LEANDRO LOSS

INTEGRAÇÃO DE SISTEMAS MULTIAGENTE INDUSTRIAIS: UMA PROPOSTA BASEADA NO INTERCÂMBIO DE INFORMAÇÕES ENTRE PADRÕES

FLORIANÓPOLIS
2003

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

INTEGRAÇÃO DE SISTEMAS MULTIAGENTE INDUSTRIAIS: UMA PROPOSTA BASEADA NO INTERCÂMBIO DE INFORMAÇÕES ENTRE PADRÕES

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica

LEANDRO LOSS

Florianópolis, Fevereiro 2003

Integração de Sistemas Multiagente Industriais: Uma Proposta Baseada no Intercâmbio de Informações Entre Padrões

Leandro Loss

'Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em Controle, Automação e Informática Industrial, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina'

_____

Prof. Ricardo José Rabelo, Dr.
Orientador

_____

Prof. Edson Roberto de Pieri, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____

Prof. Ricardo José Rabelo, Dr.
Presidente

_____

Prof. Joni da Silva Fraga, Dr.

_____

Prof. Marcelo Ricardo Stemmer, Dr. Ing.

_____

Prof. João Carlos Espíndola Ferreira, Ph.D.

Aos meus pais, Daniel e Albina,

Por todo amor e apoio, muitas vezes psicológico e outras vezes financeiro. Sem vocês eu não teria chegado até aqui!

Agradecimentos

Talvez esta seja a parte mais difícil ao escrever uma dissertação… agradecer a todas as pessoas que estiveram presentes e colaboraram de uma maneira ou de outra para a conquista de mais um desafio nesta vida, espero não esquecer ninguém! Vamos lá…

Agradeço primeiramente a Deus, por me dar força e coragem de "largar" uma vida "pacata" no interior do estado rumo à realização de um sonho, o Mestrado.

Aos meus pais, por TUDO que fizeram, e não foi pouco, sempre preocupados que eu tivesse de bem com a vida. Tenho certeza, que se necessário, você meu pai, minha mãe, fariam muito mais. Muito Obrigado. Amo vocês.

A todos os colegas do GSIGMA que me acolheram ao chegar em Florianópolis e foram como uma família, compartilhando muitas dúvidas, incertezas e, sem dúvida alguma, muitas alegrias. Não seria justo se faltasse um agradecimento específico aos colegas GSIGMAnianos Fabiano Baldo, Rui Jorge Tramontin Júnior e Carlos Eduardo Gesser pelo companheirismo, auxílio na implementação do protótipo e todos os CBs – *Well Done!*.

Em especial, ao professor Ricardo José Rabelo, por acreditar no meu potencial, aceitar o "desafio" de ser meu orientador, e suas correções sempre visando um maior aprendizado. Muito Obrigado.

Aos meus amigos Giovani e Sônia, por todo o seu apoio.

Ao meu colega Fábio por todas as festas e pela paciência em dividir o apartamento comigo neste último ano!

A todos os funcionários do Programa de Pós Graduação em Engenharia Elétrica, professores do Departamento de Automação e Sistemas que proporcionaram o meu crescimento como aluno e como pessoa e aos membros da banca pelas sugestões e críticas que engrandeceram muito este trabalho.

Ao CNPq pelo suporte financeiro.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

Integração de Sistemas Multiagente Industriais: Uma Proposta Baseada no Intercâmbio de Informações Entre Padrões

Leandro Loss

Fevereiro, 2003

Orientador: Prof. Ricardo José Rabelo, Dr.
Área de Concentração: Controle, Automação e Informática Industrial.
Palavras Chave: MMS, XML, KQML, CORBA, Sistemas Multiagente.
Número de Páginas: 168

A complexidade dos sistemas industriais computadorizados tem aumentado muito devido a sua heterogeneidade, natureza distribuída e a necessidade de trabalhar com diferentes protocolos de comunicação. Esta tarefa faz a troca de informações – a chave para uma ampla integração – uma tarefa difícil. Esta tarefa se torna ainda mais complexa quando informações providas do chão de fábrica devem ser levadas em consideração, como também quando se pretende introduzir alguns níveis de inteligência e autonomia.

Este trabalho propõe uma abordagem para facilitar a integração na comunicação entre os sistemas industriais – desenvolvidos usando a tecnologia Multiagente – e dispositivos do chão de fábrica por meio de um ambiente global integrado. Este ambiente é aberto, faz uso de um número de padrões, e provê uma maneira transparente de integrar os padrões CORBA, KQML, MMS e XML em todos os níveis requeridos.

Um sistema multiagente industrial – sendo legado ou não – usualmente requer o envolvimento de um grande número de atores. Esta proposta foca em como a comunicação de sistemas multiagente industriais podem ser harmonizada e encapsulada, incluindo todos os atores necessários e, ao mesmo tempo, mantendo as propriedades essenciais de cada

protocolo dentro dos limites para os quais fora desenvolvido. Portanto, isso significa permitir a uma aplicação externa, que necessita acessar dados do chão de fábrica, não conhecer toda sintaxe e detalhes de todos os protocolos (dos vários tipos e níveis envolvidos) e as particularidades de cada controlador de máquina que se queira comunicar.

Com esta "camada de comunicação" a ser integrada numa aplicação industrial – e mais especificamente em sistemas multiagente industriais – pretende-se contribuir no suporte a interoperação entre sistemas – industriais e legados – e, assim, propiciar melhores condições de agilidade e competitividade às empresas.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements of the degree of Master in Electrical Engineering.

