a evitar interacções prejudiciais e explorar interacções benéficas) e negociação (chegando a acordos aceites por todos os intervenientes).

Em suma, é a flexibilidade e o alto nível de interacção que distingue os sistemas multi-agente de outros tipos de software e faculta todo o poder deste paradigma.

3.1.2 Arquitecturas de Agentes

A arquitectura de agentes, refere-se, não só à do próprio agente, mas também à do sistema multi-agente, isto é, o modo como estão organizados os agentes dentro de um sistema e a forma como estão estruturados os seus relacionamentos e interacções. De um modo semelhante ao que acontece com as diversas arquitecturas de software, as arquitecturas dos agentes possuem determinadas características que permitem a avaliação da sua qualidade e eficácia. No que respeita aos agentes robóticos, a sua arquitectura alia a arquitectura de software com a arquitectura de hardware (componentes físicos e a sua interligação).

3.1.2.1 Arquitecturas Deliberativas

Neste tipo de arquitectura, os agentes actuam com pouca autonomia e são interpretados como sistemas baseados em conhecimento, onde as suas acções/decisões são executadas tendo

por base um raciocínio lógico. O agente possui uma representação interna do mundo e um estado mental explícito que pode ser modificado por alguma forma de raciocínio simbólico.

A (Figura 3.1.1) representa a forma como se traduz o mundo real através de uma descrição simbólica, utilizando a percepção para manter essa estrutura actualizada e o raciocínio (sobre essa mesma informação simbólica) para que seja possível executar as acções a cada instante.

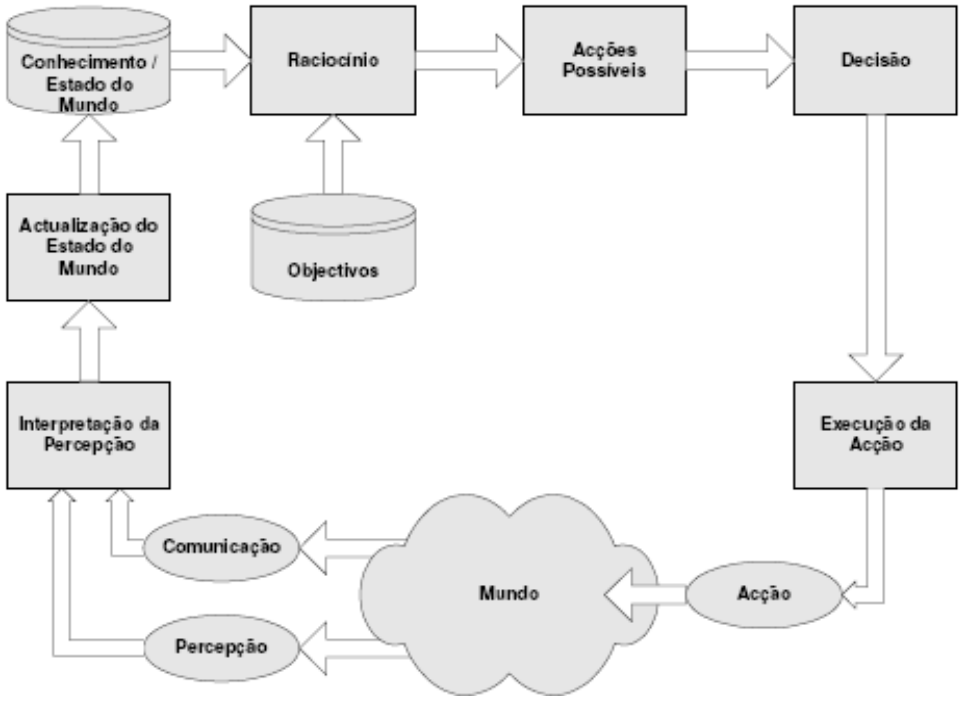

Figura 3.1.1 – Esquema genérico de uma arquitectura deliberativa.

3.1.2.2 Arquitecturas Reactivas

Este tipo de arquitectura (Figura 3.1.2) suporta a teoria de que o agente adquire inteligência através das interacções com o ambiente que o rodeia, não necessitando portanto de um modelo pré-estabelecido. As decisões são tomadas em tempo-real, com base num conjunto de informação muito limitado e regras simples de situação/acção que permitem seleccionar um certo tipo de comportamento.

Este tipo de arquitectura apresenta algumas vantagens em relação às restantes, destacando-se entre elas a simplicidade, a economia ou a boa robustez contra falhas. Em contrapartida, apresenta algumas desvantagens que tornam, por vezes, o seu uso inadequado, tais como o facto de os agentes decidirem com base na sua percepção actual, possuírem uma hierarquia pré-

definida e serem incapazes de realizar acções que impliquem a execução de planos a longo prazo.

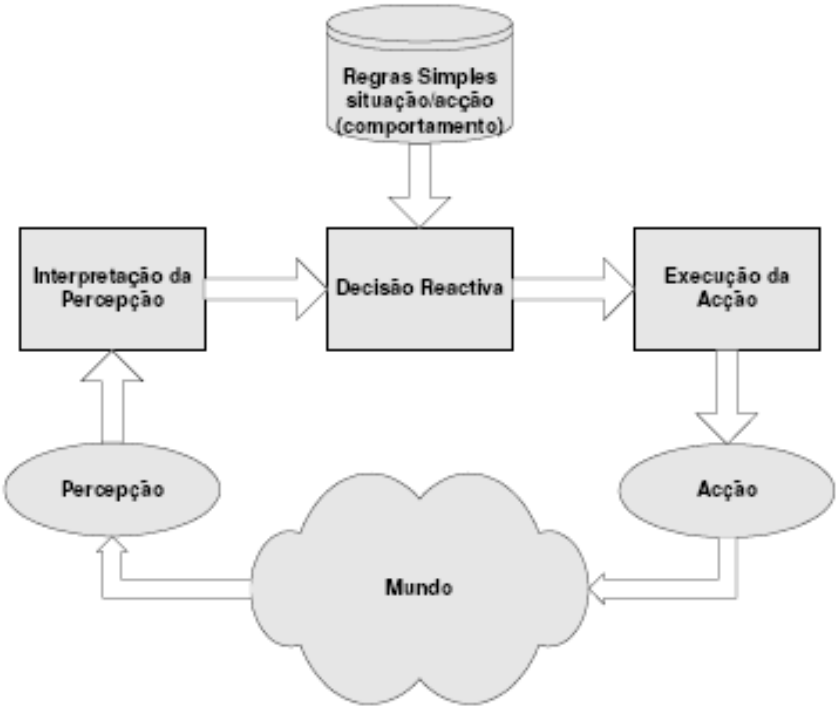

Figura 3.1.2 – Esquema genérico de uma arquitectura reactiva.

3.1.2.3 Arquitecturas Híbridas

Esta arquitectura (Figura 3.1.3) surge como alternativa às limitações existentes nas arquitecturas reactiva e deliberativa. Efectivamente, os agentes puramente reactivos não são capazes de implementar um comportamento orientado a objectivos. Por outro lado, os agentes deliberativos tornam-se muitas vezes incapazes de responder rapidamente aos estímulos do exterior, isto é, têm um tempo de reacção lento. Um agente híbrido combina as duas componentes e caracteriza-se por uma arquitectura composta por níveis ou camadas, dispostas hierarquicamente e onde normalmente a camada reactiva tem prioridade sobre a camada deliberativa, de modo a permitir uma resposta rápida aos eventos mais importantes registados no ambiente.

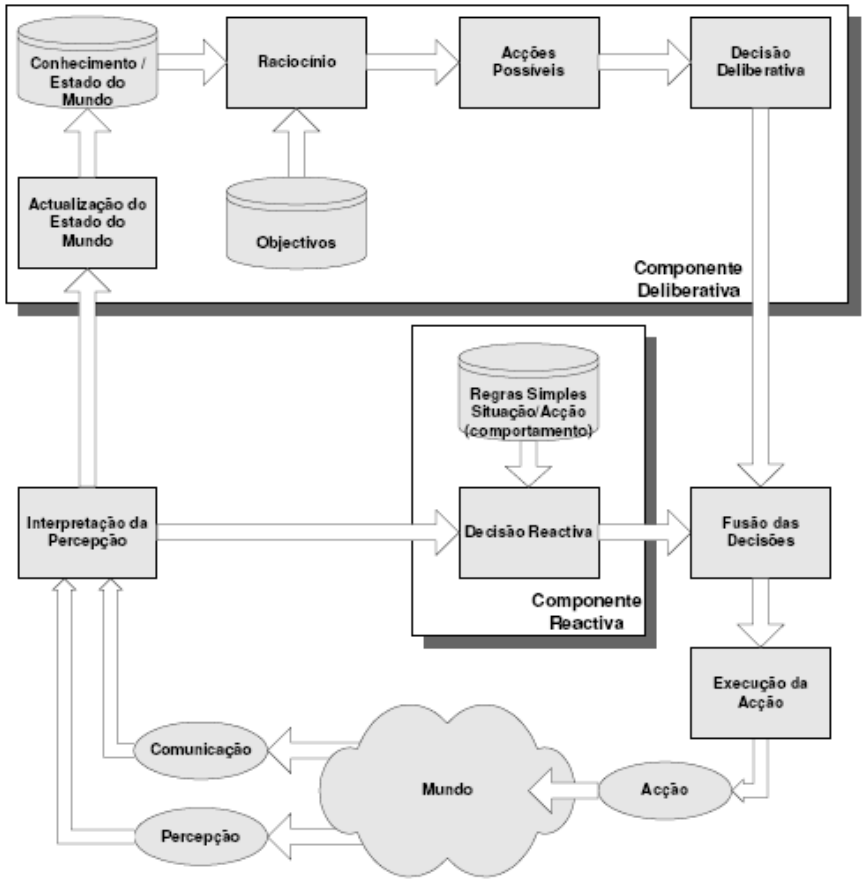

Figura 3.1.3 – Esquema genérico de uma arquitectura híbrida.

Após a interpretação da percepção e da comunicação provenientes do ambiente, o agente possui duas componentes: uma componente reactiva e uma componente deliberativa. O funcionamento da componente reactiva é extremamente simples e baseia-se num conjunto de regras situação/acção que associam directamente certas decisões a determinados estímulos. A componente deliberativa implica a construção de um estado simbólico do mundo e a utilização de raciocínio simbólico de forma a decidir a cada instante as acções a executar, de forma a atingir os objectivos. Uma parte complexa e sujeita a um elevado número de trabalhos de investigação é a forma de efectuar a fusão das decisões deliberativa e reactiva como forma de seleccionar a acção final a executar. Note-se que existe uma constante interacção entre as duas componentes, sendo a componente deliberativa capaz de alterar as regras de situação/acção da componente reactiva e esta, em situações de emergência, tem a capacidade de se sobrepor à componente deliberativa.

3.1.2.4 Arquitecturas por camadas

Este tipo de arquitectura (Figura 3.1.4) é usualmente recorrente, nomeadamente pelos sistemas híbridos. Existem dois tipos de camadas, nomeadamente:

- Camadas horizontais e
- Camadas verticais.

No primeiro caso, cada camada actua como um agente (camadas de software ligadas directamente aos sensores de input e às acções de output).

Nas camadas verticais, os sensores de input e as acções de output têm, pelo menos, uma camada a separá-los.

Deste modo, a camada horizontal é vantajosa relativamente à vertical por ser conceptualmente mais simples. Assim, o número de camadas e o número de comportamentos que o agente suporta possuem uma relação de n para n. No entanto, na arquitectura horizontal existe obrigatoriamente uma camada extra que funciona como mediador e que tenta resolver o problema proveniente do facto de existir competição entre camadas. A introdução desta camada extra, apesar de resolver um problema importante, pode provocar um estrangulamento no processo de decisão do agente.

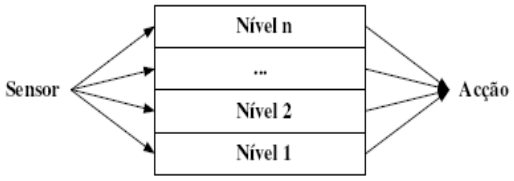

Figura 3.1.4 – Arquitectura de camadas horizontais.

Na arquitectura vertical este problema não ocorre, pois o fluxo de controlo passa sequencialmente por cada camada até à última, altura em que se executa a acção. No entanto, este método é pouco flexível pois, para o agente tomar uma decisão, é necessário que o fluxo de controlo atravesse todas as camadas. No caso de existirem falhas numa camada a acção actual do agente poderá ser anulada.

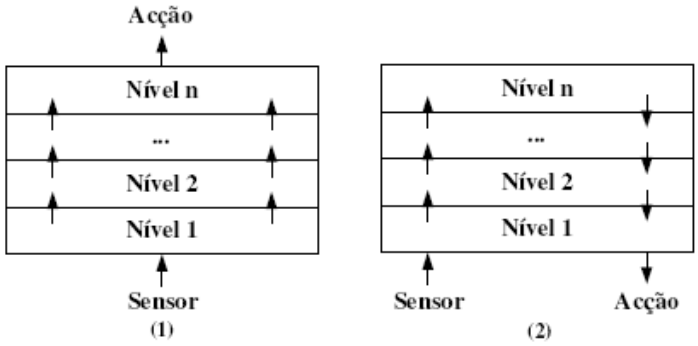

Figura 3.1.5 – Arquitectura de camadas verticais.

Analisando a (Figura 3.1.4), verifica-se uma única passagem de controlo. Contrariamente, na (Figura 3.1.5) observam-se duas passagens de controlo por cada camada. Nesta última arquitectura, o fluxo de informação sobe as várias camadas até atingir o nível superior e, seguidamente, desce atravessando novamente todas as camadas, dando origem à acção a executar.

3.1.3 Sistemas Multi-Agente

O desenvolvimento dos sistemas multi-agente tem como principais contribuintes os estudos sobre inteligência artificial, programação orientada a objectos, sistemas concorrentes e interface entre humanos e computadores.

Na inteligência artificial, quando se quer satisfazer um dado objectivo, é formulado um plano para esse objectivo, criando comportamentos para completar tarefas. Os comportamentos traduzem-se em máquinas de estado que estão constantemente a adquirir dados percepcionais, tomando acções sobre os mesmos, baseadas no estado actual do agente. Enquanto cada comportamento gera sugestões com respeito a que acção tomar, a decisão final é determinada pela interacção entre os vários comportamentos. Os comportamentos podem interagir de diversas maneiras e são organizados tipicamente em camadas hierárquicas, sendo as camadas inferiores correspondentes a comportamentos menos abstractos e as camadas superiores correspondentes a comportamentos mais abstractos.

Nos sistemas concorrentes e orientados a objectos, os objectos são definidos como entidades computacionais que encapsulam um determinado estado, são capazes de tomar acções ou métodos sobre esse estado e comunicam por troca de mensagens. Contudo, estes sistemas

diferem dos sistemas baseados em agentes, principalmente no que respeita à autonomia, ou seja, um objecto pode controlar o seu próprio estado, contudo, não pode controlar os seus comportamentos. Assim, temos que no caso dos agentes, as decisões são tomadas pelo agente que recebe os pedidos enquanto nos objectos, a decisão é tomada pelo objecto que invoca o método

Na interacção humano-computador somos capazes de interagir com a aplicação por intermédio de uma interface (manipulação directa). Contudo, por vezes torna-se desejável ter programas que tomem a iniciativa independentemente de qualquer intervenção exterior. Esta visão leva a pensar em programas computacionais como cooperativos e não apenas como simples servos.

Juntando todos estes aspectos leva-nos ao conceito de Sistema Multi-Agente, que pode ser definido como um conjunto de aplicações capazes de resolver problemas, ligados em rede e que trabalham em conjunto para atingir os objectivos pré-definidos que estão acima da capacidade ou conhecimento de cada aplicação individual. Essas aplicações (agentes) são autónomas e heterogéneas.

Em Sistemas Multi-Agente, cada agente tem informações ou capacidades incompletas para resolver o problema, ou seja, cada agente tem um ponto de vista limitado; não existe um sistema global de controlo, os dados estão descentralizados e a computação é assíncrona.

O interesse neste tipo de sistemas tem como base a sua robustez e eficiência, interoperacionalidade de sistemas legados existentes e a habilidade de resolver problemas quando os dados, conhecimento ou controlo estão descentralizados.

3.1.4 Linguagens de Comunicação entre agentes

3.1.4.1 KQML

A linguagem KQML (*Knowledge Query and Manipulation Language) (Tim, Richard et al. 1994)* consiste num protocolo de comunicação de alto nível para troca de mensagens independente do conteúdo e da ontologia aplicável. Neste caso, não há uma preocupação com o conteúdo da mensagem mas sim com a especificação da informação necessária à compreensão desse conteúdo.

Alguns dos problemas desta linguagem são:

- Ambiguidade e termos vagos
- Performativas com nomes inadequados
- Falta de performativas

Devido às limitações acima mencionados alguns autores afirmam que KQML provavelmente será substituído por FIPA-ACL, a qual será descrita de seguida. Na (Figura 3.1.6) apresenta-se um exemplo do formato de uma mensagem KQML.

```
(ask-all
  :content "price(ibm,(Price,Time))"
  :receiver stock-server
  :language standard_prolog
  :ontology NYSE-TICKS)
```

Figura 3.1.6 – Exemplo de uma mensagem KQML adaptada de (Tim, Richard et al. 1994)

3.1.4.2 FIPA-ACL

A FIPA-ACL (FIPA 2002) (*Foundation for Intelligence Phisical Agents – Agent Communication Language*) é uma linguagem, tal como o KQML, baseada em acções de fala. A sua sintaxe é bastante semelhante ao KQML, porém o conjunto de performativas é diferente. A sua especificação caracteriza-se por um conjunto de tipos de mensagens e descrições dos efeitos da mensagem sobre os agentes que a enviam e sobre o que a recebem. Possui uma semântica definida precisamente com uma linguagem de descrição de semântica.

A (Figura 3.1.7) apresenta a estrutura de uma mensagem em FIPA – ACL.

```
(communicative act
        :sender <valor>
        :receiver <valor>
        :content <valor>
        :language <valor>
        :ontology <valor>
        :conversation-id<valor>
        ...)
```

Figura 3.1.7 – Formato de uma mensagem FIPA-ACL.

Na (Figura 3.1.8) apresenta-se o conjunto de mensagens existentes na linguagem FIPA-ACL.

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

Figura 3.1.8 – Mensagens existentes na linguagem FIPA-ACL

FIPA-*request*

Este protocolo (FIPA 2002) permite a um agente pedir a outro para executar uma acção. O agente que recebe o pedido pode aceitá-lo ou pode recusá-lo mesmo sendo capaz de o executar. Este protocolo suporta todo um conjunto de *outcomes* provenientes da interacção: o agente que inicia o protocolo envia uma mensagem *request* para o destinatário com a acção que quer que este execute. Se o destinatário não souber nada sobre a acção que o emissor quer, envia a resposta com *not-understood*. Caso entenda, então o agente decide se quer executar a acção. Caso for este o caso, envia uma resposta com *agree*, caso contrário, envia uma mensagem *refuse*.

Depois do acto comunicativo de *agree*, o destinatário tenta executar a acção. Caso consiga envia um *inform*, senão envia um *failure*.