Industrial Multiagent Systems Integration: A Based Aproach in Information Exchange Among Standards

Leandro Loss

February, 2003

Advisor: Prof. Ricardo José Rabelo, Dr.
Area of Concentration: Control, Automation and Industrial Informatic.
Keywords: MMS, XML, KQML, CORBA, Multiagent Systems.
Number of Pages: 168

The complexity of computerized industrial systems has increased tremendously due to their heterogeneity, distributed nature and the need to work with different communications protocols. It makes the information exchange – the key for a wider integration – a very hard task to support. It becomes still more complex when information from the shop floor should be taken into account as well as some levels of intelligence and local autonomy is intended to be introduced.

This work proposes an approach to facilitate the integration in the communications among industrial systems - developed using Multi-agent Systems technology - and shop floor devices, by means of a global and integrated environment. This environment is open, makes use of a number of standards, and provides a transparent way to integrate the standards CORBA, KQML, MMS and XML in all the required levels.

An Industrial Multi-agent System – being legacy or not – usually requires the involvement of a large number of actors. This proposal focuses on how the communication

of Industrial Multi-agent Systems can be harmonized and encapsulated, comprising all the necessary actors and, at the same time, maintaining the essential properties of each protocol within the borders for which they have been developed/proposed. It means to allow any external application that needs to have access to shop floor data a leaner interface, without requiring to know all the syntax details of both the protocols used and the particularities of each machine controller's that it is necessary to communicate with.

With this "communication layer", that can be integrated in an industrial application – and more specifically in industrial multi-agent system, it is intended to contribute in the area of integration of industrial multi-agent system to better support the enterprise agility and competitiveness.

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

O desenvolvimento de sistemas industriais tem surgido para suprir as necessidades de agilidade que as empresas tem passado nos últimos anos, devido ao surgimento de novas tecnologias informatizadas, e as metas que o mercado impõe. Inúmeros são os requisitos para tal: um deles é a integração da informação por computador, CIM[1], que vem sendo aplicada por inúmeras empresas ao redor do mundo já há algum tempo (RABELO, 1997). A filosofia CIM busca garantir a integração das informações provenientes de vários setores de uma empresa (SILVEIRA, 1991). A integração tradicional dos equipamentos era feita for um único fabricante/vendedor e o sistema estava "preso" a um protocolo proprietário. Com o passar dos anos a integração dos equipamentos industriais heterogêneos (funcionalidades e diferentes protocolos) passou a visar o suporte ao compartilhamento de recursos, a evolutividade e escalabilidade do sistema. Um sistema industrial flexível é classicamente visto como um agrupamento de máquinas de comando numérico, robôs industriais, montagem automática, dispositivos de inspeção automática, interconectados por meios de sistemas de manuseio automático de materiais e de armazenamento, controlados por sistemas computadorizados integrados (SILVEIRA, 1991). Estas características têm possibilitado às empresas serem mais competitivas no mercado através da redução de custos, do aumento da qualidade dos seus produtos, proporcionando a entrega de uma diversidade de produtos em um tempo de resposta cada vez menor, exigido pelo mercado.

A preocupação com os aspectos de integração não é nova; várias propostas específicas tentaram solucionar os problemas de integração, sendo que muitas destas eram voltadas para aplicações particulares sem se levar em consideração o "mundo externo". Então, no começo dos anos 80 vários fornecedores iniciaram um processo de integração das funções de troca de informações no nível industrial com o objetivo de atender as necessidades do mercado. O resultado mais evidente é a arquitetura MAP[2], baseada no

---

[1] CIM – *Computer Integrated Manufacturing*
[2] MAP – *Manufacturing Automation Protocol*

modelo OSI[3], definida pela empresa General Motors (KUSIAK, 1989). Estes modelos MAP propunham uma integração entre os dispositivos de manufatura visando conectar as máquinas em uma rede de comunicação a fim de permitir a troca de informações entre estas e sistemas externos.

Estes primeiros modelos de arquiteturas trabalhavam com um único controlador/supervisor (ou sistema legado) para toda uma planta industrial; contudo, esta abordagem tornava difícil a escalabilidade dos sistemas, a possibilidade de implementar novas funcionalidades e uma melhor distribuição de tarefas. Como forma de solução destes problemas, e o progresso da relação desempenho/preço de poder de processamento computacional, as arquiteturas evoluíram de centralizadas, homogêneas e hierárquicas para arquiteturas distribuídas, heterogêneas e mais autônomas (QUINTAS, 1997). Por outro, lado surgiram uma série de dificuldades de gerenciar os sistemas com tais características (complexas) em seus diferentes níveis. A tecnologia dos sistemas multiagente vem surgindo como uma abordagem para auxiliar na solução de muitos desses problemas, dentre eles, problemas que envolvem a natureza distribuída, a manipulação inteligente de informação de diversas fontes e a comunicação entre processos.

Apesar dos resultados conseguidos, a comunicação apenas no chão de fábrica não é suficiente. Os níveis mais altos – de tomada de decisão – devem ter acesso às informações dos dispositivos industriais para uma melhor gestão destes e suas funções. A integração dos diversos níveis de informação nas empresas, principalmente do chão de fábrica com os demais níveis existentes, exige muitos esforços, pois não existe uma rede ou um protocolo único capaz de proporcionar todas as soluções das classes ou níveis de atividades existentes. Por outro lado, a adoção de tecnologias para uma integração transparente entre os diversos níveis não é uma tarefa simples, pois existem muitos formatos para transmissão das mensagens por exemplo e, conseqüentemente, um grande número destes formatos não segue um padrão comum, o que causa problemas de interoperabilidade entre níveis de comunicação existentes.

[3] OSI – *Open Systems Interconnection* ou *Open Systems Interconnect*

Tem-se verificado um verdadeiro *boom* no desenvolvimento dessas tecnologias, mas faltam propostas e/ou soluções para o uso integrado de todas elas considerando as características reais dos cenários industriais modernos. Existem inúmeros trabalhos relacionados com a integração e a utilização da informação nos diversos níveis das empresas, como por exemplo em (BARBER *et al.,* 1998), (USHER, 2000), (GUYONNET, *et al.* 2001) (RABELO, 1994) e (SILVEIRA, 1991). Porém, inúmeras das implementações relatadas nesses e em outros trabalhos têm coberto apenas aspectos individuais da comunicação e da integração da informação, cada um se restringindo às fronteiras naturais da aplicação e/ou da tecnologia empregada. Por exemplo, a comunidade multiagente tem colocado mais esforços na linguagem KQML[4] e ontologias, sem se preocupar muito com a robustez da infraestrutura de comunicação e com a integração com outros sistemas industriais abertos. A comunidade de redes de computadores tem desenvolvido protocolos de comunicação e interoperação, mas sem se preocupar com os protocolos de alto nível. A comunidade de redes industriais tem concentrado seus esforços nos seus níveis de protocolos de baixo nível e na integração entre os equipamentos de manufatura e redes de instrumentação, mas sem se ater aos requisitos de aplicação, também de alto nível.

Esta realidade tem construído um cenário no qual não se tem verificado esforços consistentes na direção de uma visão global da integração das informações, no sentido de se ter propostas que unam os vários esforços localizados, numa única plataforma. Esta visão global é tida como um grande requisito para sistemas industriais, e mais especificamente, em sistemas multiagente industriais. Isto favorece imensamente a integração da informação, base essencial para processos de decisão ágeis, consistentes e efetivamente coerentes com o real estado do chão de fábrica. O problema se torna ainda mais grave quando hoje se deseja que toda esta comunicação ocorra em plataformas e protocolos abertos, o mais padronizados possível, robustos, e suportando uma comunicação eficiente entre todos os atores envolvidos – incluindo pessoas – com o

[4] KQML – *Knowledge Query Manipulation Language*

objetivo de diminuir os problemas de interoperabilidade entre os protocolos de comunicação normalmente adotados.

A interoperabilidade é definida por KERN (1997) como a "interconexão efetiva de dois ou mais sistemas de computação, banco de dados ou redes diferentes com o propósito de suportar computação distribuída e intercâmbio de dados", ou como MISCHE, (2001) define, "a habilidade de fazer uma aplicação e uma tecnologia funcionar com outra de uma maneira que possa explorar as capacidades de ambas".

O objetivo geral desta dissertação é uma proposta que permite a o compartilhamento de conhecimento e comandos. Este compartilhamento ocorre entre os agentes que trabalham tanto com informações do chão de fábrica e dispositivos industriais (agentes industriais) quanto dos agentes de tomada de decisões (níveis mais altos), fazendo uso de tecnologias avançadas e protocolos abertos que seguem normas e padrões internacionais vindo a ser adotados. A questão principal é a capacidade de integrar diferentes ambientes, arquiteturas, tecnologias e, sobretudo, compreender cada área de atuação particular. Visando assim reduzir o problema da interoperabilidade entre os sistemas multiagente industriais, onde os dados precisam ser trocados entre aplicações que estão baseados em modelos distintos (níveis de tomada de decisões e níveis de atuação com os dispositivos industriais), criando a necessidade de tradução e localização destes dados entre as diversas aplicações existentes na comunidade. Este nível de integração é um dos mais complexos a serem alcançados devido à grande heterogeneidade nos equipamentos industriais existentes.

Para o problema abordado nesta dissertação propõe-se uma arquitetura de comunicação para a troca de mensagens entre os sistemas multiagente e os dispositivos de chão de fábrica. Mais que isto, a comunicação usa padrões internacionais que trazem suporte à interoperação. Nesta proposta usam-se quatro protocolos considerados padrão: a protocolo de comunicação KQML, a arquitetura de comunicação CORBA[5], o protocolo de

[5] CORBA – *Common Object Request Broker Architecture*

mensagens industriais MMS[6] e a linguagem XML[7], de forma integrada. Neste caso, a arquitetura de sistemas multiagente deve permitir a **integração** entre todos os **protocolos em um mesmo ambiente**.

Fisicamente, esta proposta é composta por uma biblioteca de classes, que deve ser integrada aos sistemas que a desejem utilizá-la, especificamente sistemas multiagente legados, buscando a combinação entre os agentes já existentes em um sistema multiagente e novos agentes que venham a se comunicar com eles.

Esta dissertação está organizada da seguinte forma. No capítulo 2 são apresentados os conceitos necessários para a compreensão das tecnologias e protocolos abordados nesta dissertação. Também são descritos trabalhos que fizeram uso destas tecnologias e protocolos na solução de problemas isolados ou parcialmente integrados. Um modelo conceitual sobre como pode ser a integração de um sistema multiagente industrial é proposto no capítulo 3. O capítulo 4 tem o intuito de descrever detalhes da implementação do protótipo para a validação do modelo conceitual, os requisitos de *hardware* e *software* para se utilizar a plataforma e integrá-la a sistemas multiagente "legados", e a discussão dos resultados obtidos na implementação. O capítulo 5 traz as conclusões da dissertação e sugestões para trabalhos futuros.

Ao final do documento são apresentados dois anexos (contendo a relação das principais mensagens KQML utilizadas por sistemas multiagente e a descrição dos principais serviços MMS), bem como um glossário de termos e siglas.