FIPA-*contract-net*

É o primeiro dos protocolos de interacção de alto nível do FIPA (FIPA 2002). Enquanto os outros protocolos são ponto a ponto e usam duas trocas de mensagens, o *FIPA-contract-net* é um para muitos e usa quatro trocas de mensagens. Um agente, denominado *manager*, usa este protocolo quando quer que uma acção seja executada mas falta-lhe o conhecimento prévio de um agente capaz de a executar. O tal agente pode então iniciar o protocolo enviando uma mensagem *call-for-proposal*, ou CFP, a outros agentes. Haverá agentes que responderão *not-understood* ou mesmo *refuse*. Essas mensagens são eliminadas. Outros responderão com uma mensagem de *propose*, ou seja, uma proposta para executar a acção. Nela estarão implícitas as condições que o agente quer em troca da execução. O agente que iniciou o protocolo pondera sobre as propostas e caso alguma lhe agrade, de acordo com certos critérios, envia a mensagem *accept-proposal* ao agente que fez essa proposta, e um *reject-proposal* a todos os outros que enviaram propostas.

3.1.5 JADE

3.1.5.1 Caracterização do JADE

O *Jade* (***J**ava **A**gent **DE**velopment framework) (Bellifemine, Poggi et al. 1999)* consiste num ambiente para desenvolvimento de aplicações baseada em agentes de acordo com as especificações da FIPA (***F**oundation for **I**ntelligent **P**hysical **A**gents*) para interoperabilidade entre sistemas multi-agentes totalmente implementado em Java.

O principal objectivo do Jade é simplificar e facilitar o desenvolvimento de sistemas multi-agentes garantindo um padrão de interoperabilidade entre os mesmos através de um abrangente conjunto de serviços, os quais tanto facilitam como possibilitam a comunicação entre agentes, de acordo com as especificações da FIPA: serviço de nomes (*naming service*), páginas amarelas (*yellow-page service)*, transporte de mensagens, serviços de codificação e descodificação de mensagens e uma biblioteca de protocolos de interacção pronta para ser usada. Toda a comunicação entre agentes é feita por troca de mensagens. Além disso, lida com todos os aspectos que não fazem parte do agente em si e que são independentes das aplicações tais como transporte de mensagens, codificação e interpretação de mensagens e ciclo de vida dos agentes. O Jade pode ser considerado como um "*middleware*" de agentes que implementa um *framework* de desenvolvimento e uma plataforma de agentes. O *Jade* foi desenvolvido na linguagem *Java* devido a características particulares desta linguagem, particularmente pela programação orientada a objectos em ambientes distribuídos heterogéneos.

Na (Figura 3.1.9) ilustra-se a estrutura do JADE.

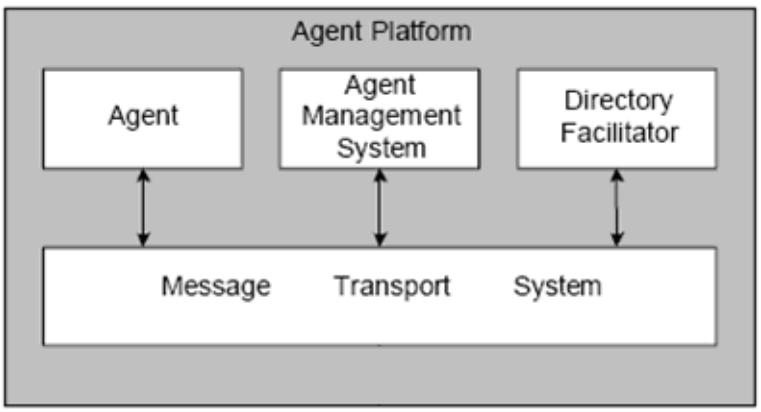

Figura 3.1.9 – Estrutura interna do JADE adaptada de (Bellifemine, Caire et al. 18-June-2007)

3.1.5.2 Características do JADE

De seguida descrevem-se as principais características do JADE:

- Plataforma distribuída de agente: JADE pode ser dividido em vários "*hosts*" ou máquinas (desde que eles possam ser ligados via RMI). Apenas uma aplicação Java e uma *Java Virtual Machine* é executada em cada *host*. Os agentes são implementados como *threads* Java e inseridos dentro de repositórios de agentes chamados *containers* (*Agent Containers)* que fornecem suporte para a execução do agente.
- GUI (Graphical User Interface): Interface visual responsável pela gestão de vários agentes e *containers* de agentes.
- Ferramentas de Debugging: Ferramentas que ajudam o desenvolvimento de aplicações multi-agentes baseados em JADE.
- Suporte a execução de múltiplas, paralelas e concorrentes actividades de agentes – através dos modelos de comportamentos (Behaviours).
- Ambiente de agentes complacente a FIPA: No qual se incluem o sistema gestor de agentes (AMS – Agent Management System), o DF (Directory Facilitator) e o canal de comunicação entre agentes (ACC – Agent Communication Channel). Todos esses três componentes são automaticamente carregados quando o ambiente é iniciado.
- Transporte de mensagens: Transporte de mensagens no formato FIPA-ACL (FIPA 2002) dentro da mesma plataforma de agentes.
- Biblioteca de protocolos FIPA: Para interacção entre agentes JADE, dispondo de uma biblioteca de protocolos prontos para ser usados.
- Automação de registos: Registo e cancelamento automático de agentes com o AMS fazendo com que o desenvolvedor se abstraia disso.
- Serviços de nomes (Naming Service) em conformidade aos padrões FIPA: Durante a inicialização dos agentes, estes obtêm seus GUID (Globally Unique IDentifier) da plataforma que são identificadores únicos em todo o ambiente.
- Integração: Mecanismo que permite aplicações externas carregar agentes autónomos JADE.

3.1.5.3 Arquitectura interna do JADE

A arquitectura da plataforma JADE (Figura 3.1.10) baseia-se na coexistência de várias máquinas virtuais Java (*Java Virtual Machine* – JVM) podendo ser distribuída por diversas máquinas independentes de sistema operacional que cada uma utiliza. Na Figura 3.1.10 existe uma visão distribuída da plataforma de agentes JADE dividida em 3 *hosts.* Em cada *host* existe uma JVM executando um JRE (***J**ava **R**un-time **E**nviroment) e* possuindo cada uma delas um *container* de agentes que fornece um ambiente completo para execução destes, além de permitir que vários elementos possam correr concorrentemente no mesmo processador (*host)*. Ou seja, durante a execução deve existir uma JVM por processador, sendo possível coexistir vários agentes por JVM. A comunicação entre JVMs é realizada através da invocação remota de métodos (*RMI – **R**emote **M**ethod **I**nvocation)* do Java.

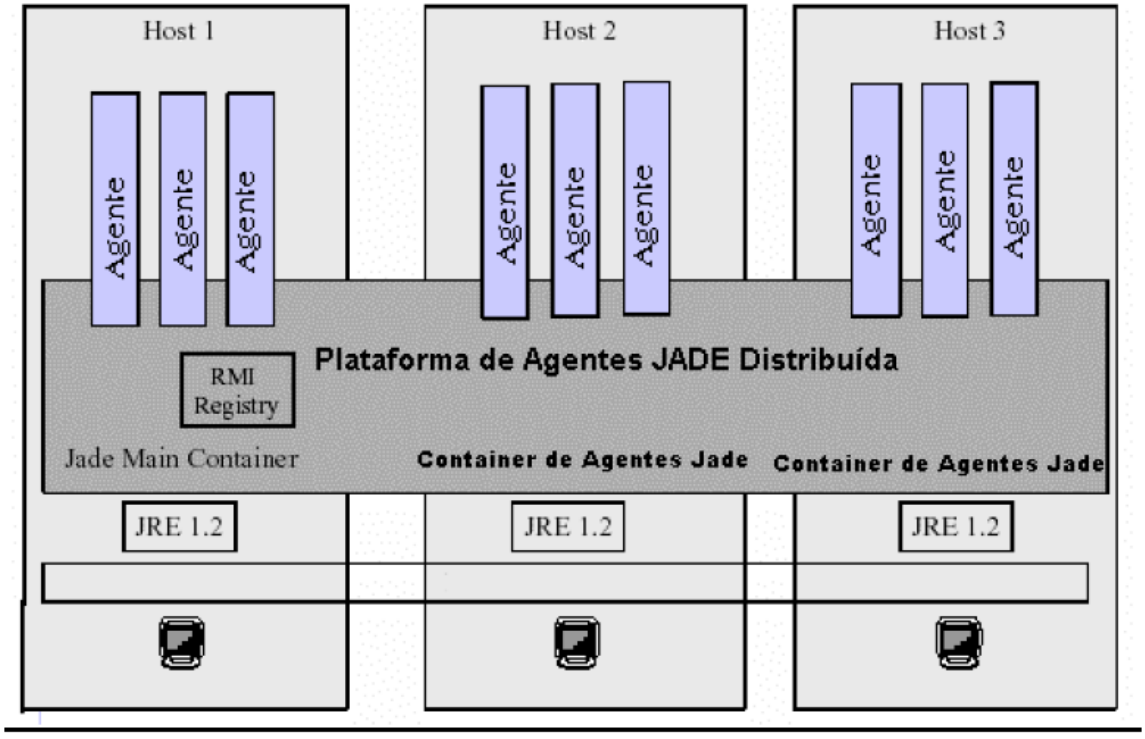

Figura 3.1.10 – Plataforma de agentes JADE distribuída em vários Hosts adaptada de (Bellifemine, Caire et al. 18-June-2007)

O *main-container,* localizado no *host 1* da figura acima (Figura 3.1.10), designa-se por *container* onde se encontra o AMS, o DF e registo RMI. Esse registo RMI é um servidor de nomes que o Java usa para registar e recuperar referências a objectos através do nome. Ou seja, é o meio que JADE usa em Java para manter as referências aos outros *containers* de agentes que se

ligam à plataforma. Os outros *containers* de agentes conectam-se ao *main-container* fazendo com que quem devolve fique abstraído da separação física dos *hosts*, caso exista, ou das diferenças de plataformas que cada *host* possa ter. Essa abstracção é ilustrada na figura anterior na zona central na qual a plataforma JADE integra todos os três *hosts* actuando como um elo de ligação e fornecendo um completo ambiente de execução para qualquer conjunto de agentes JADE.

3.2 Ontology Web Language (OWL)

3.2.1 Ontology Web Language

As linguagens de ontologias permitem a construção de ontologias através da codificação do conhecimento sobre um domínio específico onde muitas vezes incluem regras de processamento desse mesmo conhecimento. Estas linguagens são, normalmente, baseadas na sua lógica de primeira ordem ou na lógica descritiva. A definição destas lógicas é feita no próximo subcapítulo.

As linguagens mais conhecidas, onde a codificação do conhecimento é feita através de XML são:

- DARPA Agent Markup Language (DAML+OIL);
- Ontology Inference Layer (OIL);
- Ontology Web Language (OWL);
- Resource Description Framework (RDF);
- SHOE.

A construção da ontologia elaborada neste trabalho foi focada na utilização da linguagem OWL-DL (Bechhofe, Harmelen et al. 10 Fev, 2004) (Ontology Web Language – Description Logic) que é uma linguagem, tal como descrito mais à frente, com influências das outras linguagens, aproveitando assim os pontos fortes das diversas linguagens e visto permitir introduzir mais complexidade nas classes que não é possível utilizando OWL-lite.

OWL-DL é baseado numa linguagem descritiva, permitindo assim criar um modelo hierárquico de forma automática e ser possível, também de forma automática, o processamento do conhecimento ("*reasoning*") da ontologia, que não é possível ser executado quando se utiliza OWL-full, apesar de esta última ser a linguagem mais expressiva delas todas. Mas o grau de expressividade oferecido por OWL-DL é mais que suficiente para o caso estudado. OWL apresenta também algumas funcionalidades que não estão presentes nas outras linguagens, como é o caso de operações lógicas (e.g. *and, or* and *not*).

Ontology Web Language (OWL) é uma nova linguagem para a Web Semântica, desenvolvida pelo Consórcium da *World Wide Web* (W3C). Começou por ser uma linguagem

desenhada para representar a informação de objectos de forma categórica e a forma como esses objectos estão relacionados entre eles (Horrocks, Patel-Schneider et al. 2003).

A linguagem OWL não foi a primeira linguagem de ontologias virada para a Web e o seu desenho teve influências de diversas linguagens já existentes, tais como RDFS, SHOE, OIL, DAML-ONT e DAML+OIL (Bechhofer, Goble et al. 2001). Esta última linguagem referida foi a que mais influenciou a construção da linguagem OWL. Por sua vez, a linguagem DAML+OIL foi fortemente influenciada pela linguagem OIL (OIL) que adicionalmente foi influenciada pelos trabalhos de DAML-ONT e RDFS. As classes OWL podem ser específicas como combinações lógicas (intersecções, uniões ou complementos) de outras classes estendendo as capacidades de RDFS. A maior extensão feita pelo RDFS para o OWL foi a capacidade de se definir restrições do comportamento de propriedades que pertencem a uma determinada classe.

O modelo lógico que compõe a linguagem OWL permite o uso de um *"Reasoner"* para processar o conhecimento da ontologia. Este processamento serve não só para verificar se todas as definições estão feitas de uma forma consistente, reconhecendo que conceitos pertencem a que definições mantendo uma hierarquia correcta na ontologia, mas também irá servir como motor de inferência de conceitos.

Um *reasoner* é um programa de software que tem a capacidade de inferir consequências lógicas através de um conjunto de factos já existentes. Ou seja, através das classes existentes na ontologia, este programa de software tem a capacidade de verificar se toda a hierarquia existente se encontra coerente.

As ontologias OWL podem ser subdivididas em três categorias consoante a sua expressividade. As subcategorias são: OWL-Lite, OWL-DL e OWL-Full.

OWL-Lite é a sublinguagem menos expressiva, sendo utilizada em ontologias onde a estrutura hierárquica das classes é simples e onde as restrições são simples (por exemplo, cardinalidade apenas de 0 e 1).

OWL-DL suporta a máxima expressividade garantindo que todas as condições existentes são computacionais. Ou seja, todas as definições existentes na ontologia podem ser processadas por computador e esse processamento irá finalizar num tempo limitado (McGuinness and Harmelen 10 Fev, 2004).

OWL-Full suporta a máxima expressividade, tal como o OWL-DL, mas também tem toda a liberdade sintáxica do RDF sem quaisquer garantidas computacionais. Este facto faz com que

não existam garantias de um *reasoner* suportar na totalidade todas as funcionalidades de uma ontologia.

3.2.2 Lógica Descritiva (DL)

A investigação na área da representação do conhecimento está, na maioria dos casos, focada em fornecer descrições de alto nível do mundo em que se aplicam no sentido de desenvolver aplicações inteligentes. Neste contexto, a palavra "inteligente" refere-se à habilidade de um sistema em encontrar consequências implícitas através do seu conhecimento explícito. Este tipo de sistemas denominam-se sistemas baseados em conhecimento (do inglês *Knowledge-based systems*) (Nardi and Brachman 2002).

A Lógica Descritiva é o nome mais recente de um formalismo de representação de conhecimento sobre um determinado domínio através da definição dos seus conceitos relevantes e utilizando estes mesmos conceitos, especificar propriedades e indivíduos que pertençam a esse domínio.

Tal como o nome Lógica Descritiva indica, uma das principais características destas linguagens é que têm uma semântica formal, baseada em lógica. Outro aspecto que distingue esta linguagem das outras é o facto de dar ênfase à inferência de conhecimento como um serviço central. Esta inferência permite inferir conhecimento, de forma implícita, através do conhecimento que se encontra explícito, definido na base de conhecimento.

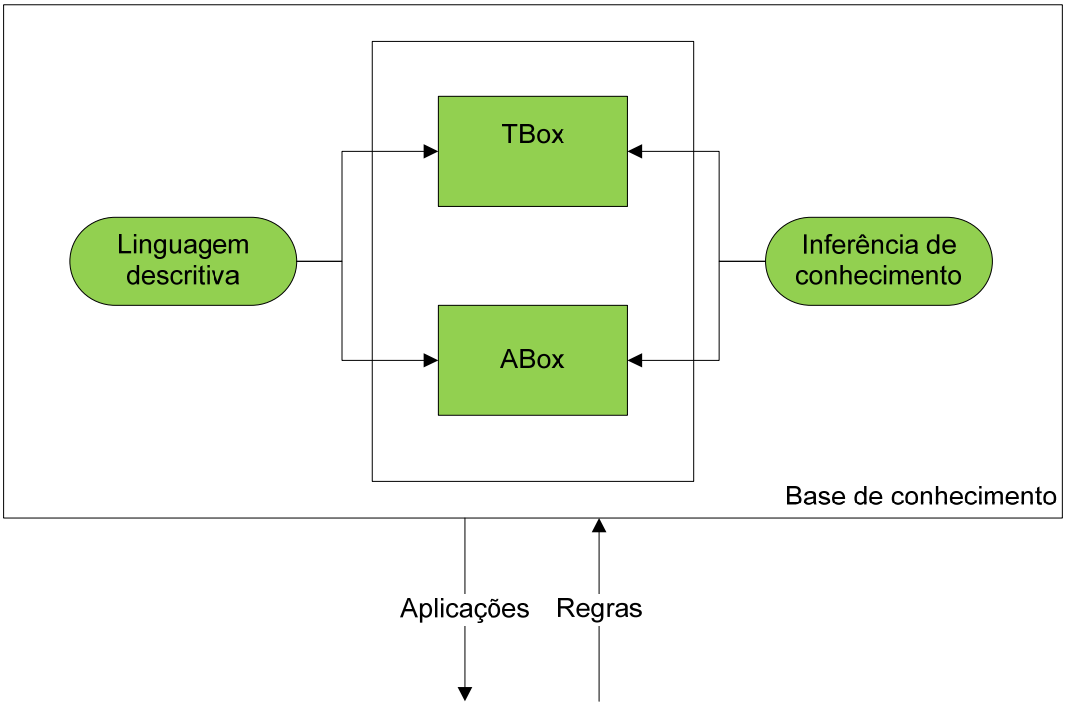

Figura 3.2.1 – Arquitectura da Linguagem Descritiva

Uma representação do conhecimento baseado em Lógica Descritiva fornece métodos para definir bases de conhecimentos, para inferir consequências lógicas sobre o seu conteúdo e para manipular estas bases. Na figura (Figura 3.2.1) é apresentada a arquitectura de um sistema baseado em Lógica Descritiva. Uma base de conhecimento é composta por dois componentes: *TBox* e *ABox*. A *TBox* é responsável pela terminologia (*TBox*, do inglês *terminology*) do sistema, ou seja, por todo o vocabulário existente no domínio de uma aplicação. A *ABox* (do inglês *assertation*) contém afirmações sobre os indivíduos no que respeita ao vocabulário da *TBox* (Baader and Nutt 2002). Por outras palavras, a *ABox* contém o conhecimento sobre os indivíduos do domínio em questão. A *TBox* pode ser vista como tendo o conhecimento sobre o sistema em si (conhecimento explicito), e a *ABox* como tendo o conhecimento implícito que varia ao longo do tempo, dependendo do estado em que o sistema se encontre. A *ABox* estende o conhecimento da *TBox*.

O vocabulário consiste em conceitos que estão presentes nos conjuntos de indivíduos e por relações binárias entre estes indivíduos. Estes sistemas, além dos conceitos atómicos, permitem definir descrições complexas desses mesmos conceitos. O componente *TBox* pode ser utilizado para atribuir nomes a essas descrições complexas. A linguagem para definir as descrições é uma característica específica de cada sistema (Baader and Nutt 2002).

A Lógica Descritiva expressa a sua semântica através da Lógica descritiva de primeira ordem (FOL, do inglês *First Order Logic*) (Nardi and Brachman 2002; Ait-Kaci 2007) que

também é uma linguagem de representação do conhecimento. A Lógica Descritiva pode ser obtida através da restrição da sintaxe das fórmulas da Lógica de Primeira Ordem a duas variáveis, visto que os predicados em DL são simples ou duplos. Outra característica da DL está relacionada com a restrição numérica que não existe na FOL, bem como o tratamento de Indivíduos na *ABox* que também não é feita quando se utiliza FOL (Nardi and Brachman 2002).

Um exemplo de uma linguagem baseada em Lógica Descritiva é a linguagem utilizada no desenvolvimento desta ontologia, apresentada no subcapítulo 3.1, designada por *Ontology Web Language – Description* Logic (OWL-DL).

3.3 Protégé

Protégé (John, Mark et al. 2003) é um programa livre de edição de ontologias desenvolvido pelo departamento de Medicina Informática da Universidade de Stanford. Apesar de historicamente ter sido desenvolvido para aplicações biomédicas, o sistema é independente do domínio onde se pretende trabalhar, podendo ser utilizado em qualquer área (como na área da manufactura). Este software foi utilizado para a criação de toda a ontologia implementada nesta tese.

A arquitectura do Protégé está separada em duas componentes. A componente do módulo (Protégé API) e a componente de vista (Protégé GUI). Na primeira componente situa-se o mecanismo de representação interna de ontologias e bases de conhecimento. A segunda componente tem como função servir de interface com o utilizador de modo a que este consiga visualizar e manipular a ontologia (Knublauch, Fergerson et al.).

A API do Protégé representa ontologias através de classes, propriedades, características dessas propriedades e instâncias.

Na figura (Figura 3.3.1) pode ser visto o ambiente gráfico do Protégé onde é possível visualizar uma ontologia.

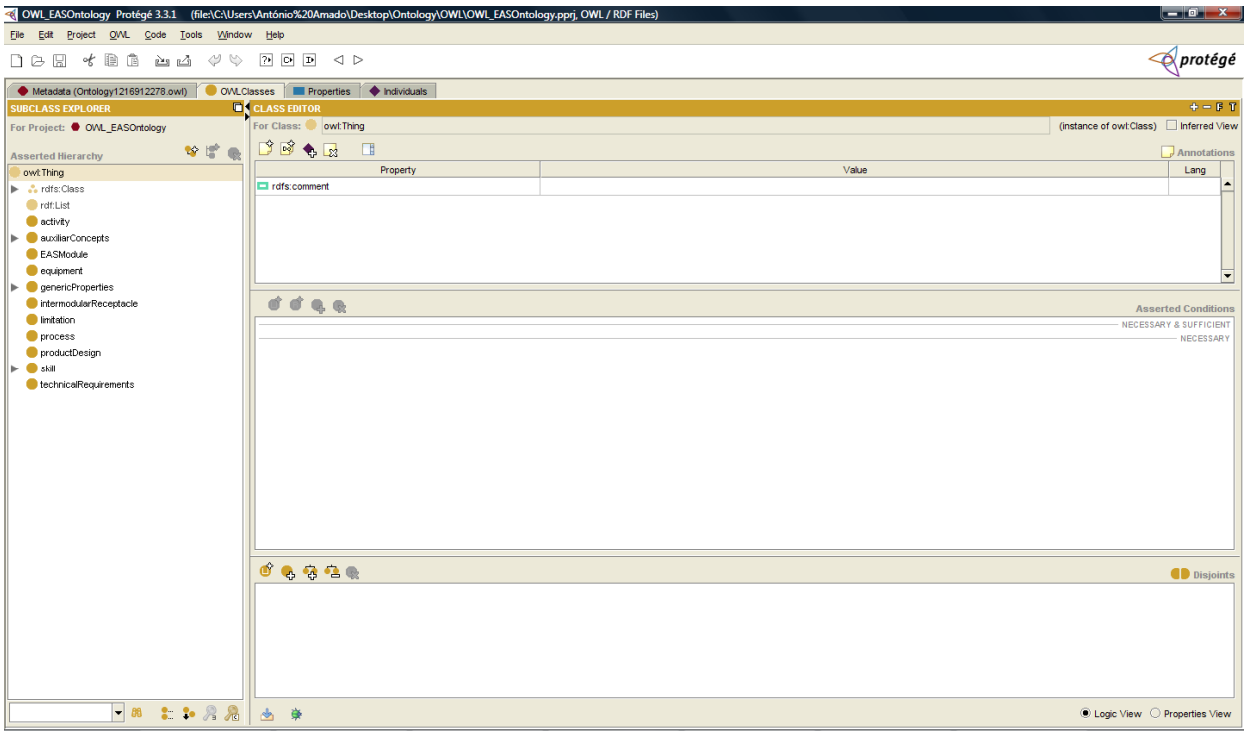

Figura 3.3.1 – Protégé OWL 3.3.1

3.3.1 Protégé OWL Plug-in

De modo a ser possível a construção de ontologias no Protégé utilizando a linguagem OWL, é necessário utilizar um *plug-in* próprio que fica integrado no Protégé.

Este *plug-in* estende a arquitectura do Protégé, tal como mostrado na figura (Figura 3.3.2), de modo a permitir ao utilizador editar ficheiros OWL bem como bases de dados (Knublauch, Fergerson et al.). A inclusão deste *plug-in* permitiu desenvolver a ontologia recorrendo à linguagem OWL tal como pretendido.

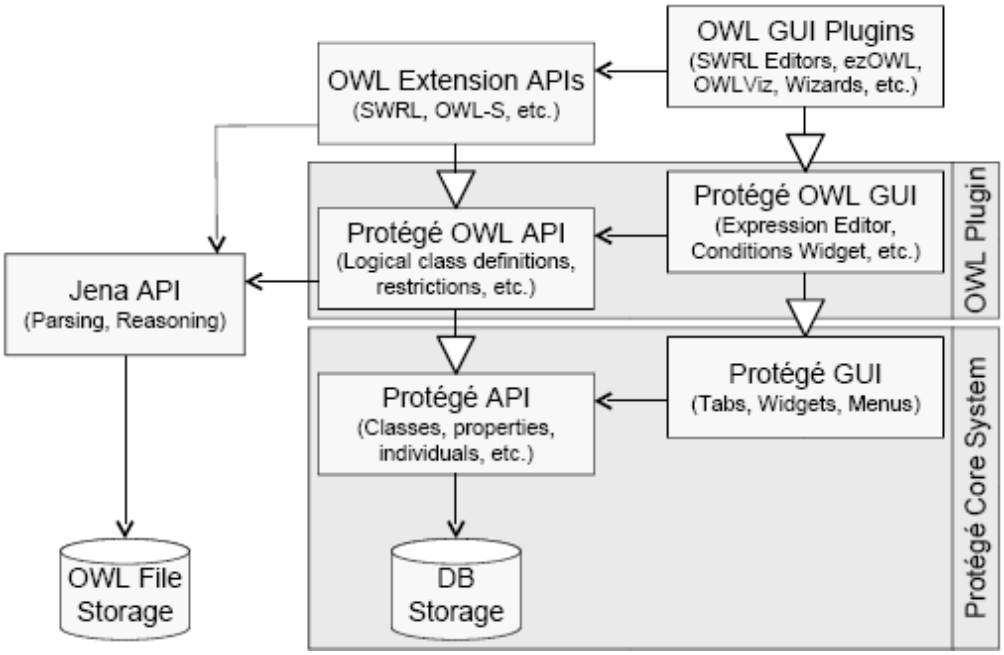

Figura 3.3.2 – Arquitectura do plug-in OWL adaptada de (Knublauch, Fergerson et al.)

Na figura é possível ver o núcleo do Protégé separado pelos seus dois componentes: Protégé API e Protégé GUI, que corresponde, respectivamente, à componente do módulo e à componente de vista, acima referidas. A interagir com este núcleo está o plug-in do OWL tal como já foi referido. Tal como o Protégé, também este plug-in se encontra dividido em duas componentes, uma referente ao módulo e outra à interface gráfica. Este plug-in suporta linguagens como *Resource Description Framework* (RDF), OWL Lite e OWL Full (não na sua totalidade).

Apesar de o núcleo do Protégé conter funções para aceder a classes, propriedades e instancias, o OWL plug-in dispõe de outras funções, também em Java, para aceder e manipular ontologias OWL. Estas funções são específicas para as várias classes OWL.

3.4 Java Expert System Shell (JESS)

Os sistemas peritos (*Expert Systems*) pertencem ao ramo da inteligência artificial e têm como objectivo simular o comportamento humano. Existem vários tipos de sistemas peritos:

- Redes neuronais;
- Sistemas *Blackboard*
- Redes Bayesian
- Sistemas de inferência baseada em casos
- Sistemas baseados em regras

O JESS é um sistema perito e é baseado em regras. Este sistema consiste numa base de regras, numa base de factos e num motor de execução (Figura 3.4.1). O motor de execução irá fazer corresponder factos da base de factos às regras da base de regras. Estas regras podem inferir novos factos que irão ser postos na base de factos ou podem simplesmente executar funções feitas em Java.

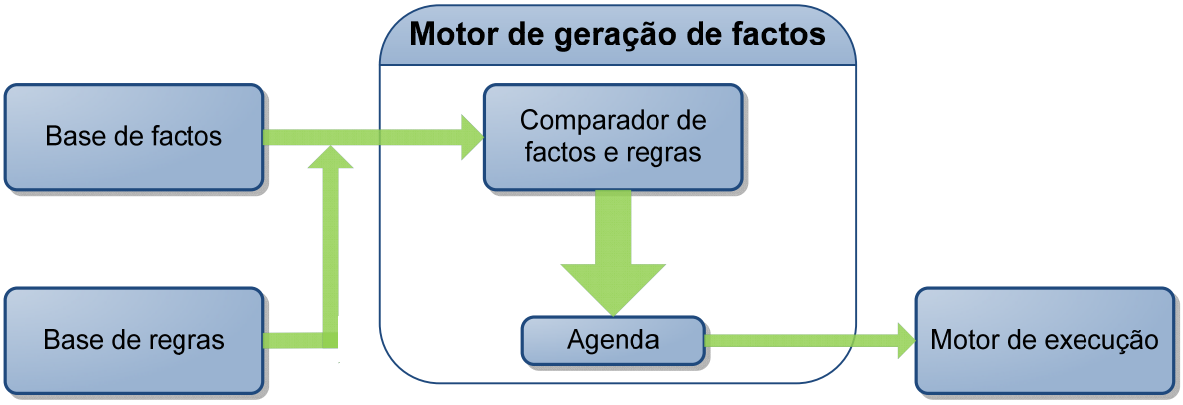

Figura 3.4.1 – Arquitectura do JESS

A base de factos é onde estão armazenados todos os factos presentes na ontologia OWL que são previamente carregados através de uma ponte de ligação entre o JESS e a linguagem *Semantic Web Rule* Language – SWRL (descrita no capítulo seguinte). A base de regras contém todas as regras definidas através da linguagem SWRL que irão servir de base de comparação no motor de geração de factos. Este motor de geração de factos vai comparar os factos com as regras e irá agendar (através da Agenda) uma execução. Esta execução tanto pode servir para lançar funções programadas em Java como pode servir para gerar novos factos e adicioná-los à base de factos já existente, fazendo com que o sistema vá evoluindo ao longo do tempo e

adquirindo novos conceitos (factos) que se adaptem a qualquer evolução que tenha ocorrido no sistema.

Sendo o JESS um processador de regras, este foi incluído no projecto para ser possível processar possíveis regras criadas através da linguagem SWRL (descrita mais à frente). Além de processá-las, O JESS também vai actualizar a ontologia já existente. Entenda-se por actualização, a manipulação (alteração; adição e remoção) das classes da ontologia.

3.5 Semantic Web Rule Language (SWRL)

A Web Semântica tem como um dos seus grandes objectivos a interoperabilidade. O desenvolvimento de uma linguagem de partilha de regras é um grande passo para se atingir esse objectivo (O'Connor, Knublauch et al. 2005). A *Semantic Web Rule Language* (ou SWRL) (Horrocks, Patel-Schneider et al. 2004) é o resultado de uma dessas linguagens. Esta linguagem foi desenvolvida principalmente para interoperar com o *Protégé-OWL*.

Esta linguagem é baseada numa combinação entre sublinguagens da linguagem OWL, OWL-DL e OWL-Lite. SWRL permite, aos seus utilizadores, escrever regras em termos de conceitos OWL que irão inferir consequências lógicas nos indivíduos OWL presentes na ontologia.

As regras SWRL são compostas por dois pares, o par antecedente e o par consequente. O par antecedente é referido como o corpo da regra, e o par consequente é referido como a cabeça da regra. Tanto a cabeça como o corpo da regra consistem na conjunção de um ou mais Átomos.

O *Protégé SWRL Editor* é uma extensão do *Protégé-OWL* que permite aos seus utilizadores criarem, editarem, eliminarem regras SWRL. Este editor permite o acesso directo às classes OWL, às propriedades e aos indivíduos presentes na ontologia que se está presentemente a editar. Também é possível aceder a um conjunto de funções específicas presentes na Ontologia SWRL (designada por *SWRL Built-in Ontology*) (Figura 3.5.1 – Ontologia SWRL).

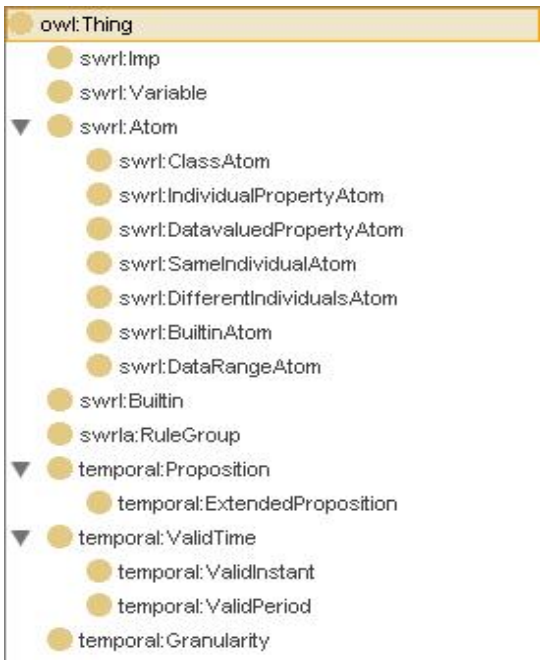

Figura 3.5.1 – Ontologia SWRL

Todas as regras são introduzidas na forma de texto. O editor de regras SWRL verifica tanto sintáctica como semanticamente o texto que se está a editar.

As regras são guardadas como indivíduos OWL na altura em que se guarda um projecto OWL com a ontologia que lhe está associada. As classes que descrevem estes indivíduos pertencem à ontologia SWRL (Figura 3.5.1). A classe de mais alto nível denomina-se *SWRL:Imp* e representa uma única regra SWRL. A cabeça e o corpo da regra acima referidos são ambos instâncias da classe *SWRL*.A ontologia SWRL inclui uma classe designada por *SWRL:Builtin* que representa todas as funções suportadas por esta linguagem. A classe *SWRL:Variable* é utilizada para representar as variáveis da linguagem.

Este editor SWRL inclui uma Java API designada por *Protégé SWRL Factory*, que permite a programadores aceder e manipular directamente regras SWRL numa ontologia OWL. Esta fábrica executa um mapeamento directo dos indivíduos OWL que representam regras SWRL para instâncias Java.

A fábrica SWRL é utilizada para integrar o motor de regras JESS com o editor SWRL. Através de JESS, os utilizadores podem executar as regras de forma a criarem novos conceitos OWL que posteriormente serão inseridos na ontologia OWL (O'Connor, Knublauch et al. 2005).

Esta linguagem foi utilizada para permitir a criação de regras a incluir na ontologia. Tanto o SWRL como o JESS foram escolhidos por existir uma boa integração entre estas duas tecnologias e a sua utilização ser relativamente fácil recorrendo às respectivas APIs disponibilizadas por ambas as tecnologias. O SWRL e o OWL também apresentam uma boa

relação entre elas. São duas linguagens que estão bastante bem interligadas e são 100% compatíveis, logo a escolha da linguagem SWRL.

Um exemplo muito simples da utilização de regras SWRL pode ser:

```
hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒ hasUncle(?x1,?x3)
```

Figura 3.5.2 – Exemplo de uma regra escrita em SWRL adaptada de (Horrocks, Patel-Schneider et al. 2004)

Esta regra utiliza uma combinação das propriedades "*hasParent*" e "*hasBrother*" implicando a regra "*hasUncle*". Ou seja, a combinação das duas primeiras propriedades, vai gerar a segunda propriedade. Os indivíduos x1 e x3, presentes na regra, não têm nenhuma relação explícita da ontologia OWL. Após a aplicação desta regra passariam a ter, através da propriedade *hasUncle*.

Um outro exemplo de regras que utiliza funções específicas da ontologia SWRL é:

```
hasBrother(?x1,?x2) ^ hasAge(?x1,?age1) ^
hasAge(?x2,?age2) ^ swrlb:greaterThan(?age2,?age1) →
hasOlderBrother(?x1,?x2)
```

Figura 3.5.3 – Regra SWRL com funções específicas da ontologia SWRL adaptada de (O'Connor, Knublauch et al. 2005)

Nesta regra é possível ver a aplicação da função "*greaterThan*". Esta função devolve o maior dos números que se encontram em argumento. Esta regra encontra-se definida na ontologia SWRL da figura (Figura 3.5.1) e está representada pela classe "*swrlBuiltIn*".

3.6 Pellet

O Pellet é um *Reasoner* livre, desenvolvido em Java, que foi criado como uma ferramenta capaz de provar os conceitos de um sistema para ajudar a ir ao encontro dos requisitos de implementação do W3C para a linguagem OWL (Sirin, Parsia et al. 2007).

O Pellet suporta funcionalidades como:

- Serviços de *Reasoning*:
    - Restrições de cardinalidade;
    - Sub-propriedades complexas de axiomas;
    - Propriedades Disjuntas;
    - Propriedades Reflexiveis, Irreflexiveis, Simétricas e Anti-simétricas
    - Partilha de vocabulário entre indivíduos, classes e propriedades;
    - Definição de propriedades negativas

O Pellet suporta todos os serviços standards de inferência que são suportados por *Reasoners* de DL:

- Verificação da consistência da ontologia – Assegura que a ontologia não contém factos contraditórios;
- Satisfação de conceitos – Determina se é possível uma determinada classe ter instâncias. Se uma classe não for satisfeita, definir uma instância para essa classe irá fazer com que a ontologia fique inconsistente;
- Classificação – Verifica a relação entre uma subclasse e todas as classes de forma a criar a hierarquia completa dessa classe;
- Realização – Procura a classe mais específica a que um determinado indivíduo pertence. Esta realização só pode ser efectuada depois da classificação visto que os tipos directos são definidos de acordo com a hierarquia de uma classe;

Capítulo 4. Arquitectura

Neste capítulo irá ser descrita toda a arquitectura desenvolvida de modo a ser possível implementar o agente e a respectiva ontologia. Numa primeira parte é descrita a arquitectura em si e depois o modo de interacção entre os diversos componentes nela existentes.

4.1 Descrição da arquitectura

A arquitectura do sistema mostra o modo como o sistema está implementado de forma a ser possível atingir os objectivos pretendidos definidos no primeiro capítulo. Não mostra a única arquitectura possível de implementar este sistema, mas tenta mostrar aquela que poderá ser a melhor aproximação conjugando simplicidade e eficiência. Estas duas características dever-se-ão ter sempre em conta num sistema de manufactura onde financeiramente existem sempre limites, mas ao mesmo tempo se pretende obter o melhor sistema ágil de produção em massa, com suporte a reconfigurabilidade elaborando produtos personalizados, tal como foi referido no primeiro capítulo.

Como tal, a próxima figura (Figura 4.1.1), mostra a arquitectura implementada neste sistema.