[6] MMS – Manufacturing Message Specification
[7] XML – Extensible Markup Language

2. Revisão Bibliográfica

Uma comunidade de agentes ou um sistema multiagente trabalha com a noção de troca de mensagens. A interação entre os agentes tem como objetivo compartilhar conhecimento e informações sobre um determinado escopo, visando a solução de um certo problema. Este tipo de integração é importante pois está relacionada ao fato de proporcionar às empresas, ou quem quer que esteja "utilizando" um sistema multiagente, informações atualizadas que auxiliam na tomada de decisões estratégicas e confiáveis. Como (MISCHE, 2001) cita em seu trabalho, "a maior razão para integração é a necessidade de se tornar mais competitivo em um ambiente que está constantemente em mudança e a alta competitividade entre as organizações". Conseqüentemente, a integração proporciona a capacidade do acesso à informação correta, no tempo e local corretos, com um formato interoperável, "qualidades" cruciais para capacitar o suporte e a agilidade na tomada de decisões. A agilidade na tomada de decisões é um requisito importante para organizações de sucesso em períodos de turbulência de mercado e mudanças sócio-econômicas imprevisíveis (CAMARINHA-MATOS *et al.* 2000), (KUSIAK, 1996).

A integração de tecnologias com os novos projetos e processos de negócios dentro das organizações permite a informação e o conhecimento serem compartilhados simultaneamente por trabalhadores, parceiros de negócios, e mesmo colaboradores e competidores, sem se preocupar onde as pessoas e/ou sistemas estão localizados (MISCHE, 2001).

No caso da comunicação com sistemas multiagente industriais, a troca de informações e o compartilhamento do conhecimento ocorrem nos/entre os diversos níveis dentro da indústria, desde o chão de fábrica até o nível de negociação e troca de conhecimento. Com o aumento das soluções de softwares, particularidades de tecnologias e hardware, o ambiente industrial tem se tornado muito heterogêneo. Por outro lado, o suporte de integração de sistemas existentes, considerando a sua heterogeneidade, de uma forma eficiente é também muito complexo e envolve o aspecto de interoperação entre sistemas (RABELO, 2001).

O intercâmbio de dados e informações, a necessidade de acesso aos dados entre as diversas aplicações existentes desenvolvidas – em linguagens de programação diferentes – o funcionamento em sistemas operacionais diferentes, localizações diferentes e o suporte a paradigmas diferentes formam as chamadas "ilhas de automação" ou "ilhas de produção" (WOBBE, 1996) (OSÓRIO *et al.* 2000), que dão origem aos problemas essenciais de interoperabilidade.

Este capítulo está dividido em três partes principais. A seção 2.1 faz uma explanação das tecnologias utilizadas no trabalho e o seu enquadramento dentro da proposta de comunicação apresentada. A seção 2.2 traz alguns trabalhos já realizados nas "subáreas" abordadas nesta dissertação, como por exemplo o interfaceamento dos dispositivos industriais, a comunicação entre sistemas multiagente, a comunicação distribuída entre dispositivos industriais, e aplicações dos sistemas multiagente no ambiente industrial. A seção 2.3 traz como funciona a abordagem tradicional de comunicação dos sistemas de manufatura.

2.1 Tecnologias Utilizadas

A seguir é feita uma breve descrição das principais características dos protocolos e tecnologias abordadas nesta dissertação, com o objetivo de familiarizar o leitor com os nomes e conceitos abordados, para somente depois fazer a apresentação de um texto mais completo sobre o assunto.

KQML – O KQML: *Knowledge Query Manipulation Language* (FININ *et al.* 1992, FININ 1994), (KQML, 2001), (LABROU, 1999, 1999b) é usado como uma linguagem de comunicação entre sistemas e hoje é considerado como a proposta padrão mais usada como linguagem de comunicação entre agentes. Em outras palavras, o KQML serve para permitir a troca de mensagens entre os agentes. O KQML é uma linguagem de comunicação orientada a mensagens e um protocolo para troca de informações de alto nível, independente da sintaxe de conteúdo e de uma *ontologia*[8] (GRUBER, 1992). Deste

[8] Ontologia – especificação dos objetos, conceitos e relacionamentos de uma dada área de conhecimento

modo, o KQML é independente do mecanismo de transporte (TCP/IP[9], SMTP[10], ou outro qualquer), independente de como o conteúdo da linguagem é representado (KIF[11], SQL[12], Prolog ou outros) e independente da ontologia assumida pelo conteúdo. A Figura 1 ilustra basicamente a troca de mensagens entre três agentes (A, B e C) utilizando uma das mensagens do protocolo KQML. Maiores detalhes sobre o KQML serão explanados mais adiante neste capítulo.