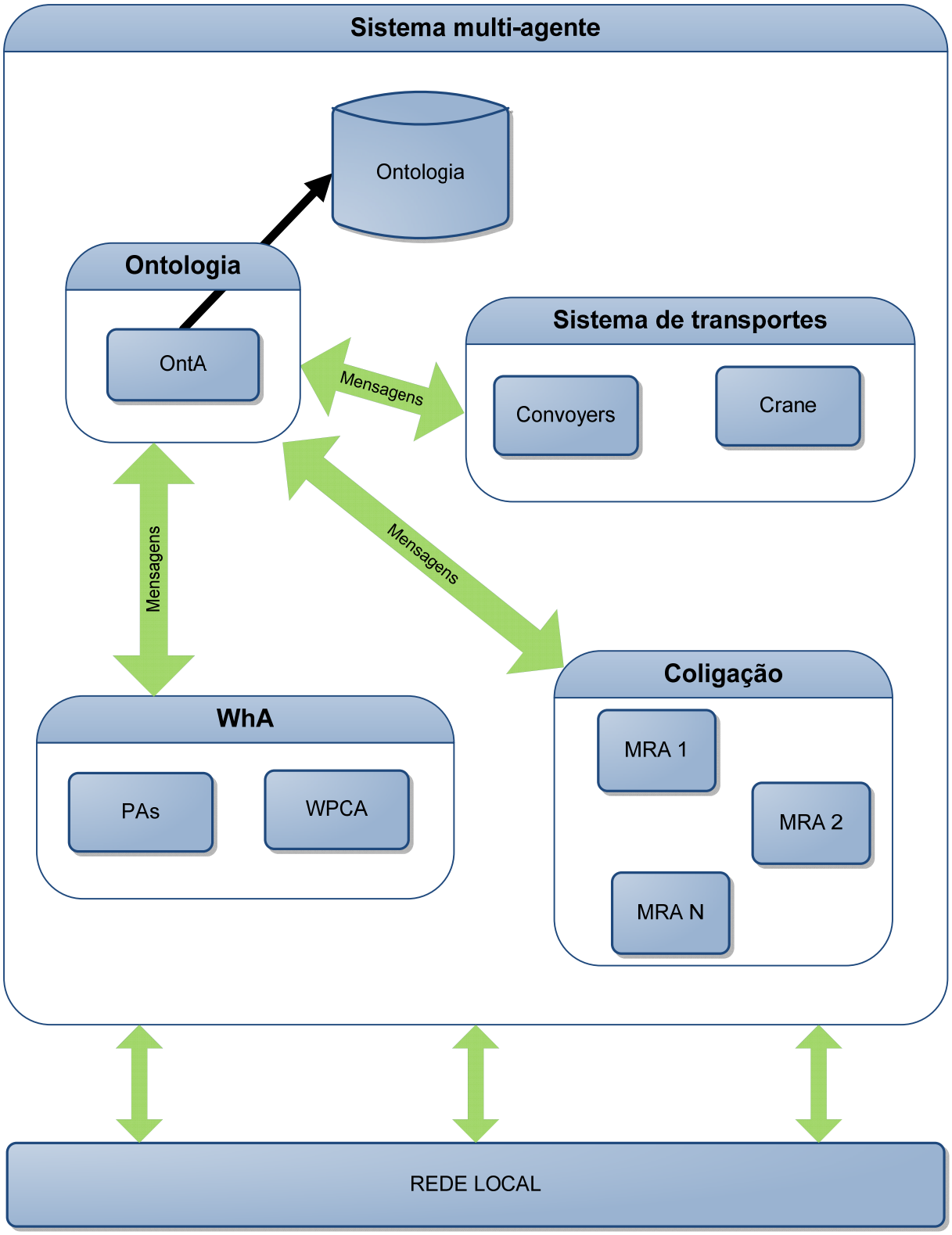

Figura 4.1.1 – Arquitectura implementada

A arquitectura aqui apresentada (Figura 4.1.1) assenta sob uma rede local de computadores. Dentro desta rede irão coexistir diversos agentes. Todos estes agentes interagem com o objectivo de controlar um sistema de manufactura de forma a seguir um conjunto de ordens que irão fazer sair do sistema o produto final pretendido.

Os agentes aqui presentes, também descritos em (Ferreira 2009) à excepção do agente da Ontologia (OntA) e da própria ontologia, são apresentados de seguida:

- WhA – Armazém do sistema (*WareHouse Agent*);
- PA – Agente Produto (*Product Agent);*
- WPCA – Palete que transporta o produto pela planta fabril (*Work Piece Carrier Agent*);
- MRA – Agente responsável pela ligação entre o sistema multi-agente e o sistema físico (*Manufacturing Resource Agent*), recorrendo a um outro agente denominado por AMI (*Agent Machine Interface*);
- *Crane* – Grua presente no sistema;
- *Convoyer* – Tapetes existentes no sistema.

O agente da ontologia tem como função fazer a ponte de ligação entre todos os agentes e a ontologia. A ontologia apenas se apresenta acoplada a este agente. Desta forma, todos os outros agentes sabem que existe um outro responsável pela ontologia, ao qual lhe irão enviar mensagens sempre que necessitarem de informação do sistema que se encontra na ontologia. A comunicação entre todos os agentes é feita, tal como referida anteriormente, através de troca de mensagem de acordo com o protocolo *Fipa-Request* (FIPA 2002).

De forma a controlar o sistema físico recorre-se a um conjunto de bibliotecas de baixo nível (tipicamente funções programadas recorrendo à linguagem C/C++ (Kernighan and Ritchie 1988; Stroustrup 2004)).

Cada componente físico do sistema irá ter acoplado um agente pertencente ao sistema. Este agente é responsável pelo controlo do respectivo componente. Cada agente irá conter um ficheiro XML (Bray, Paoli et al. 2008) com a descrição do respectivo componente. O único agente que difere dos outros é o agente responsável pela ontologia.

O objectivo deste agente é permitir a integração da ontologia no resto do sistema, bem como efectuar todas as operações necessárias sobre a ontologia de modo a que esta sofra as mesmas evoluções que o resto do sistema. Ele vai disponibilizar um conjunto de funções que irá permitir aos outros agentes efectuarem pedidos, aos quais serão respondidos consoante o que estiver registado naquele momento, na ontologia. Será naquele momento, visto que um sistema que sofre evoluções poderá ter respostas diferentes para a mesma pergunta em espaços temporais diferentes.

Em todo o sistema, este agente é único fazendo com que a ontologia seja centralizada. Sendo centralizada faz com que o conhecimento do sistema não esteja disperso, evitando incoerências. Se a ontologia estivesse dívida por vários componentes, poderia existir informação

num determinado componente que contradissesse o que estava registado noutro componente. Sendo centralizada, este problema não se põe. Outro problema que se evita utilizando uma abordagem centralizada está relacionado com o facto de se saber sempre onde está a informação. Ou seja o conhecimento do sistema, que está guardado na ontologia, está sempre nesta ontologia. Caso existissem diversos componente com partes diferentes do conhecimento, os outros componentes nunca iriam saber onde pesquisar o que pretendiam. Também se pode pensar em ter vários componentes, mas com a totalidade da ontologia em vez de ter partes dela. Esta abordagem também iria trazer problemas. Em caso de diferenças, por falta ou falha na actualização, qual a versão correcta? Esta abordagem iria trazer mais incoerência ao sistema.

Sendo uma abordagem centralizada, pode-se pensar na possibilidade de perda da informação quando a máquina onde a ontologia estivesse a correr fosse abaixo. Sim, de facto este problema pode acontecer, mas o agente da ontologia guarda no disco da máquina um ficheiro referente à ontologia de tempos a tempos (está programado para ser de 10 em 10 segundos, mas este valor pode ser sempre alterado) com a informação actualizada. Portanto em caso de avaria da máquina é possível reiniciar o sistema sem que o conhecimento seja perdido. Embora não seja possível garantir a 100% que não exista perca de conhecimento, esta probabilidade é bastante reduzida utilizando a gravação automática do conteúdo da ontologia.

4.1.1 Comunicação entre os agentes

Os agentes necessitam de uma rede de computadores de forma a comunicarem entre eles. Como cada agente estará acoplado a um determinado componente físico do sistema, será coerente pensar que cada agente terá um computador dedicado. Como tal será necessária uma rede de computadores de modo a ser possível a comunicação entre os diversos agentes.

Para suportar a comunicação entre os diversos agentes, foi utilizado o sistema distribuído JADE (Bellifemine, Poggi et al. 1999) referido no capítulo três. A comunicação entre cada agente e o agente da ontologia é feita através de um protocolo de comunicação presente no JADE, designado por FIPA-REQUEST, tal como referido no capítulo anterior. A figura (Figura 4.1.2) mostra a interacção existente entre o agente responsável pela ontologia e outro agente do sistema.

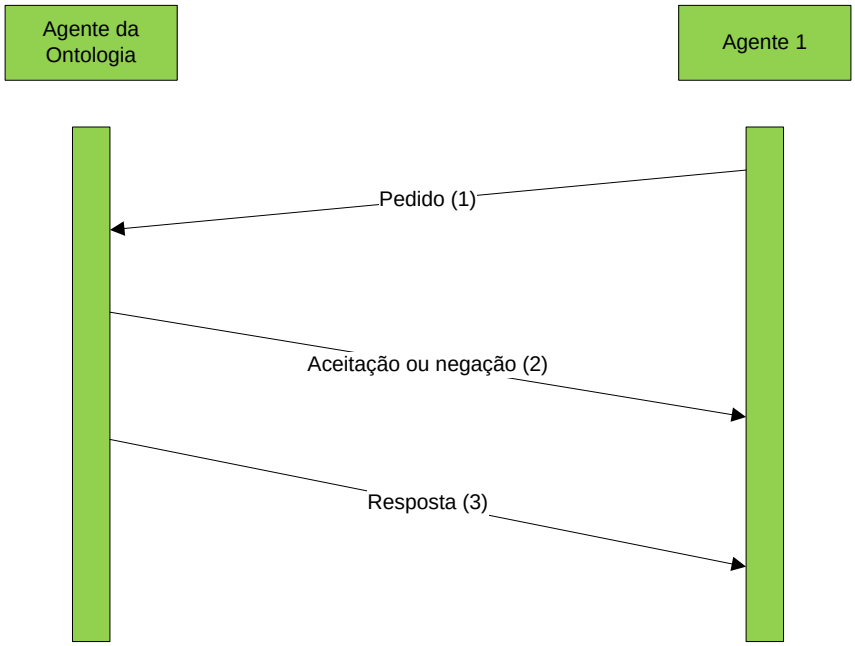

Figura 4.1.2 – Troca de mensagens entre o agente da ontologia e outro agente do sistema

O início da conversa entre estes dois agentes é feita através do envio de uma mensagem contendo o pedido pretendido (mensagem 1 da figura) ao agente da ontologia. Para cada pedido são enviadas duas respostas. A primeira consiste numa mensagem de aceitação ou negação, para dizer ao outro agente que o pedido foi aceite com sucesso ou não. Caso o pedido seja reconhecido é enviada uma mensagem de aceitação, caso contrário é enviada uma mensagem de negação. Após o envio desta mensagem e caso tenha sido de aceitação, o agente da ontologia irá processar o pedido, obtendo uma determinada resposta, por parte da ontologia. Essa resposta é então enviada de volta finalizando assim a troca de mensagens para cada pedido.

4.2 Descrição da Ontologia

A ontologia é a base de conhecimento de todo o sistema. Aqui é onde se encontra toda a informação relativa ao sistema, tanto sistema de agentes como sistema físico de manufactura.

Por conhecimento entende-se como a informação e os factos conhecidos sobre um determinado dominio. O conhecimento é uma faculdade humana que resulta da interpretação da informação. O conhecimento deriva da combinação da informação e da interpretação feita por cada pessoa. Transpondo esta definição para o dominio da manufactura inteligente, o conhecimento resulta da interpretação feita da informação existente na ontologia. A ontologia armazena factos e a interpretação desses factos irá dar origem ao conhecimento explícito da ontologia.

Ao contrário de outras ontologias convencionais, esta ontologia tem a capacidade de se adaptar às alterações efectuadas no sistema, de modo a acompanhar os requisitos do produto a implementar. Quer isto dizer, que a as classes existentes na ontologia (tal como as suas propriedades e as suas instâncias), vão sofrendo alterações ao longo do tempo, acompanhando a evolução de todo o sistema. Estas alterações são efectuadas pelo agente da ontologia, de acordo com as mensagens que vão sendo enviadas pelos outros agentes do sistema.

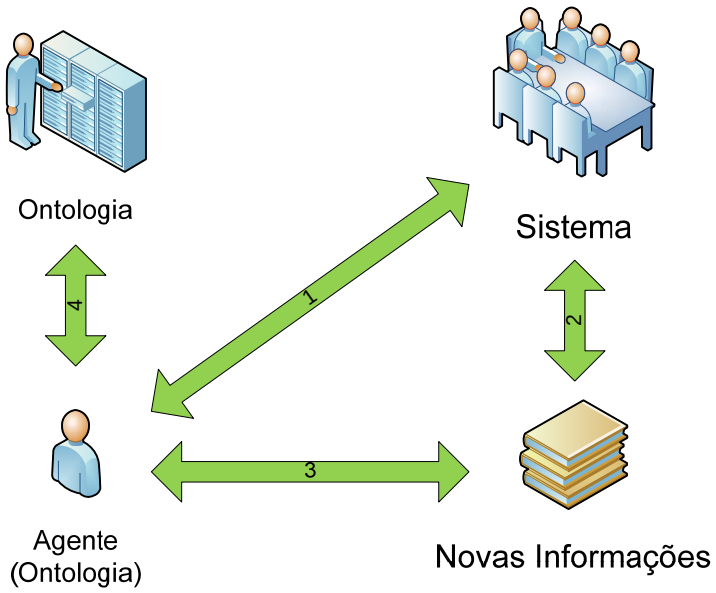

Figura 4.2.1 – Actualização das classes da Ontologia

A figura (Figura 4.2.1) mostra de uma forma genérica como é actualizada a informação na ontologia. Sempre que existem novas informações geradas pelo sistema multi-agente, que pode

acontecer, por exemplo, com a adição de um novo componente, essas informações são enviadas, através de um ficheiro XML para o agente da ontologia.

Na figura, a interacção 1, indica que o sistema acede à ontologia (através do agente da ontologia) para ter conhecimento do sistema actual. Se não existir a informação pretendida no sistema, essa informação pode ser gerada pelo sistema. Por exemplo, se um determinado agente precisar de uma determinada tarefa e nenhum agente conseguir oferecer essa tarefa (não existe nenhum agente registado na ontologia com capacidade de oferecer essa tarefa), irá ser criada uma coligação (coligações detalhadas em (Ferreira 2009)). A criação da coligação gera novas informações (por exemplo novas tarefas; novas propriedades; novas limitações) (interacção 2 da figura). Essas informações serão então enviadas para a ontologia (interacção 3 da figura) através de um ficheiro XML, que através da interpretação de mesmo ficheiro (feita pelo agente da ontologia), as classes existentes são actualizadas e se necessário serão criadas novas classes, propriedades e/ou instâncias.

4.2.1 Geração de novo conhecimento

A figura (Figura 4.2.2) mostra de que forma o sistema consegue gerar novo conhecimento baseado em informações enviadas por todos os agentes existentes no sistema.

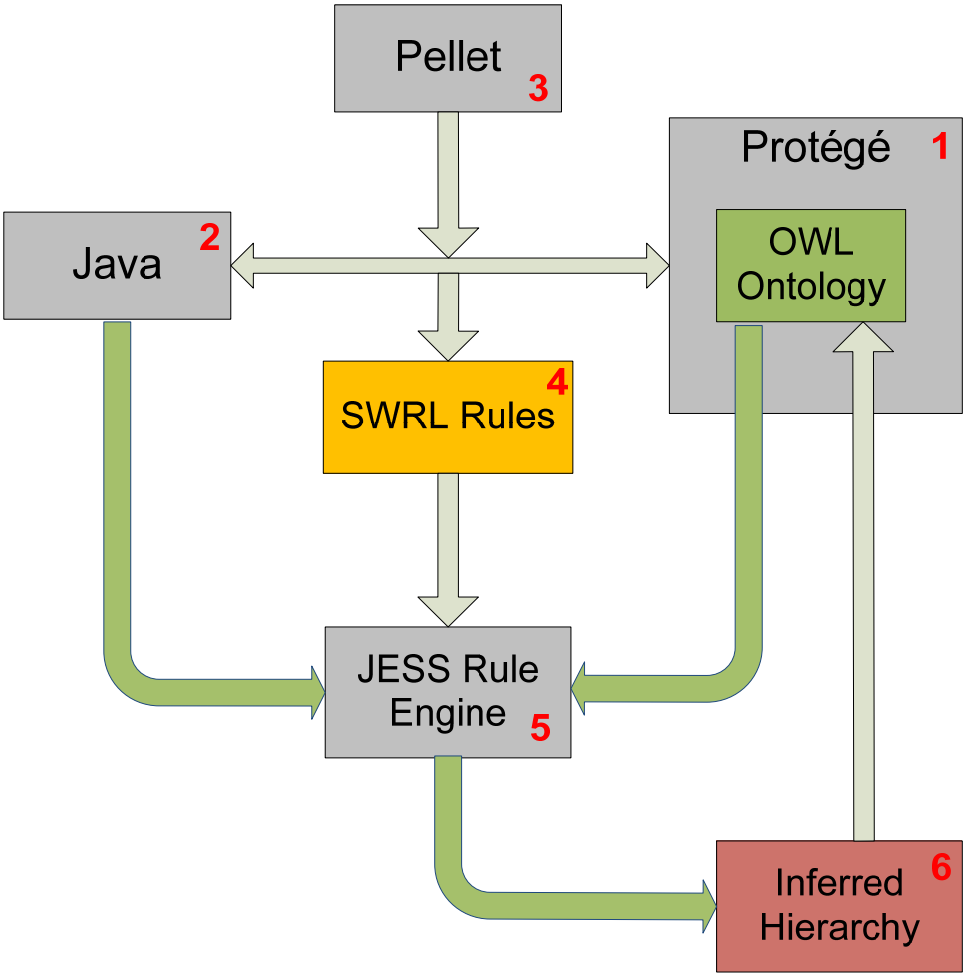

Figura 4.2.2 – Esquema de geração do conhecimento

Através do Protégé é criada a ontologia na sua forma inicial (passo 1 da figura), contendo as classes que irão descrever o sistema. Esta ontologia é criada recorrendo à linguagem OWL-DL, como descrito no capítulo três. De uma forma geral, todas as outras operações são efectuadas através da linguagem de programação Java. Ou seja, é através do agente da ontologia que se efectuam as restantes operações. Este agente começa por carregar a ontologia para a sua memória, recorrendo a bibliotecas do Protégé e da linguagem OWL.

Sempre que existe a necessidade de criar novas classes, estas são criadas através de Java (passo 2 da figura) e guardadas indirectamente na ontologia. Indirectamente dado ao facto que sempre que é efectuada uma operação de escrita na ontologia, o *Pellet* vai verificar a consistência da ontologia (passo 3 da figura). Quer isto dizer que é verificada não só a sintaxe da

ontologia como também a sua semântica. Deste modo é então possível garantir que a ontologia se mantém consistente ao longo de todas as adaptações efectuadas no sistema. O *Pellet* contém uma API que é acoplada às classes já existentes e, desta forma, é só utilizar as funções nela existentes para dar início ao motor do *Pellet* e todas as outras operações são feitas de forma automática.

Através de SWRL é possível implementar regras sobre as classes, propriedades e instâncias existentes na ontologia (passo 4 da figura). Estas regras também são implementadas recorrendo à linguagem Java e a bibliotecas de programação do Protégé e de SWRL (também através do agente da ontologia). As regras são criadas em forma de texto e guardadas em objectos disponibilizados pela API do SWRL. Estas regras em conjunto com a ontologia OWL serão enviadas para o sistema de processamento de regras, o JESS, (passo 5 da figura) que irá processar todas as informações da ontologia e em conjunto com essas regras irá gerar novo conhecimento. Este novo conhecimento é gerado sob a forma de novas classes, propriedades e/ou instâncias. A comunicação entre o SWRL e o JESS é feita, também em java, através de uma "ponte" existente entre estas duas entidades. Esta ponte designa-se por "*SWRLJessBridge*". Para utilizar esta comunicação entre o JESS e o SWRL basta inicializar um objecto pertencente à classe SWRLJessBridge e utilizar os métodos que permitem enviar as regras criadas para o motor do JESS. Após ter sido gerado este conhecimento, a ontologia é então actualizada (passo 6 da figura). Esta actualização é feita sobrepondo o novo conhecimento ao já existente, sendo o objecto java sempre o mesmo. Com o sistema actualizado, as novas informações ficam de imediato disponíveis para o resto do sistema podendo ser consultadas a qualquer momento, através do agente da ontologia.

Capítulo 5. Implementação e Resultados

Neste capítulo irá ser descrita toda a implementação efectuada de modo a dar vida à arquitectura apresentada no capítulo quarto. Aqui irá ser descrita toda a ontologia, bem como o agente por ela responsável. Também irá ser detalhada a interface gráfica existente no agente da ontologia que possibilita ao utilizador intervir no agente e na ontologia. A interacção entre os diversos agentes do sistema também irá ser detalhada tecnicamente. Por fim irá ser apresentado um cenário de teste onde foram testadas as funcionalidades implementadas bem como os resultados obtidos.

5.1 A Ontologia

Neste subcapítulo a ontologia desenvolvida nesta tese irá ser descrita detalhadamente. A ontologia foi desenvolvida através do programa de computador Protégé versão 3.3.1 utilizando a linguagem OWL-DL, tal como referido no capítulo 3.

O objectivo da criação desta ontologia é obter uma representação abstracta do mundo existente num sistema de manufactura, numa planta fabril. Cada classe irá representar diferentes partes dessa planta fabril de uma forma abstracta. Por exemplo, a classe "*Limitation*" pretende ser uma representação das limitações físicas dos componentes do sistema onde esta ontologia pode ser inserida. A classe "*Skill*" pretende representar todas as tarefas que podem ser executadas na planta fabril. Cada classe irá ter propriedades para melhor caracterizarem cada classe.

Ao longo deste capítulo irão ser descritas todas as classes criadas, bem como todas as propriedades que foram consideradas importantes para a ontologia fazer sentido no contexto do mundo da manufactura.

A figura (Figura 5.1.1) mostra as classes principais existentes na ontologia, sem entrar em detalhe nas subclasses que cada classe tem. As subclasses irão ser apresentadas mais à frente neste capítulo, durante a descrição de cada classe.

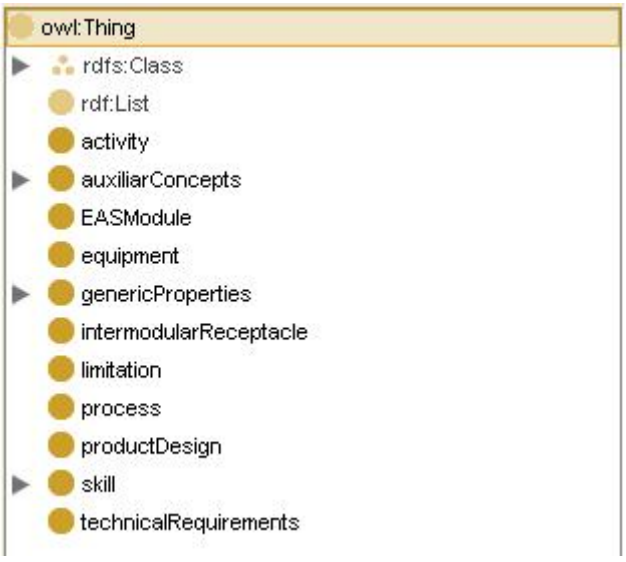

Figura 5.1.1 – Classes presentes na Ontologia

Podemos observar que a estrutura hierárquica da Ontologia começa com a classe "*owl:Thing*". Isto porque, por definição da linguagem OWL, todas as classes são subclasses da classe "*owl:Thing*". As duas primeiras classes que aqui se apresentam ("*rdfs:Class*" e "*rdf:list*") são classes inerentes a qualquer ontologia OWL. Tal como explicado no capítulo três, a linguagem OWL teve como origem uma evolução na linguagem RDF, e como tal, as classes OWL necessitam de definições da linguagem RDF, logo estas duas classes estão presentes na ontologia, embora não desenvolvam um papel directo no sistema.

As outras classes apresentadas na figura são as classes que realmente desempenham um papel directo no sistema e que foram implementadas propositadamente. A ordem pela qual as classes aparecem não tem qualquer significado, não alterando o significado ontológico no sistema.

De seguida descrevem-se as todas as classes criadas na ontologia bem como as suas propriedades e qual o seu significado.

- Classe ***Activity***:

Esta classe representa uma actividade que um determinado componente pode efectuar. Uma actividade descreve um determinado processo. Ou seja, uma única actividade pode estar presente em processos diferentes. Uma actividade está também associada a diversas habilidades (ou *skill*). A figura (Figura 5.1.2) mostra as propriedades desta classe.

Figura 5.1.2 – Propriedades da classe *Activity*

- Classe ***AuxiliaryConcepts***:

A classe ***AuxiliaryConcepts*** apresenta duas subclasses e esta classe existe com a finalidade de representar determinados conceitos, matemáticos e não matemáticos, utilizados em qualquer sistema de manufactura. A figura (Figura 5.1.3) mostra a classe ***AuxiliaryConcepts*** com todas as suas subclasses.

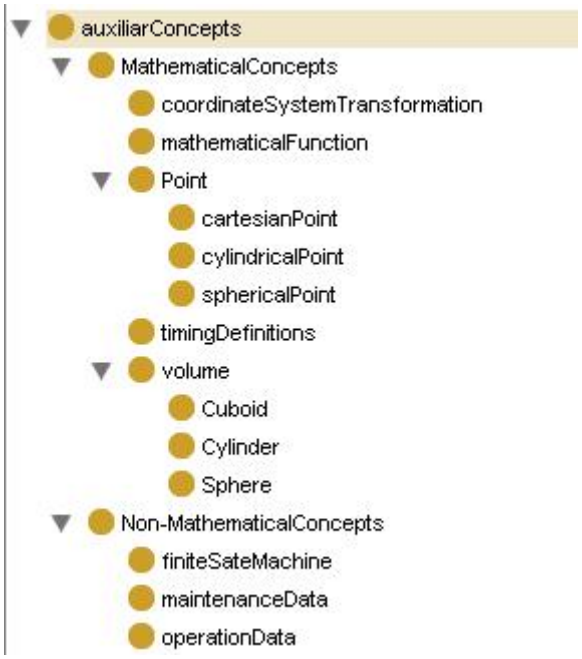

Figura 5.1.3 – Classe ***AuxiliaryConcepts*** com todas as suas subclasses

Esta classe encontra-se dividida, em duas subclasses. A primeira, designada por ***MathematicalConcepts*** tem como objectivo descrever todos os conceitos matemáticos necessários num EPS. Como subclasses da classe ***MathematicalConcepts*** foram criadas quatro subclasses:

- ***coordinateSystemTransformation***: Representa a notação de Euler para o sistema de transformação de referenciais. Esta subclasse apresenta várias propriedades que representam as posições lineares e angulares. A figura (Figura 5.1.4), ilustra essas propriedades:

Figura 5.1.4 – Propriedades da classe ***coordinateSystemTransformation***

- ***mathematicalFunction***: Esta subclasse tem como único objectivo descrever qualquer função matemática que seja utilizada no sistema. Aqui fica registado tanto a função, como as suas variáveis (Figura 5.1.5):

Figura 5.1.5 – Propriedades da classe ***mathematicalFunctions***

- ***Point***: A classe ***Point*** representa as diversas possibilidades de se descrever um ponto matematicamente. As suas subclasses mostram essas possibilidades: Ponto cartesiano; ponto cilíndrico e ponto esférico. Cada uma destas subclasses contém as propriedades necessárias a que seja descrito o respectivo ponto:

Figura 5.1.6 – Propriedades das subclasses ***Point***: a) ***cartesianPoint***; b) ***cilindricalPoint*** e c) ***sphericalPoint***

- ***timingDefinitions***: Esta classe descreve as definições temporais do sistema. Esta classe não apresenta quaisquer propriedades, sendo que o objectivo é o utilizador definir em concreto esta classe.

- ***volume***: Esta classe define matematicamente um volume de qualquer componente do sistema. Matematicamente é possível definir um volume de três formas diferentes: volume cúbico; volume cilíndrico e volume esférico. Cada uma destas subclasses apresenta propriedades que descrevem o respectivo volume (Figura 5.1.7):

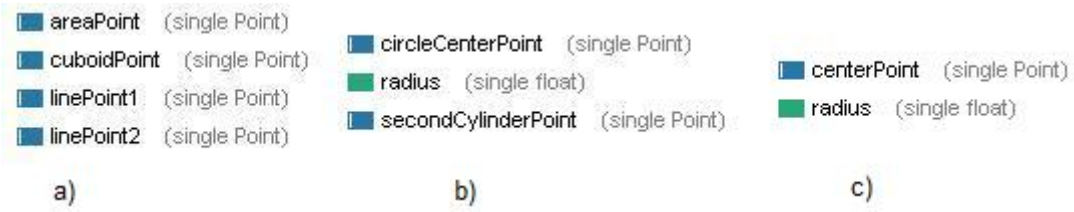

Figura 5.1.7 – Propriedades das subclasses ***volume***: a) ***cuboid***; b) ***cylinder*** e c) ***spherical***

Passando agora à próxima subclasse da classe ***genericProperties***, temos a classe onde são descritos conceitos genéricos não matemáticos, a classe ***non-mathematicalConcepts*** (Figura 5.1.3):

- ***finiteStateMachine***: Classe para representar uma máquina de estados (Wagner 2006). Aqui fica registado qual o estado actual da máquina através da sua propriedade (Figura 5.1.8):

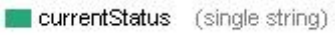

Figura 5.1.8 – Propriedade da classe ***finiteStateMachine***

- ***maintenanceData***: Esta classe regista informação importante relativo a manutenção efectuada a qualquer componente do sistema. A informação é representada pelas propriedades desta classe (Figura 5.1.9):

Figura 5.1.9 – Propriedades da classe ***maintenanceData***

- ***operationData***: Esta classe regista informação relevante à operação a decorrer no sistema. A informação aqui registada é a descrição da *skill* e o estado em que se encontra actualmente a máquina de estados (Figura 5.1.10):

Figura 5.1.10 – Propriedades da classe ***operationData***

A classe ***auxiliaryConcepts*** fica assim definida ontologicamente. A próxima classe descrita na ontologia é a classe ***EASModule***.

- ***EASModule***: Esta classe define um módulo EAS que representa a unidade mínima que se pode obter num sistema EPS e que oferece como serviço pelo menos um *skill*. Esta classe está definida pelas propriedades apresentadas na figura (Figura 5.1.11):

Figura 5.1.11 – Propriedades da classe ***EASModule***

A propriedade ***consistsOf*** designa que o módulo consiste num equipamento do sistema ou numa conexão inter-módulo. Mais abaixo esta classe (conexão inter-módulo) é descrita. A outra propriedade visível na figura (***name***) designa o nome do módulo.

- ***Equipment***: A classe ***Equipment*** é responsável por descrever qualquer equipamento físico existente no sistema de manufactura. Um componente pode pertencer a um ***EASModule*** e executa um determinado *skill*. Como qualquer componente físico, um equipamento tem sempre limitações. Toda esta descrição é visível pelas propriedades desta classe (Figura 5.1.12):

Figura 5.1.12 – Propriedades da classe ***Equipment***

- ***IntermodularReceptacle:*** Esta classe representa um componente físico de interligação entre módulos, ou seja, sempre que dois módulos (EASModules) pretendem interligar-se, existe um componente físico responsável por essa interligação, designado receptáculo intermodular que é descrito na ontologia por esta classe. As suas propriedades podem ser vistas na figura (Figura 5.1.13):

Figura 5.1.13 – Propriedades da classe ***intermodularReceptacle***

Estas propriedades dizem que um módulo de interligação é definido por ***skills*** ou por ***genericProperties*** e que este módulo é parte de um ***EASModule***, ou seja, o que define um módulo de interligação tanto pode ser um *skill* como também por propriedades genéricas (definidas mais à frente), e que este módulo de interligação é parte de um módulo EAS do sistema. Estas propriedades foram definidas dado ao facto que esta interligação tem de ser efectuada entre dois módulos EAS, e que essa interligação vai definir propriedades.

- ***GenericProperties:*** Esta classe tem como finalidade descrever todo o tipo de propriedades que poderão classificar um sistema de manufactura com os requisitos de um EPS/EAS. Para melhor classificar as diversas propriedades, foram criadas subclasses. A figura mostra essas subclasses.

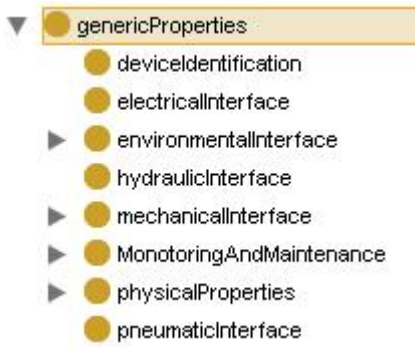

Figura 5.1.14 – Subclasses da classe ***GenericProperties***

Esta classe tem propriedades que geram limitações do respectivo componente associado e que definem um ou mais módulos de interligação. Estas limitações são inerentes ao facto de estarem definidas propriedades. Por exemplo, ao definirmos um componente como tendo uma determinada tensão de ligação, não podemos ligar outra tensão, logo temos uma limitação do componente. Estas propriedades vão definir um módulo de interligação de modo a ser possível conhecer esse módulo, senão não seria possível conhecermos a interligação. A figura (Figura 5.1.15) mostra essas propriedades definidas na ontologia.

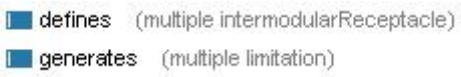

Figura 5.1.15 – Propriedades da classe ***GenericProperties***

As propriedades genéricas foram classificadas em diversas classes. A primeira classe refere-se à identificação do componente:

- ***DeviceIdentification***: Esta classe define as propriedades que irão identificar o componente físico e permitir identificá-lo de forma unívoca:

Figura 5.1.16 – Propriedades da classe ***DeviceIdentification***

Cada componente apresenta uma descrição sobre o seu funcionamento, um ID como sendo um identificador único de cada componente, um nome, um tipo, um *part number* identifica o componente como sendo de um determinado fabricante, um número de série que lhe é atribuído aquando do sua construção que também o identifica de forma unívoca.

As outras subclasses definidas na ontologia são:

- ***ElectricalInterface***: Esta classe define electricamente o componente, ou seja, caracteriza-o consoante ao tipo de electricidade de funcionamento.

Figura 5.1.17 – Propriedades da classe ***ElectricalInterface***

As propriedades definidas na figura (Figura 5.1.17) definem o que pode ser caracterizado a nível eléctrico num componente:

i. ***maxElectricalCurrent*** – Define a corrente eléctrica máxima que pode ser utilizada com o respectivo componente;
ii. ***maxElectricalVoltage*** – Define a tensão máxima que pode ser aplicada ao componente;
iii. ***minEletricalCurrent*** – Define a corrente mínima que pode ser aplicada ao componente para garantir o seu correcto funcionamento;

iv. ***minElectricalVoltage*** – Define a tensão mínima que pode ser aplicada ao componente para garantir o seu correcto funcionamento;

v. ***suggestedElectricalVoltage*** – Define a tensão recomendada de funcionamento para o respectivo componente.

- ***EnvironmentInterface***: Nesta classe são definidas as condições ambientais de funcionamento de cada componente. Nesta ontologia foram definidas duas categorias ambientais, tal como definido na figura (Figura 5.1.18):

Figura 5.1.18 – Subclasses da classe ***EnvironmentInterface***

Cada uma destas subclasses tem parâmetros para caracterizar os limites de funcionamento do respectivo componente:

- Humidade máxima e mínima;
- Temperatura máxima e mínima.

Estas propriedades foram criadas na ontologia, nas respectivas classes (Figura 5.1.19):

Figura 5.1.19 – Propriedades que definem as condições ambientais: a) condições de humidade; b) condições de temperatura

- ***HydraulicInterface*** – Esta classe tem a finalidade de classificar o componente quanto às suas ligações hidráulicas. São definidas as pressões máxima e mínima que o componente suporta e a pressão recomendada para o seu funcionamento (Figura 5.1.20):

Figura 5.1.20 – Propriedades que caracterizam a interface hidráulica do componente

- ***MechanicalInterface***: Esta classe tem como função definir a interface mecânica de um componente, tendo como subclasse uma classe que caracteriza o seu espaço de trabalho:

Figura 5.1.21 – Subclasse da classe ***MechanicalInterface***

A subclasse apresentada na figura (Figura 5.1.21) caracteriza o espaço de trabalho de um componente do sistema através das propriedades:

Figura 5.1.22 – Propriedades da classe ***Envelope***

Estas propriedades definem o volume de trabalho (***envelopeDescription***) e o sistema de transformação de coordenadas (***coordinates***) de modo a permitir trabalhar em diversas coordenadas e não apenas nas definidas pelo componente. Esta última propriedade é herdada da classe de cima, ou seja, da classe ***MechanicalInterface***. Deste modo é possível adicionar mais parâmetros mecânicos e a todos eles definir um sistema de transformação de coordenadas.

- ***MonitoringAndMaintenance***: Esta classe define diversos parâmetros referentes à monitorização e à manutenção do componente. As subclasses desta classe definem os parâmetros para esta classe (Figura 5.1.23):

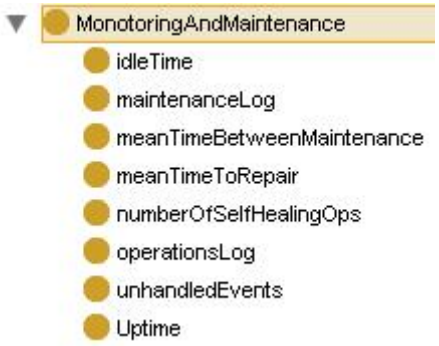

Figura 5.1.23 – Parâmetros de monitorização e de manutenção

Todas estas subclasses têm como propriedade um valor que representa o respectivo parâmetro:

i. ***idleTime***: Tempo em que o componente se encontra parado;
ii. ***maintenanceLog***: Registo de manutenção do componente;
iii. ***meanTimeBetweenMaintenance***: Tempo entre manutenções;
iv. ***meanTimeToRepair***: Tempo de reparação do componente;
v. ***numberOfSelfHealingOps***: Número de operações realizadas que permitiram ao componente auto reparar-se caso estejam definidas essas operações;
vi. ***operationsLog***: Registo de operações efectuadas ao componente a nível de monitorização e a nível de manutenção;
vii. ***unhandledEvents***: Número de eventos não definidos pelo sistema;
viii. ***Uptime***: Tempo que o componente demora até estar pronto a ser utilizado, desde que é ligado.