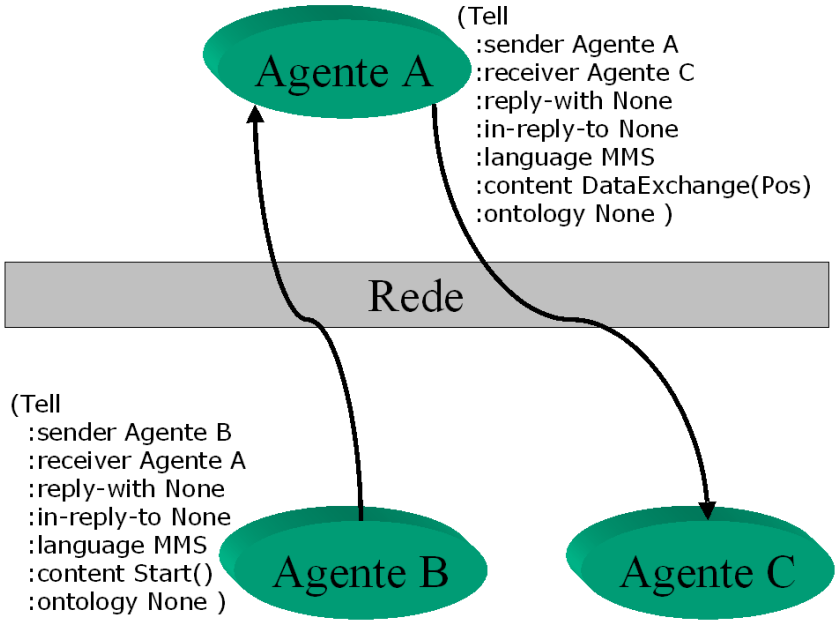


Figura 1 – Troca de mensagens entre agentes através do KQML

CORBA – O CORBA (*Common Object Request Broker Architecture*) (CHAFFEE, 1999), (PIRES, 1997), (SIEGEL, 2000), (TAJKUMAR, 2001), é uma arquitetura padrão que suporta o conceito de objetos distribuídos entre sistemas que também podem ser distribuídos. A arquitetura CORBA permite uma coleção heterogênea e distribuída de objetos interoperarem, definindo uma arquitetura para objetos distribuídos. Um objeto CORBA é uma entidade abstrata que representa um conceito concreto do mundo real. É formado por dados e métodos que podem ser chamados por outros objetos CORBA, pois

[9] TCP/IP – Transmission Control Protocol / Internet Protocol
[10] SMTP – Simple Mail Transfer Protocol
[11] KIF – Knowledge Interchange Format
[12] SQL – Structured Query Language

oferece um conjunto de atributos e operações que compõem a sua interface IDL[13] (THURAISINGHAM, 2001), descrita em uma linguagem declarativa na qual interfaces de objetos são publicadas de acordo com a arquitetura CORBA. A função da IDL é esconder a implementação do objeto. Um cliente do objeto simplesmente usa essa interface para chamar os métodos, não importando a localização, plataforma ou detalhes de implementação. Um dos mais importantes componentes CORBA é o ORB[14]. O ORB é um *middleware* que estabelece a relação cliente/servidor entre objetos (FRAGA *et al.*, 1997) (OBELHERO *et al.,* 2001), (LUNG *et al.*, 2000). O ORB implementa as requisições para os objetos remotos, provendo a interoperabilidade entre aplicações em diferentes máquinas em ambientes heterogêneos distribuídos. Isto quer dizer que o ORB provê mecanismos para fazer chamadas de procedimentos/funções/métodos de modo transparente, e receber respostas de objetos localizados local ou remotamente, sem que o cliente necessite ter conhecimento sobre os mecanismos usados para representar, comunicar com, ativar ou armazenar objetos. A Figura 2 ilustra a entrega de uma requisição de um serviço qualquer pelo ORB de um objeto cliente para a implementação do objeto.

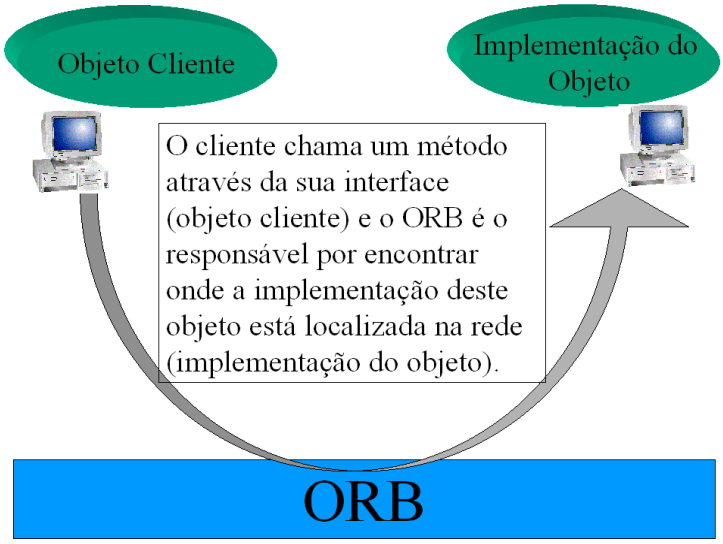


Figura 2 – Troca de Mensagens Através do ORB

[13] IDL – *Interface Definition Language*
[14] ORB – *Object Request Broker*

MMS – O MMS (*Manufacturing Message Specification*) (SISCO, 1995), (GUYONNET, 2001) (SISCO, 1998a) é um protocolo de mensagens padronizado internacionalmente para troca de dados e controle de informações entre dispositivos em rede e/ou aplicações de computadores. O MMS é independente da função da aplicação a ser executada (LEITÃO *et al.*, 1996). O MMS tem por objetivo facilitar a integração em ambientes industriais heterogêneos e distribuídos e dispositivos programáveis, tais como, controladores lógicos programáveis, controladores de robôs, comando numéricos e outros equipamentos de controle de processos. Estes dispositivos podem ser de fabricantes diferentes e complexidade diversa. O MMS permite o acesso a dados e o controle de uma vasta gama de sistemas e equipamentos de chão de fábrica. A Figura 3 ilustra a troca de mensagens utilizando o MMS.