- ***physicalProperties***: Esta classe define as propriedades físicas de um componente pertencente ao sistema. Podem ser definidas várias propriedades, tais como as definidas na figura (Figura 5.1.24):

Figura 5.1.24 – Propriedades físicas definidas na ontologia

O centro de gravidade define um ponto que será o ponto que representa o centro de gravidade do componente. As dimensões do componente são definidas como três componentes, uma para cada dimensão. O momento de inércia do componente é definido como um ponto com coordenadas nos três eixos (x, y, e z) cartesianos. O ponto de localização representa o ponto geográfico onde o componente se situa fisicamente no espaço da linha de montagem. A massa é representada por um valor e por uma unidade.

- ***pneumaticInterface*** – Esta classe define a gama de pressões pneumáticas que um componente suporta para o seu funcionamento. Define também a pressão recomendada, tal como mostra a figura (Figura 5.1.25):

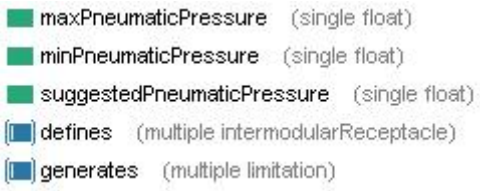

Figura 5.1.25 – Propriedades da interface pneumática

- ***Limitation***: Esta classe define todas as limitações existentes no sistema, ou seja, todos os equipamentos físicos presentes no sistema contêm limitações e essas limitações são descritas por esta classe. As propriedades desta classe são as representadas na figura (Figura 5.1.26).

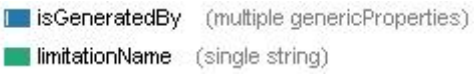

Figura 5.1.26 – Propriedades da classe ***Limitation***

- ***Process***: Esta classe descreve um conjunto de transformações a serem executadas num determinado produto. Um processo é descrito por uma actividade e tem conhecimento dos requisitos técnicos do produto. A figura mostra as propriedades definidas para esta classe (Figura 5.1.27):

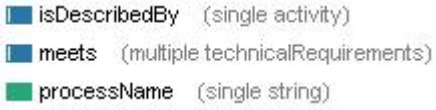

Figura 5.1.27 – Propriedades da classe ***Process***

- ***ProductDesign***: Esta classe representa o desenho do produto com o objectivo de criar requisitos técnicos tendo em conta o produto final que se pretende obter. Como tal esta classe apresenta uma propriedade (Figura 5.1.28):

Figura 5.1.28 – Propriedades da classe ***ProductDesign***

- ***Skill***: Um determinado *skill* representa o que um determinado módulo do sistema consegue executar, e como tal é esse *skill* que é oferecido como serviço aos outros módulos do sistema. Para representar qualquer *skill* foi então criada uma classe na ontologia, designada por ***Skill***. De modo a definir qualquer *skill* foram criadas as propriedades definidas na figura (Figura 5.1.29):

Figura 5.1.29 – Propriedades da classe ***Skill***

A propriedade ***hasSkill*** designa que qualquer *skill* pode ser composto por outras *skill*, passando a designar-se por *skill* complexo. Um determinado *skill* estar associado a diversas actividades pela propriedade ***isAssociatedTo***. Qualquer *skill* é executado por ou mais equipamentos. Esses equipamentos ficam registados na propriedade ***isExecutedBy***. Uma *skill* define um ou vários módulos de interligação, visto que pode ser alterado se o módulo de interligação for alterado. Esta definição fica descrita pela propriedade ***defines***. Por fim, cada *skill* tem um nome próprio, registado na propriedade ***SkillName***.

- ***TechincalRequirements***: Esta classe representa os requisitos técnicos gerados pelo desenho do produto (propriedade ***isCreatedBy***) e são conhecidos pelos processos que irão ter que efectuar tarefas no produto de forma a obter o produto final desejável (propriedade ***isMetBy***). Desta forma ficam definidas as suas propriedades (Figura 5.1.30):

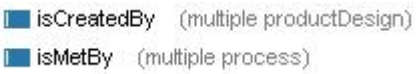

Figura 5.1.30 – Propriedades da classe ***TecnicalRequirements***

5.2 Agente da Ontologia

O agente responsável pela ontologia, como referido anteriormente, tem a função de fazer a integração entre a ontologia e o resto do sistema. Este agente encontra-se integrado no resto do sistema e tem a capacidade de interagir com outros agentes através da troca de mensagens, tal como demonstra a figura (Figura 5.2.1).

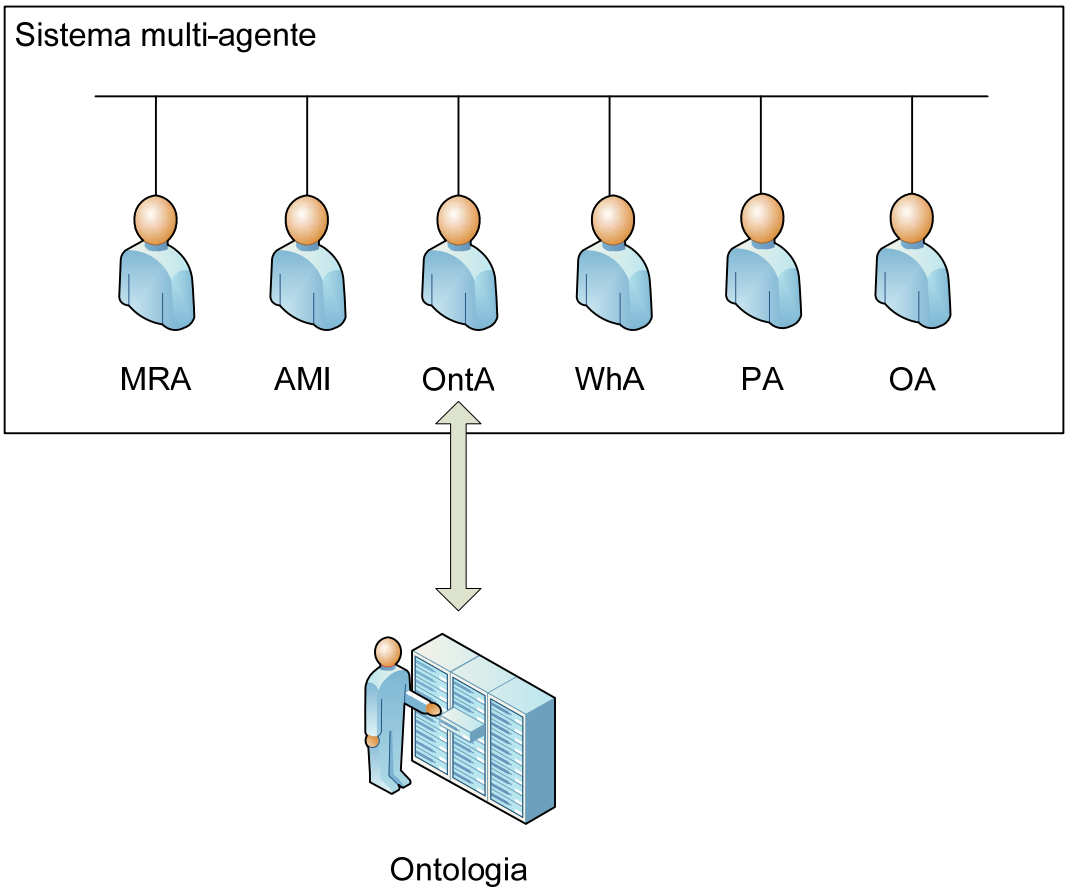

Figura 5.2.1 – Sistema multi-agente

Este agente foi desenvolvido recorrendo à linguagem Java e encontra-se dividido em quatro classes Java:

1. Classe *myAgent*: Nesta classe estão implementados os métodos necessários ao registo do agente no DF e à recepção e envio de mensagens;
2. Classe *Agents*: Esta classe representa todos os outros agentes do sistema. Aqui foram implementados métodos que permitem armazenar informação sobre os outros agentes, tal como, endereços, nome e documento XML para o registo;

3. Classe *Ontology*: Esta classe representa a ontologia OWL aqui desenvolvida. Através desta classe é possível efectuar todas as operações sobre a ontologia, através de métodos desenvolvidos;
4. Classe *GUIAgent*: Esta classe representa a interface gráfica do agente. Todos os métodos aqui desenvolvidos permitem ao utilizador efectuar operações sobre o agente e sobre a ontologia.

Estas classes interagem entre elas sempre que necessário. A figura (Figura 5.2.2) mostra quais as interacções existentes entre as classes.

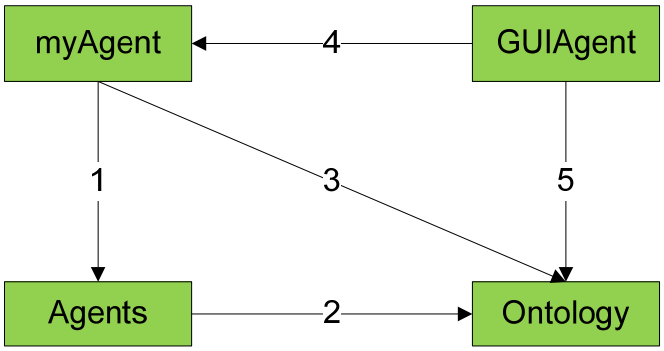

Figura 5.2.2 – Interacção entre as classes Java

Sempre que o agente recebe uma mensagem de registo, a classe *myAgent* irá chamar um método próprio da classe *Agents* para efectuar o registo (interacção 1 da figura). Este método é responsável por ler o ficheiro XML recebido e prepará-lo para ser enviado para a ontologia. Após a leitura, a classe *Agents* chama um método para o devido efeito na classe *Ontology* que irá processar a informação de modo a efectuar com sucesso o registo do novo agente (interacção 2 da figura).

Outras mensagens, além da de registo, podem chegar a este agente. Essas chegam também pela classe *myAgent* e, consoante o pedido que vem na mensagem, é chamado o método apropriado que se encontrada na classe *Ontology* (interacção 3 da figura).

Através da classe *GUIAgent* é possível inicializar o agente da ontologia (interacção 4 da figura) e também efectuar diversas operações na ontologia (interacção 5 da figura).

5.3 Interface gráfica do Agente

A interface gráfica do agente permite ao utilizador interagir e intervir no agente sempre que necessário. A interface gráfica do agente mostra diversas informações relativas ao agente e ao sistema multi-agente.

De seguida encontra-se uma descrição desta interface com exemplos ilustrados.

Na figura (Figura 5.3.1) são visíveis duas tabelas referentes aos agentes presentes no sistema.

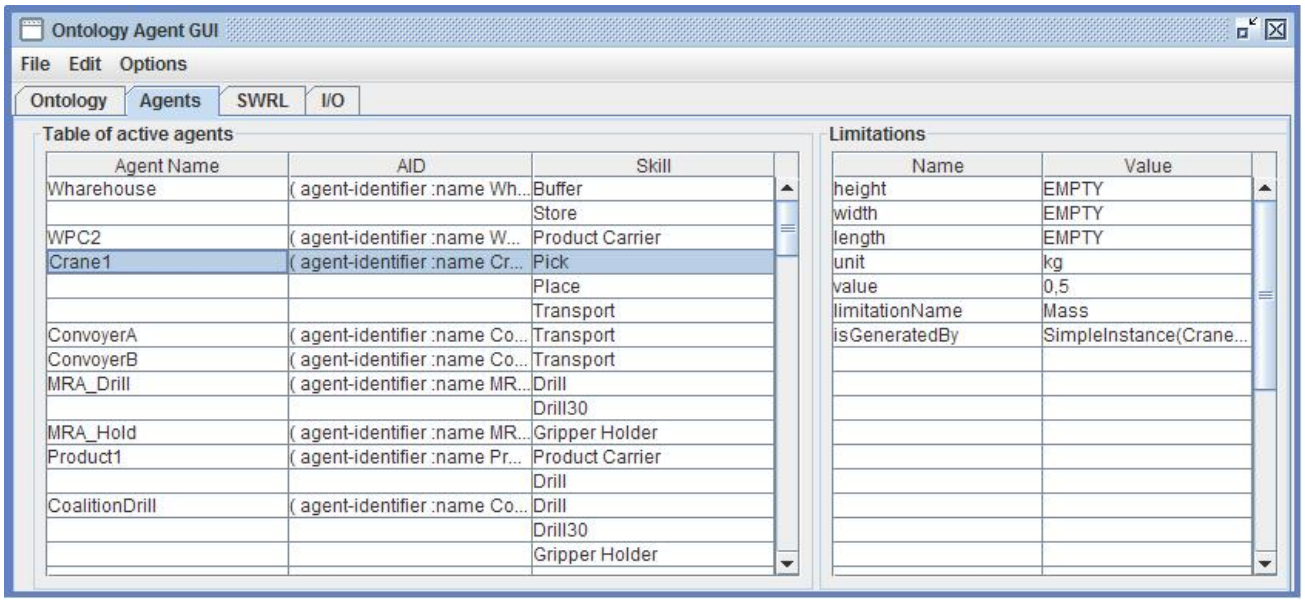

Figura 5.3.1 – Tabelas informativas sobre os agentes presentes no sistema

A primeira tabela (lado esquerdo da figura) mostra todos os agentes presentes no sistema que se encontram registados na ontologia. Nesta tabela são visíveis três colunas: a primeira mostra o nome do agente; a segunda mostra o endereço do agente (endereço atribuído pelo JADE) e a terceira mostra a *skill* que esse agente oferece de acordo com o componente físico a que se encontra acoplado.

Na primeira coluna são visíveis entradas em branco. Isto significa que, sempre que um determinado agente tem a capacidade de oferecer mais de um *skill*, só na linha do primeiro é que é visível o nome do agente; nos restantes não é colocado nada, assumindo-se que aquele *skill* pertence ao mesmo agente.

A segunda tabela (lado direito da figura) mostra todas as limitações que um determinado agente apresenta. Esta tabela é sempre actualizada quando o utilizador selecciona qualquer agente da primeira tabela, com o rato. Esta está dividida em duas colunas. A primeira indica o nome da limitação e a segunda indica o respectivo valor da limitação.

Um exemplo de uma limitação pode ser as dimensões físicas de um determinado componente. Se pensarmos num tapete para transporte, ele não irá suportar peças onde as dimensões dessas peças sejam superiores à largura do próprio tapete. Também poderá existir uma limitação de peso. Se pensarmos numa garra robótica, onde é necessário agarrar peças, a garra desse braço poderá estar limitada a um determinado peso máximo.

A figura (Figura 5.3.2) mostra o registo de entrada e de saída das mensagens recebidas e enviadas, de e para o agente. O objectivo desta interface é o de poder controlar todo o tráfego de mensagens com o agente da ontologia. Deste modo é possível perceber, caso aconteça, que erro pode ter acontecido. Sempre que o agente recebe uma mensagem, independentemente de ele a conhecer ou não, é apresentada neste registo para informar o utilizador. Por cada mensagem a seguinte informação é exibida:

1) ***ConversationID***: representa o ID da mensagem que irá identificar a mensagem permitindo ao agente saber o que o outro agente pretende;
2) ***Message performative***: Representa um número com o tipo de mensagem. A cada número corresponde uma mensagem. Estas mensagens são definidas pelo JADE;
3) ***Sent Message To***: Apresenta o AID do agente para onde foi enviada a mensagem;
4) ***Message received from***: Apresenta o AID do agente que enviou a mensagem;

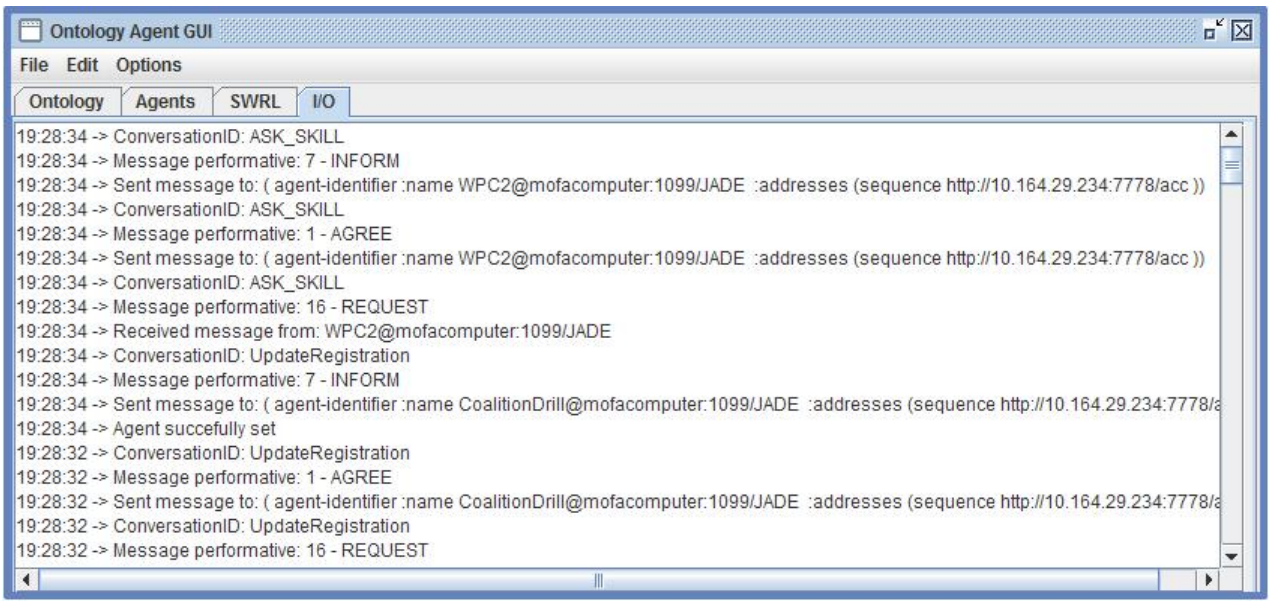

Figura 5.3.2 – Registo de entrada e saída de mensagem do agente

A figura (Figura 5.3.3) mostra a interface gráfica responsável pela ontologia. Ou seja, nesta interface é possível ao utilizador tanto consultar a ontologia como efectuar alterações à mesma. Começando pelo lado esquerdo da figura, é possível visualizar todas as classes da ontologia de uma forma hierárquica (secção assinalada com o número 1). Quando o agente carrega a ontologia, é feita uma pesquisa genérica por todas as classes existentes na ontologia e são então mostradas nesta hierarquia.

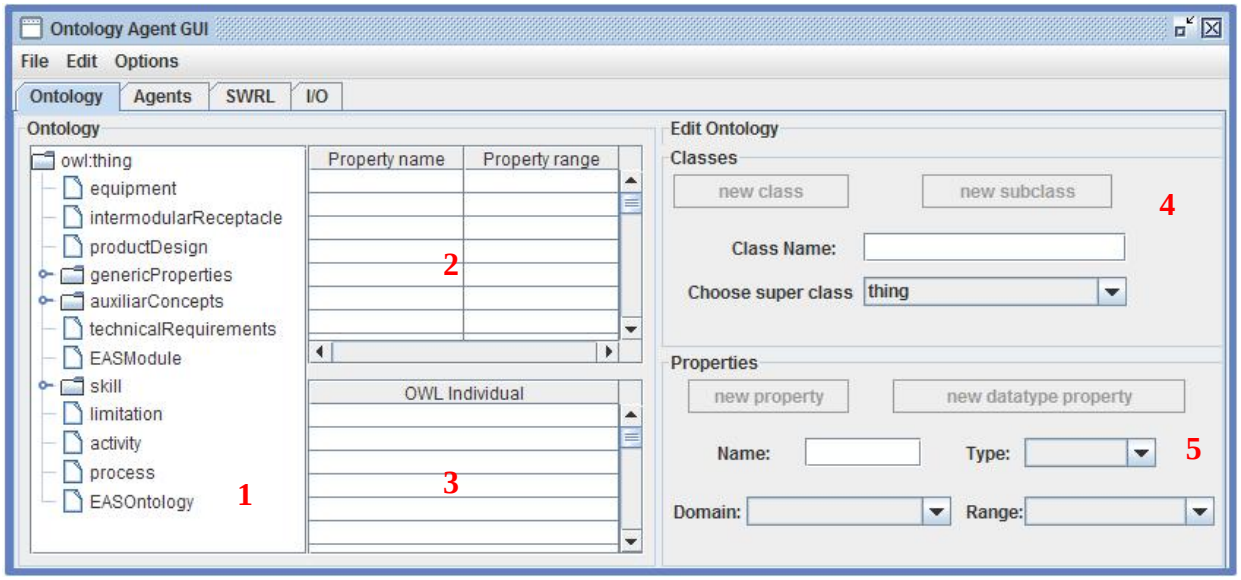

Figura 5.3.3 – Interface responsável pela ontologia

Ao lado desta hierarquia existem duas tabelas. A tabela número 2 apresenta todas as propriedades e o tipo de valor que elas aceitam para uma determinada classe. Essa classe é

escolhida através da selecção feita na hierarquia com o rato. Na tabela número 3 irão ser disponibilizados todos os indivíduos que foram criados para a mesma classe.