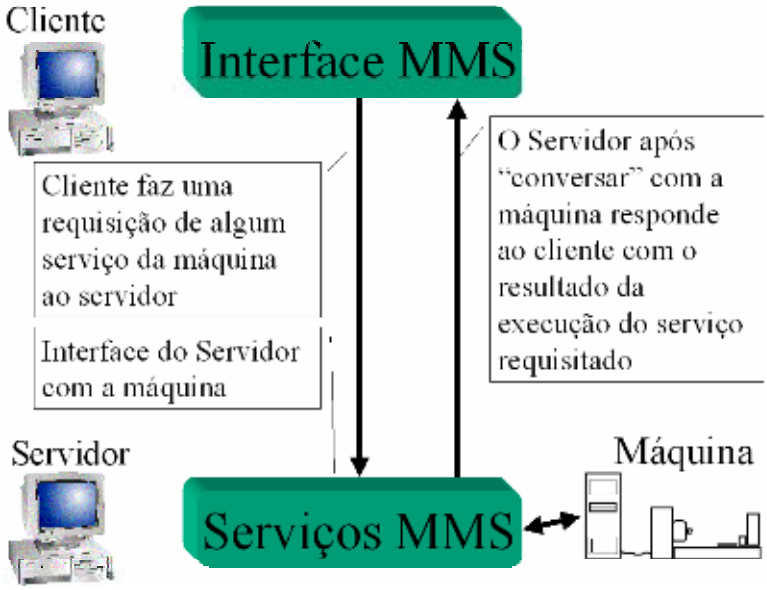


Figura 3 – Troca de Mensagens Através do MMS

XML – O XML (*Extensible Markup Language*) (MORRISON, 2000), (PIMENTEL *et al.*, 2000) é uma linguagem de "marcadores" (*tags*), tal como o HTML[15], e está se tornando o padrão para o intercâmbio de dados entre aplicações complexas e distribuídas. A linguagem XML foi criada para descrever dados (metadados), mas não se preocupa em como estes dados serão visualizados. Um documento XML é um recipiente

de informações para componentes reutilizáveis e customizáveis. Os *tags* XML são extensíveis, conseqüentemente podem ser definidos pelos desenvolvedores da aplicação com os nomes que melhor se adequam às aplicações. A estrutura das mensagens XML é previamente definida em um DTD[16] para descrever formalmente a sua estrutura. Um DTD provê uma gramática que diz quais estruturas podem ocorrer, e em qual seqüência. As mensagens XML podem ser usadas como fontes de dados em documentos HTML, armazenar dados em arquivos XML e, principalmente, como formato para troca de informações entre aplicações. A Tabela 1 mostra um exemplo de DTD para um documento XML. Na seqüência, a Figura 4 mostra o documento XML referente a este DTD. Este documento XML representa as informações contidas em uma mensagem KQML "modelada" em XML.

```
<!—DTD KQML DTD padrão para as mensagens KQML -->

<!ELEMENT kqml (KQML)>

<!ATTLIST kqml msg_id CDATA #REQUIRED>

<!ELEMENT KQML (Performative)>
    <!ELEMENT Performative (sender, receiver, reply-with, in-
reply-to, language, content, ontology, from?, to?)>

     <!ATTLIST Performative name CDATA #REQUIRED>
          <!ELEMENT sender (#PCDATA)>
          <!ELEMENT receiver (#PCDATA)>
          <!ELEMENT reply-with (#PCDATA)>
          <!ELEMENT in-reply-to (#PCDATA)>
          <!ELEMENT language (#PCDATA)>
          <!ELEMENT content (#PCDATA)>
          <!ELEMENT ontology (#PCDATA)>
          <!ELEMENT from (#PCDATA)>
          <!ELEMENT to (#PCDATA)>
```

Tabela 1 – Exemplo de DTD

[15] HTML – *Hyper Text Markup Language*
[16] DTD – *Document Type Definition*

DOCUMENTO XML

```
<KQML>
<Performative name="Tell">
    <sender>Agente A</sender>
    <receiver>Agente C</receiver>
    <reply-with>None</reply-with>
    <in-reply-to>None</in-reply-to>
    <language>MMS</language>
    <content>DataExchange(pos)</content>
    <ontology>None</ontology>
</Performative>
</KQML>
```

Figura 4 – Documento XML

2.1.1 Agentes e Sistemas Multiagente

Nesta seção será apresentada uma introdução sobre Agentes e Sistemas Multiagente, uma breve revisão sobre a linguagem de comunicação KQML, o protocolo MMS, a arquitetura CORBA e a linguagem de marcação XML, que servem como base deste trabalho.

2.1.1.1 Agentes

Agentes de softwares são uma "evolução" da Inteligência Artificial Distribuída (DAI[17]) (BITTENCOURT, 1998), um ramo da Inteligência Artificial (AI[18]) (BITTENCOURT, 2001) e (RABUSKE, 1995), que está relacionada com a solução cooperativa de problemas dentro de um certo ambiente (LESSER, 1999). Os seus objetivos são resolver problemas de qualquer natureza, trabalhando com a solução de

[17] DAI - *Distributed Artificial Intelligence*
[18] AI – *Artificial Intelligence* – mariores detalhes sobre Inteligência Artificial ver em BITTENCOURT, 2001

problemas distribuídos e a solução distribuída de problemas (FERBER, 1999), (ANDERL, 2000), (O'BRIEN, 2002). A solução de problemas distribuídos significa que os agentes podem estar distribuídos geograficamente e ter o mesmo nível de habilidade para auxiliar na solução, enquanto que na solução distribuída de problemas o conhecimento de como resolver o problema está distribuído entre os agentes, onde cada agente contribui com as suas habilidades individuais para a solução global (FERBER, 1999). Estas características ajudam os usuários em diversas tarefas distribuídas como monitoramento de sensores, manipulação de robôs ou mesmo na reconfiguração eficiente da planta de produção industrial. Um agente utiliza sua base de conhecimento embutida e aprendida sobre uma pessoa ou processo para tomar decisões e executar tarefas de um modo que realize as intenções do usuário. Existem diferentes definições para agentes, conforme Tabela 2:

SINGH (1998)	Um agente é um processo persistente que pode perceber o seu ambiente, argumentar e agir sozinho ou com outros agentes. Os conceitos chave para esta definição são interoperabilidade e autonomia.
WEINSTEIN (1999)	Agentes são processos computacionais que operam sem a supervisão direta de humanos, e eles interagem com outros agentes para executar tarefas ou caso contrário perseguir os seus objetivos.
ANDERL (2000),	Um agente é definido como um sistema ou um programa em um ambiente, que está atento ao seu ambiente e influencia no seu contexto sobre suas tarefas (uma possibilidade infinita) que são afetadas pelas influências no ambiente.
VIEIRA (2000)	Um agente é uma entidade que reside em determinado ambiente onde interpreta dados percepcionais que refletem eventos e estados desse ambiente e que tem a capacidade de, de maneira autônoma, executar ações que produzem efeitos sobre o

	ambiente.
O'BRIEN (2002)	Os agentes são sistemas de informação baseados em conhecimento com finalidades especiais que executam tarefas específicas para os usuários.

Tabela 2 – Definições de agentes

Apesar da definição de Agente ser uma questão essencial, ela é uma das questões de menor consenso na comunidade científica de Inteligência Artificial Distribuída, pois a definição de agente é fortemente influenciada pelo domínio da aplicação e do problema, das formas de cooperação e dos seus níveis de autonomia (RABELO, 1997).

Os aspectos essenciais que diferenciam um agente de outras entidades, tais como processos e objetos, são classificados por FERBER (1999) em: agentes de hardware e agentes de software, agentes estacionários e móveis, agentes persistentes e temporários (RABELO, 1998) e agentes reativos e cognitivos.

Agentes de Hardware estão posicionados em um dado ambiente, normalmente "fechado", são guiados por uma função de satisfação/sobrevivência e possuem recursos próprios (ferramentas, atuadores, etc.). Este tipo de agente capaz de perceber o seu ambiente, mas normalmente de uma forma muito limitada; raramente possui alguma representação do seu ambiente. O seu comportamento segue estritamente a sua função de satisfação, considerando os recursos que possui (FERBER, 1999).

Os Agentes de Software estão posicionados em um dado ambiente, quase sempre "aberto", são guiados por um conjunto (dinâmico) de objetivos e possuem recursos próprios. Estes agentes tem uma representação parcial dos outros agentes, isto é, conhecem o funcionamento parcial dos outros agentes; também possuem certas habilidades, tipicamente na forma de "serviços" que podem ser oferecidos aos demais agentes. O seu comportamento segue seus objetivos, considerando não apenas os recursos que possui, mas o que sabe dos outros agentes e o que recebe dos outros agentes. Eles podem se comunicar com os outros agentes (FERBER, 1999).

Os Agentes Estacionários são agentes de software que, uma vez lançados em um dado ambiente computacional, não têm a habilidade de se moverem pela rede para outros ambientes e/ou computadores, já os Agentes Móveis são agentes de software que podem se mover para outros ambientes através da rede e, quando se movem, levam consigo seus "estados internos" (representação + memória) (FERBER, 1999).

Os Agentes Persistentes são agentes de software que, uma vez lançados em um dado ambiente computacional, não são ou não podem ser excluídos do sistema. Os Agentes Temporários são agentes de software que tem uma vida finita, normalmente de duração igual ao tempo de uma dada tarefa, ou seja, tão logo eles finalizam sua "missão", eles são excluídos do sistema (normalmente por eles próprios) (RABELO, 1998).

A divisão entre agentes Reativos e agentes Cognitivos é considerada a mais importante divisão de classificação para agentes.

O objetivo dos agentes reativos é "sobreviver". O objetivo não é explicitamente representado. Estes agentes possuem ações baseadas em estímulo-resposta, conseqüentemente não raciocinam sobre o mundo, não têm "memória", isto é, não armazenam informações sobre as ações já realizadas, e raramente interagem com outros agentes, e por isso agem reativamente. Esta classe de agentes tem como característica não se planejar para futuras ações. Agentes reativos trabalham com aplicações de baixa complexidade e normalmente estão em grande quantidade em um sistema. Quando trabalham em comunidade podem ter um comportamento global inteligente. Podem ser citados como exemplos para agentes reativos as aplicações na área de robótica móvel terrestre, aérea espacial, submarina, sistemas de transporte, sistemas de simulação (FERBER, 1999).

O objetivo dos agentes cognitivos é "cooperar". Estes agentes possuem objetivos explicitamente representados, ações baseadas em intenções (racionais), raciocinam sobre o mundo, têm memória e interagem com outros agentes e ambientes, e por isso, agem inteligentemente. Os agentes classificados como cognitivos possuem um plano de ações construído/adaptado dinamicamente com planejamento para futuras ações, também trabalham com atividades de complexidades média/alta e são encontrados, normalmente,

em pequenas quantidades em um sistema. Os agentes cognitivos requerem sofisticados mecanismos de coordenação e protocolos de alto nível de suporte à interação. Podem ser citados como exemplos para agentes cognitivos as aplicações nas áreas de planejamento de sistemas; negociação em bolsa de valores; alocação de tarefas; tutores; controle de tráfego aéreo (FERBER, 1999).