Na secção 4 da figura é possível editar a ontologia. É possível tanto criar uma nova classe na ontologia, como editar uma já existente. Quando se selecciona uma classe na hierarquia, na caixa de texto com o nome *Class Name* irá aparecer o nome da classe seleccionada e o nome da classe mãe dessa mesma classe. Desta forma é então possível editar estes valores. Na secção 5 é possível criar propriedades para a classe seleccionada:

- *Name*: Esta caixa de texto mostra o nome da classe seleccionada;
- *Type*: Mostra o tipo de valor caso seja uma propriedade do tipo *OWLDataTypeProperty* (String, Int, Float, Boolean, Any)
- *Domain*: Apresenta o domínio da propriedade, ou seja, em que classe é que a propriedade se encontra;
- *Range*: Caso a propriedade seja do tipo *OWLObjectProperty*, vai apresentar como alcance uma outra classe que não aquela definida no seu domínio.

A figura (Figura 5.3.4) apresenta um exemplo de uma classe seleccionada onde é possível visualizar as suas propriedades e os seus indivíduos. As propriedades estão definidas na ontologia e os indivíduos são criados em tempo real consoante os agentes activos no sistema.

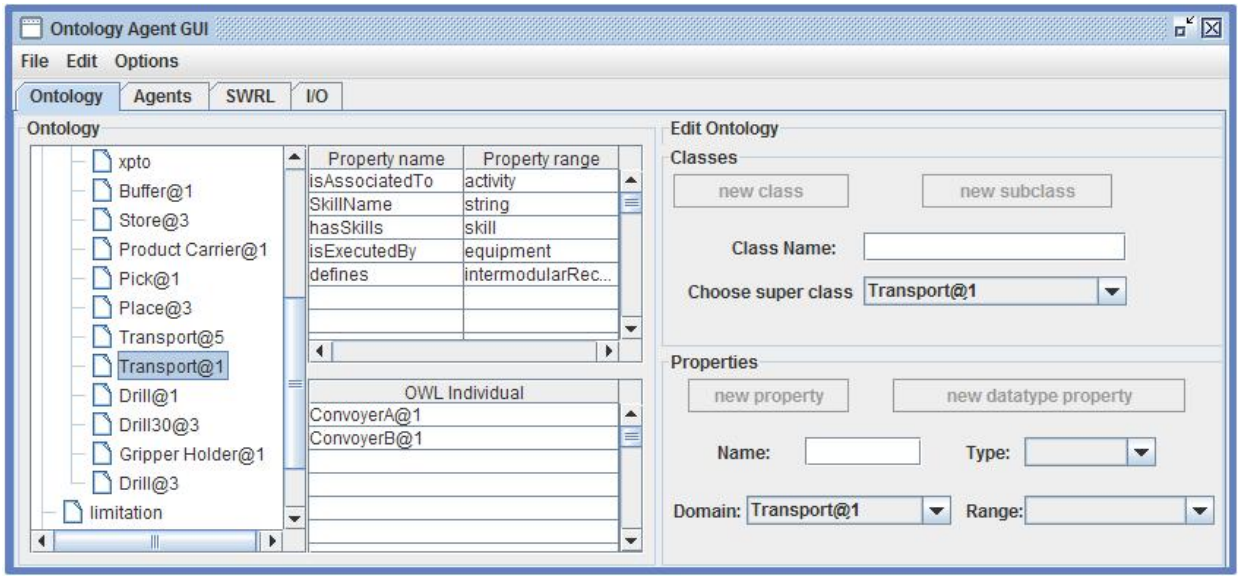

Figura 5.3.4 – Exemplo de uma classe da ontologia

De modo a suportar a implementação de regras SWRL na ontologia, foi criada uma interface que permite ao utilizador efectuar certas operações sobre essas regras. Na figura (Figura 5.3.5) é mostrada essa interface.

Aqui o utilizador pode criar, editar e apagar as regras criadas. Do lado esquerdo da figura são visíveis botões que permitem adicionar classes, propriedades, variáveis SWRL, indivíduos e funções específicas de SWRL (denominadas *built-in*). No meio da figura aparece o nome da regra, a regra em si com todas as especificações criadas e uma tabela onde irão aparecer todas as regras criadas. Na primeira coluna desta tabela é disponibilizada a opção de activar/desactivar a regra; na coluna do meio aparece o nome da regra e na última coluna a expressão que compõe a regra. Do lado direito da figura temos diversas opções como a de adicionar alguns símbolos à regra como por exemplo"(" e ")", "^", "[" e "]" e outros botões que permitem efectuar acções directas na tabela, como por exemplo criar nova regra, que irá adicionar uma nova entrada na tabela com a regra descrita; apagar uma regra, que apaga a regra seleccionada na tabela; editar a regra seleccionada que faz com que o nome da regra e a regra em si apareçam nas caixas de texto devidas para o utilizador alterar o necessário.

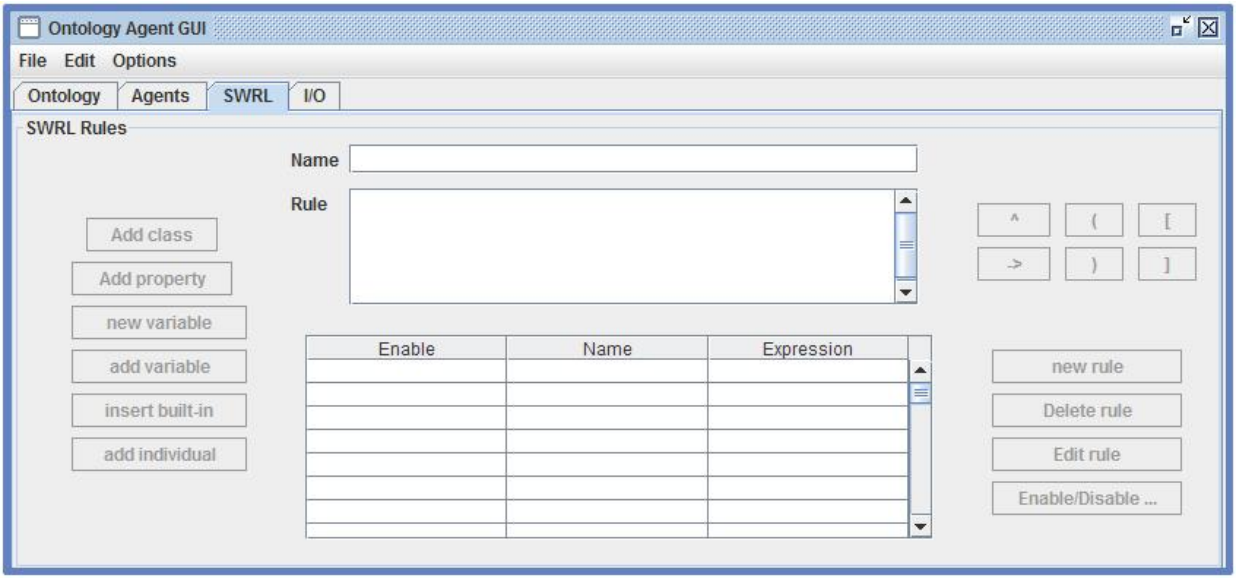

Figura 5.3.5 – Interface responsável pelo SWRL

Recorrendo à ontologia desenvolvida, pode ser criada, por exemplo a regra:

creates (?f,?b) ∧ meets (?a,?b) ∧ executes (?c,?d) ∧ describes (?e,?a) ∧ defines (?d,?g) ∧ isDefinedBy (?g,?h) ∧ has (?c,?i) → generates (?h,?i)

Esta regra diz que (em itálico estão definidas as classes da ontologia e entre parêntesis as variáveis SWRL utilizadas nas regras):

- ✓ *ProductDesign*(f) *creates TechnicalRequirments*(b)
- ✓ *Process*(a) *meets TechnicalRequirements*(b)
- ✓ *Equipment*(c) *executes* um *skill*(d)

- ✓ *Activity*(e) *describes Process*(a)
- ✓ *Skill*(d) *defines IntermodularReceptacle*(g)
- ✓ *IntermodularReceptacle*(g) *isDefinedBy genericProperties*(h)
- ✓ *Equipment*(c) *has Limitation*(i)

Então isto quer dizer que: *GenericProperties*(h) *generates Limitation*(i)

Ou seja, seria inferido na ontologia OWL a propriedade que diz que as *GenericProperties* geram (propriedade *generates*) limitações (classe *Limitations*).

Deste modo fica assim descrita toda a interface gráfica existente no agente que permite ao utilizador interagir com o sistema.

De seguida irá ser descrita a inicialização do sistema. Esta inicialização mostra os passos necessários a serem efectuados no início de forma a colocar o sistema em funcionamento e capaz de controlar o sistema de manufactura.

5.4 Interacção entre os agentes do sistema

Todos os agentes que compõem o sistema necessitam de comunicarem entre si. É necessário existir troca de informação para que uns agentes têm conhecimentos de outros agentes. Este subcapítulo demonstra como e quais as interacções existentes no sistema de modo a que o sistema funcione da maneira planeada e que o objectivo final seja cumprido.

Assim que o agente da ontologia inicia a primeira operação é registar-se no DF (*Directory Facility*) do JADE. Este registo inclui um serviço onde é atribuído um tipo e um nome. Para o caso específico deste agente o tipo de serviço designado foi *Ontology* e o nome do serviço foi *AgentOntology*. Com este registo feito, todos os outros agentes podem pesquisar por tipo ou nome do serviço de forma a encontrarem o agente responsável pela ontologia.

Todos os outros agentes presentes no sistema terão de efectuar um registo semelhante ao aqui efectuado pelo agente da ontologia.

Assim que todos os agentes se encontrem registados no sistema é então possível começar a troca de mensagens entre os diversos agentes. De seguida encontram-se descritas todas as

interacções possíveis de se efectuarem com o agente da ontologia. Todas as interacções entre o agente da ontologia e qualquer outro agente do sistema seguem a estrutura indicada na figura (Figura 5.4.1)

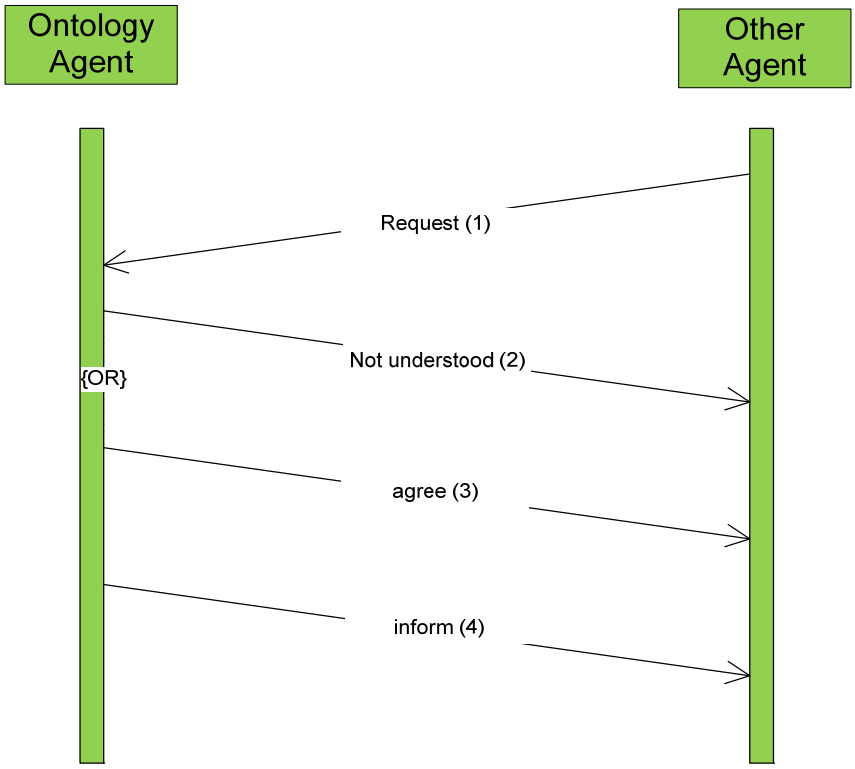

Figura 5.4.1 – Interacção entre um agente do sistema e o agente da ontologia

Todas estas interacções seguem o protocolo *Fipa-Request* existente na plataforma JADE. A primeira interacção apresentada na figura (interacção número 1) corresponde a uma mensagem de *Request*. Quer isto dizer que a mensagem recebida contém um pedido de um outro agente. No conteúdo da mensagem encontra-se qual o pedido específico. Estas especificações, de acordo com o protocolo, são:

- *Conversation ID*: Representa um ID de conversação atribuído para cada tipo de interacção;
- *Message Performative*: Existe uma lista de inteiro no JADE onde cada número corresponde a um tipo de mensagem diferente. Esta lista cobre todas as possíveis mensagens de serem enviadas pelo protocolo *Fipa-Request*;
- *Message Content*: Neste campo da mensagem encontra o conteúdo da mensagem que irá depender do objectivo do pedido. Irá conter dados importantes para a obtenção da resposta ao pedido.

Estas especificações são comuns a todas as mensagens, com as respectivas adaptações de modo a alterar o tipo de mensagem enviada.

A segunda interacção da figura corresponde ao agente da ontologia responder de imediato ao outro agente de forma a informá-lo se concorda (mensagem *Agree*) ou não (mensagem *not-understood*) com o pedido recebido. Esta mensagem é enviada sem conter qualquer resposta ao pedido. Tem como objectivo informar o outro agente se o pedido irá ser atendido ou não. Por exemplo se o agente não se encontrar registado na ontologia, será enviada a mensagem *not-understood* com o *conversation ID* igual a *AgentRegistration*, informando o agente que não se encontra registado. Caso o pedido vá ser efectuado (interacção 3 da figura), é então enviada uma mensagem do tipo *Agree* indicando ao agente que a resposta será enviada em breve.

Com última interacção da figura, é então enviada a resposta ao pedido. Esta mensagem é do tipo *Inform*, contendo o *Conversation ID* igual ao do pedido para o agente identificar melhor a mensagem e os dados da mensagem encontram-se no campo *Message Content*.

De seguida são descritos todos os tipos de pedidos que se podem efectuar ao agente da ontologia.

O primeiro pedido que se pode efectuar é o pedido de registo. Este pedido, além das interacções acima descritas, tem uma outra (do tipo *Refuse*) que é enviada caso o agente que efectue um pedido de registo já se encontre registado (Figura 5.4.2).

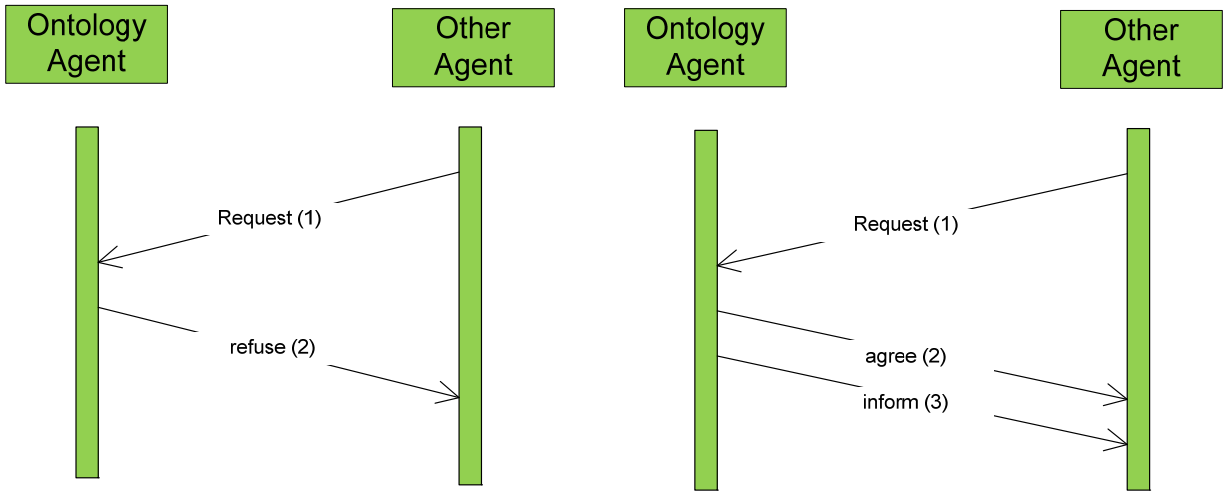

Figura 5.4.2 – Pedido de registo ao agente da ontologia (recusado e aceite)

Para um pedido de registo são necessários os seguintes parâmetros na mensagem:

- *Conversation ID*: definido como texto igual a "*AgentRegistration*";
- *Message Content*: Neste campo deverá estar o ficheiro XML de configuração do agente.

A resposta a esta mensagem será uma mensagem do tipo *Inform* com o mesmo *Conversation ID* que o pedido e sem conteúdo somente para informar que o registo foi efectuado.

É possível efectuar pesquisas na ontologia através deste agente. Para pesquisar um determinado *skill* (Figura 5.4.3) é enviada uma mensagem específica:

- *Conversation ID*: definido como texto igual a "*ask_skill*";
- *Message Content*: Neste campo encontra-se um objecto Java do tipo *hashtable* contendo um par de objectos. Esse par corresponde a dois objectos do tipo *String*. O primeiro será igual a *"SkillName"* e o segundo será o *skill* que se pretende obter.

A resposta a esta mensagem irá conter uma lista com os endereços de todos os agentes do sistema que oferecem como serviço aquele *skill*.

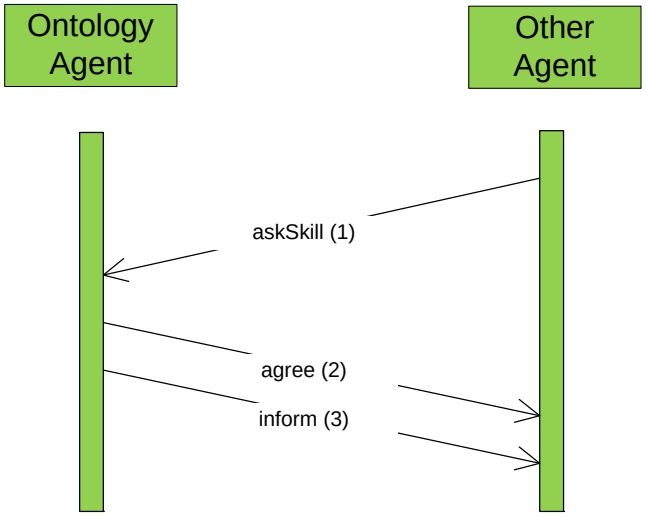

Figura 5.4.3 – Pedido de pesquisa por um *skill*

Outra pesquisa também possível de ser realizada através deste agente é uma pesquisa para obter os agentes que se encontram numa determinada posição geográfica, dada por coordenadas específicas:

- *Conversation ID*: Campo definido com texto igual a "*getAgentFromLocation*";
- *Message Content*: Este campo irá também conter um objecto do tipo *hashtable*, que representa uma tabela. Cada entrada da tabela contém dois objectos. Um com a coordenada ("x", "y" ou "z") e outro com o valor dessa coordenada.

O resultado desta pesquisa irá ser uma lista com o endereço de todos os agentes que se encontrem na posição especificada no conteúdo da mensagem. A localização do agente corresponde à localização física do componente que o respectivo agente controla.

Estas pesquisas são fundamentais nos sistemas evolutivos de produção. Elas permitem a existência de adaptabilidade e da reconfigurabilidade. Como prova dessas características, em

conjunto com este agente foram testados outros agentes de manufactura onde foi possível a criação de coligações de forma dinâmica. Estas coligações são formadas sempre que existe um requisito no sistema que não consegue ser satisfeito por nenhum outro agente presente no sistema isoladamente. Desta forma é gerado um agente que representa a união de diversos agentes obtendo assim capacidade de satisfazer esse requisito.

A última interacção com o agente da ontologia tem como função a actualização da informação existente sobre os diversos agentes. Ou seja, sempre que existem alterações no sistema, os agentes que sofreram essas alterações irão enviar um novo ficheiro XML, onde a informação registada na ontologia é então actualizada, permitindo à ontologia acompanhar todas as evoluções que o sistema sofra.

Para esta interacção é então necessário enviar uma mensagem específica (também do tipo *fipa request*):

- *Conversation ID*: Campo definido como texto igual a "*UpdateRegistration*";
- *MessageContent*: Neste campo da mensagem irá existir o ficheiro de XML, tal como na mensagem de registo, mas agora com as informações actualizadas.

Após ter sido finalizada a actualização da ontologia, é enviada uma mensagem do tipo *inform* a informar o outro agente que a actualização da ontologia foi efectuada com sucesso (Figura 5.4.4).

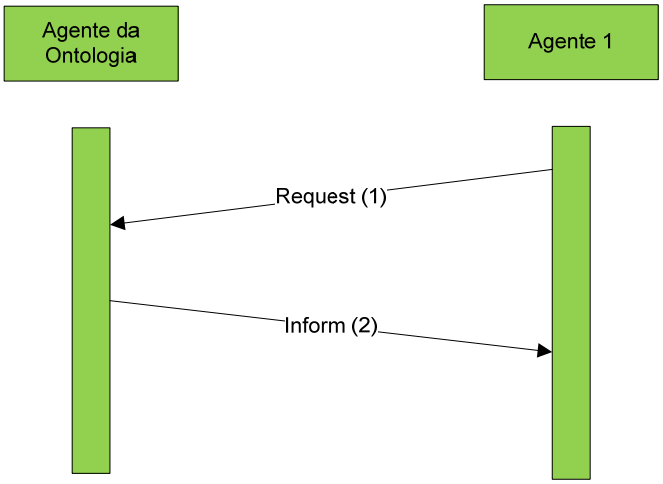

Figura 5.4.4 – Interacção para actualização da ontologia

Desta forma ficam finalizadas todas as interacções do agente da ontologia com os outros agentes pertencentes ao sistema. De seguida é apresentada a forma como o sistema é inicializado, até estar preparado a trabalhar.

5.5 Inicialização do sistema

Antes de todo o sistema estar operacional e pronto a controlar o sistema de manufactura, é necessário efectuar alguns passos de modo a que tudo fique interligado e operacional. Este subcapítulo pretende demonstrar quais são esses passos e de que maneira eles são efectuados.

Em primeiro lugar é necessário arrancar com o agente da ontologia. Isto porque todos os outros agentes do sistema se irão registar na ontologia no momento do seu arranque. Caso não exista ontologia, todos os agentes irão estar em erro por não encontrarem o agente da ontologia.

O agente da ontologia começa por carregar a ontologia para a sua memória. Em seguida regista-se no JADE (registo efectuado no DF que pertence ao JADE). Por último, o *Pellet* é arrancado, ficando assim o agente da ontologia pronto para qualquer interacção dentro do sistema. Todas estas operações são efectuadas pelo utilizador através da interface gráfica do agente (Figura 5.5.1).

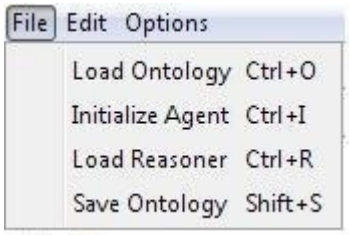

Figura 5.5.1 – Menus existente na interface que permitem iniciar o agente da ontologia

A última opção presente na figura (Figura 5.5.1) tem como função salvar toda a informação da ontologia num ficheiro específico. Desta forma é possível desligar o sistema e voltar a ligá-lo sem perder quaisquer informações sobre o mesmo.

A operação de carregar a ontologia consiste em aceder ao ficheiro que contêm a ontologia OWL e carregar todas a informações nele existente para o agente.

Em seguida é necessário iniciar todos os outros agentes do sistema. Para iniciar os outros agentes é necessário lançar a respectiva aplicação. Ao iniciar a aplicação cada agente irá ficar registado no DF de modo a poder pesquisar pelo agente da ontologia nesse mesmo DF.

Todos os agentes do sistema pesquisam no *Description Facilitator* (DF) (Bellifemine, Poggi et al. 1999) (referido no capítulo três) pelo serviço oferecido pelo agente da ontologia. Esta pesquisa irá resultar no endereço (único) deste agente. Com este endereço é então possível

iniciar-se uma conversação. Em primeiro lugar os agentes efectuam o seu registo na ontologia (Figura 5.5.2). Sem este registo nenhuma outra conversa poderá ser efectuada, evitando assim que qualquer outro agente que seja adicionado ao sistema inicie uma conversa sem que ninguém o conheça convenientemente. O registo de um agente na ontologia consiste no envio de um ficheiro XML contendo todas as informações (especificações eléctricas, dimensões, limitações, propriedades, serviços oferecidos, entre outras propriedades) desse agente que se referem à máquina que o respectivo agente controla.

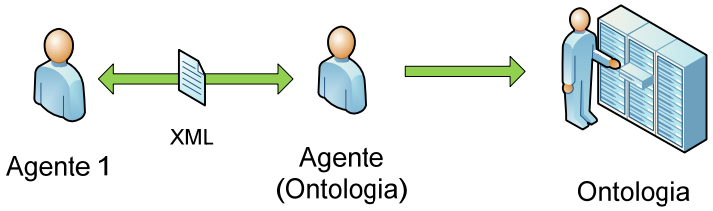

Figura 5.5.2 – Registo efectuado por um agente do sistema na ontologia

Após o registo ter sido efectuado com sucesso, é possível então realizar-se um conjunto específico de conversas com o agente da ontologia. Este conjunto de conversas representam um conjunto de mensagens à qual o agente está preparado para reagir e responder e que foram descritas anteriormente.

Após todos estes passos terem sido efectuados com sucesso o sistema fica então inicializado e pronto a actuar no sistema de manufactura.

No próximo capítulo é apresentado um caso de estudo onde todo este sistema multi-agente foi testado e comprovado.

5.6 Cenário de teste

De seguida irá ser descrito o sistema de manufactura que serviu de caso de uso para comprovar todo o funcionamento do sistema multi-agente.

Este sistema de manufactura é um sistema didáctico que se encontra no Departamento de Engenharia Electrotécnica da Universidade Nova de Lisboa. Este sistema tem a designação de MOFA France e a figura mostra esse mesmo sistema (Figura 5.6.1).

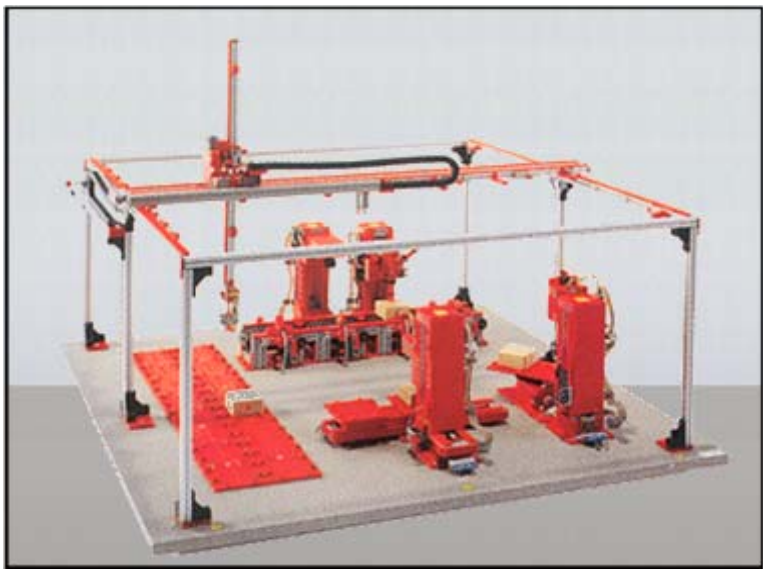

Figura 5.6.1 – MOFA France

O modelo MOFA France simula um sistema flexível de produção, constituindo um exemplo de um circuito fechado de manufactura, com o qual se torna possível criar diversas tarefas de montagem / fabrico possíveis por não ter qualquer função específica pré-definida.

Este sistema é constituído por um sistema de transporte, um sistema de armazém e um sistema de manufactura. O sistema de transporte consiste em três tapetes rolantes e uma grua. Ao contrário da grua, os tapetes são fixos e só conseguem transportar produtos cuja dimensão seja inferior ao tapete. A grua tem a capacidade de se deslocar por toda a planta fabril e através de um sistema de magnetos permanentes irá atrair o produto, que por sua vez também tem uma peça metálica por onde a grua irá segurar o produto. O sistema de armazém corresponde exclusivamente a um local da planta fabril que se encontra organizado por espaços onde é possível introduzir um único produto. O sistema de manufactura representa as máquinas que são

utilizadas para simular todas as operações de manufactura (como por exemplo, perfurar, lixar, entre outras).

Um exemplo de uma arquitectura multi-agente onde foi utilizado o mesmo modelo MOFA France, pode ser visto em (Barata, Cândido et al. 2007)

Capítulo 6. Conclusões

Com a criação de uma ontologia é possível descrever todo o sistema de manufactura. Tendo por base que a reconfigurabilidade é a tecnologia de suporte aos sistemas evolutivos de produção é possível criar um sistema distribuído sem que todos os seus módulos (agentes) tenham o conhecimento de todos os outros. Com a presença de uma ontologia neste tipo de sistemas (EPS/EAS) torna-se mais fácil todo o processo de evolução, de criação de coligações dinâmicas, tal como o sistema apresentado em (Ribeiro, Barata et al. 2008). A ontologia dos EPS/EAS (Semere, Barata et al. 2007) define a relação existente entre as entidades processo, produto e tarefa. Também na ontologia aqui definida essa relação existe.

A utilização de uma linguagem OWL-DL foi escolhida tem em conta o grau de expressividade necessário para a descrição de todo o sistema e de forma a ser possível suportar a implementação de regras utilizando a linguagem SWRL, seria necessário implementar a ontologia utilizando OWL-DL, visto ser a única linguagem compatível com SWRL. O facto de a linguagem OWL-DL permitir inferir conhecimento através de um *reasoner* (tal como o *Pellet* (Sirin, Parsia et al. 2007)), também foi um factor decisivo na escolha da linguagem.

De modo a melhorar o trabalho aqui efectuado, poderá ser feita uma maior investigação ao nível de regras SWRL e implementar um conjunto de políticas de modo a afectar a reconfigurabilidade do sistema. Para se estender a utilização da linguagem SWRL pode-se recorrer à linguagem SQWRL (*Semantic Query-Enhanced Web Rule Language*) que permite efectuar operações do tipo SQL formatando o conhecimento retirado da ontologia OWL.

Sabendo que se trata de um sistema distribuído, baseado em agentes, poderá ser interessante explorar a utilização de agentes móveis. Através da implementação de um agente

móvel, é possível fazer com que um determinado agente deixe de funcionar numa determinada máquina e passe a funcionar noutra, continuando no ponto onde tinha ficado, através de rede existente. Este tipo de agentes traria vantagens para a ontologia, no sentido em que a ontologia deixaria de estar centralizada, num único local físico, podendo "circular" pela rede. Por exemplo, se a ontologia se encontra-se num local congestionado da rede, esta poder-se-ia deslocar para outra máquina, noutro local da rede, menos congestionado, não impedindo o seu funcionamento.

Durante a realização deste trabalho, foi publicado um documento científico na conferência *Intelligent Manufacturing Systems 2008* (IMS'08), intitulado de "*OWL Ontology to Support Evolvable Production System"* (Ribeiro, Barata et al. 2008).

Capítulo 7. Bibliografia

Ait-Kaci, H. (2007). Description Logic vs. Order-Sorted Feature Logic.

Arnold, K., J. Gosling, et al. (1998). The Java Programming Language (Third Edition).

Baader, F. and W. Nutt (2002). Basic Description Logics. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, Cambridge University Press**:** 47-100.

Barata, J. (2003). Coalition Based Approach for Shop Floor Agility – A Multiagent Approach. Departement of Electrical Engineering Lisbon, New University of Lisbon. **PhD**.

Barata, J., G. Cândido, et al. (2007). A multiagent-based control system applied to an educational shop floor. Information Technology For Balanced Manufacturing Systems, Springer Boston. **220/2006:** pp. 119-128.

Barata, J., M. Onori, et al. (2007). Evolvable Production Systems: Enabling Research Domains. 2nd International Conference on Changeable, Agile, Reconfigurable and Virtual Production. CARV. Toronto, Canada.

Bechhofe, S., F. v. Harmelen, et al. (10 Fev, 2004). "OWL Web Ontology Language Reference." 2007, from http://www.w3.org/TR/owl-ref/.

Bechhofer, S., C. Goble, et al. (2001). "DAML+OIL is not Enough." First Semantic Web Working Symposium (SWWS'01).

Bellifemine, F., G. Caire, et al. (18-June-2007). "JADE programmer's guide." from http://jade.tilab.com.

Bellifemine, F., A. Poggi, et al. (1999). JADE – A FIPA-compliant agent framework. PAAM'99. London**:** 97-108.

Bray, T., J. Paoli, et al. (2008). "Extensible Markup Language (XML) 1.0 (Fifth Edition)." from http://www.w3.org/TR/REC-xml/.

Breitman, K. K. and J. C. S. d. P. Leite (8-12 Sept. 2003). Ontology as a requirements engineering product. Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International.

Brussel, H. V., J. Wyns, et al. (1998). Reference Architecture for Holonic Manufacturing Systems: PROSA. Computers In Industry, special issue on intelligent manufacturing systems. **Vol. 37, No. 3:** pp. 255 - 276.

Ferreira, B. D. (2009). Self-Organization and Complexity Theory to Support Evolvable Production Systems. Lisbon, FCT-UNL.

FIPA. (2002). "FIPA ACL Message Structure Specification." 2008, from http://www.fipa.org/specs/fipa00061/SC00061G.html.

FIPA. (2002). "FIPA Contract Net Interaction Protocol Specification." 2008, from http://www.fipa.org/specs/fipa00029/SC00029H.html.
FIPA. (2002). "FIPA Request Interaction Protocol Specification." 2008, from http://www.fipa.org/specs/fipa00026/SC00026H.html.
Frei, R., J. Barata, et al. (2006). Evolvable Assembly Systems basic principles. BASYS. Niagara Falls - Canada.
Frei, R., J. Barata, et al. (2007). Evolvable Production Systems Context and Implications. Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on. Vigo**:** 3233-3238.
Guarino, N. (1998). Formal Ontology and Information Systems. Proceedings of FOIS'98. Trento, Italy, IOS Press.
Hoda, E. (2006). "Flexible and reconfigurable manufacturing systems paradigms." International Journal of Flexible Manufacturing Systems **17(4)**(Reconfigurable Manufacturing Systems): 261-276.
Horrocks, I., P. F. Patel-Schneider, et al. (2004). "SWRL: A Semantic Web Rule Language Combining OWL and RuleML." 2008, from http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.
Horrocks, I., P. F. Patel-Schneider, et al. (2003). "From SHIQ and RDF to OWL: the making of a Web Ontology Language." Jornal of Web Semantics **1**(1): 7-26.
John, H. G., A. M. Mark, et al. (2003). "The evolution of Protégé: an environment for knowledge-based systems development." International Journal of Human-Computer Studies **58**(1): 89-123.
Kernighan, B. W. and D. M. Ritchie (1988). The C Programming Language (Second Edition), Prentice Hall, Inc.
Knublauch, H., R. W. Fergerson, et al. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. Third International Semantic Web Conference - ISWC - An architectural overview for developers and decision-makers. Hiroshima, Japan (2004).
Koestler, A. (1989). "The Ghost in the Machine", Arkana Books.
Koren, Y., U. Heisel, et al. (1999). Reconfigurable Manufacturing Systems. Annals of the CIRP. **48(2):** pp. 527-540.
Lohse, N., S. Ratchev, et al. (2006). Evolvable Assembly Systems - On the role of design frameworks and supporting ontologies. IEEE ISIE. Montréal, Québec, Canada.
McGuinness, D. L. and F. v. Harmelen. (10 Fev, 2004). "OWL Web Ontology Language Overview." 2007, from http://www.w3.org/TR/owl-features/.
Mehrabi, M. G., A. G. Ulsoy, et al. (April 2002). "Trends and perspectives in flexible and reconfigurable manufacturing systems " Journal of Intelligent Manufacturing **13(2)**: pp. 135-146.
Nardi, D. and R. J. Brachman (2002). An Introduction to Description Logics. Description Logic Handbook. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, Cambridge University Press**:** 5-44.
Noy, N. F. and D. L. McGuinness (2001). Ontology Development 101: A Guide to Create Your First Ontology**:** 25.
O'Connor, M. J., H. Knublauch, et al. (2005). Supporting Rule System Interoperability on the Semantic Web with SWRL. Fourth International Semantic Web Conference (ISWC2005), Galway, Ireland.
OIL. "Ontology Inference Layer." from http://www.ontoknowledge.org/oil/.
Onori, M. (2002). Evolvable Assembly Systems - A New Paradigm? In Proceedings of ISR2002 - 33rd International Symposium on Robotics. Stockholm.
Pine II, B. J. (1993). Mass Customization : the new frontier in business competition. Boston, Massachusetts, Harvard Business School Press.

Ribeiro, L., J. Barata, et al. (2008). OWL Ontology to support Evolvable Production System. IMS'08. Polland.

Semere, D., J. Barata, et al. (2007). Evolvable Assembly Systems Developments and Advances. Internacional Symposium on Assembly and Manufacturing, Michigan, USA.

Simon, P. (2006). "Spinning the semantic web edited by Dieter Fensel, James Hendler, Harry Lieberman and Wolfgang Wahlster, MIT Press, 479\ pp., $23, ISBN 0-262-56212-X." Knowl. Eng. Rev. **21**(1): 93-94.

Sirin, E., B. Parsia, et al. (2007). "Pellet: A Practical OWL-DL Reasoner." Journal of Web Semantics **5**.

Stroustrup, B. (2004). The C++ Programming Language (Third Edition), Addison-Wesley.

Tim, F., F. Richard, et al. (1994). KQML as an agent communication language. Proceedings of the third international conference on Information and knowledge management. Gaithersburg, Maryland, United States, ACM.

Wagner, F. (2006). Modeling Software with Finite State Machines: A Practical Approach, Auerbach Publications.