Levando-se em consideração as questões citadas anteriormente um agente pode ser definido como: "Um agente é um sistema computacional, posicionado em algum ambiente, que é capaz de agir com autonomia flexível visando atingir os objetivos para o qual foi projetado" (FERBER, 1999).

O posicionamento refere-se aos termos em que o agente recebe sinais de entrada dos seus sensores vindos do ambiente do qual está localizado, e de que ele pode executar ações contextualizadas que modifiquem de alguma forma o ambiente.

A autonomia quer dizer que o agente deve ter a possibilidade de agir sem a intervenção direta de usuários ou de outros agentes, e de que deve poder controlar totalmente suas ações e seu estado interno.

A flexibilidade em termos de como o agente atua para atingir os seus objetivos. Envolve as capacidades de:

- Receptividade – Os agentes devem poder perceber o seu ambiente e responder adequadamente às mudanças que ocorrem nele.
- Pró-Atividade – Os agentes não devem apenas agir em resposta ao seu ambiente, mas devem agir oportunisticamente por iniciativa própria de acordo com seus objetivos.
- Sociabilidade – Os agentes devem poder interagir, quando apropriado, com outras entidades do ambiente de forma a finalizar seus problemas e a ajudá-las nas suas atividades.

Esta definição permite caracterizar uma arquitetura para os agentes que compreende algumas funções:

- Percepção – responsável por "sentir" o ambiente onde o agente está imerso, filtrar a informação e produzir descrições do ambiente relevantes para o agente;
- Ação – responsável por provocar no ambiente os estímulos que o agente pretende imprimir;
- Tomada de Decisões – concentra todos os aspectos cognitivos do agente como o objetivo de cooperação, ações baseadas em intenções, planejamento para ações futuras e um comportamento individual.

Algumas características adicionais habitualmente consideradas importantes para agentes segundo VIEIRA (2000):

- Mobilidade – característica interessante em muitos domínios, associada à capacidade que o agente tem de se mover e de se inserir em ambientes diferentes.
- Continuidade – associada ao fato de o agente não ser, em princípio, uma entidade computacional transitória, e sim, possuir uma característica de funcionamento contínuo.
- Adaptabilidade - corresponde à capacidade que o agente possui de se adaptar aos seus ambientes.

Embora estas características não tenham que aparecer em simultâneo em todos os agentes, elas abrangem um espectro vasto de requisitos considerados potencialmente úteis para os agentes.

A arquitetura de agentes é concebida para permitir a um agente executar as tarefas para atingir os seus objetivos que contribuem para o sistema como um todo. Este agente deve ser capaz de participar em uma sociedade dirigida por alguns conjuntos de protocolos básicos. Esta participação requer que um agente se comunique com outros objetos (incluindo agentes) no seu ambiente e leve em consideração as responsabilidades que irão afetar diretamente ou indiretamente o ambiente. As linguagens para comunicação de agentes permitem agentes se comunicarem uns com os outros, os auxiliando em várias tarefas, dentre elas, sobre como particionar e executar trabalhos complexos. Conseqüentemente, a identificação e definição destes protocolos para a

indústria são usadas como base para caracterizar um importante requisito de uma arquitetura de controle.

2.1.1.2 Sistemas Multiagente

A tecnologia de sistemas multiagente tem surgido como uma alternativa para apoiar uma resolução cooperativa de sistemas distribuídos, apresentando vantagens se comparado com as arquiteturas monolíticas tradicionais. Sistemas Multiagente (*Multi-agent Systems*) são sistemas baseados em agentes onde grupos de agentes trabalham juntos como um único sistema para integrar suas funcionalidades (NODINE, 1999). Eles consistem em grupos de agentes que operam entre si, cooperando na execução de várias tarefas de maneira distribuída visando suportar a capacidade de tomada de decisões de maneira descentralizada, a escalabilidade e a flexibilidade (RABELO *et al.* 2000b). Cada agente é normalmente um especialista em uma tarefa ou sub-tarefa particular. Em geral um sistema multiagente corresponde a "resolvedores de problemas em rede" que trabalham juntos para resolver problemas que estão além das suas capacidades individuais (RABELO *et al.,* 2000a).

Considera-se um Sistema Multiagente Industrial um sistema multiagente que está envolvido no ambiente industrial, tanto na troca de informações de alto nível, negociação entre fornecedores, clientes, etc., como no intercâmbio de informações dentro da indústria, dados que circulam desde o nível de chão de fábrica até os níveis de administração/gerência.

Um sistema multiagente industrial usualmente requer o envolvimento de vários atores, por exemplo, os agentes propriamente ditos, os dispositivos industriais e seus serviços legados – quando existirem – e os esquemas utilizados para a troca/compartilhamento de informações, que variam desde protocolos de infraestrutura básica de comunicação até o suporte às aplicações, através do compartilhamento do conhecimento.

Características dos Sistemas Multiagente

Sistemas multiagente são caracterizados por suas habilidades para alcançar seus objetivos através de comunicação e colaboração entre si. Agentes em um sistema multiagente usualmente têm procedimentos simples que podem ser descritos por um conjunto de protocolos de comunicação. Sistemas multiagente são usualmente distribuídos – estar em várias máquinas e/ou plataformas e concorrentes – podem ser descritos como processos distribuídos que se comunicam entre si (WEN, 1999).

São comumente associadas as seguintes características aos sistemas multiagente:

- Desenvolvimento Modular – os agentes permitem dividir os problemas complexos, permitindo que na fase de desenvolvimento o foco seja na atividade mais simples de cada agente, possibilitando assim uma forma efetiva de controlar a complexidade. Esta característica permite também atingir níveis de escalabilidade elevados, dado que os agentes podem entrar e sair de uma comunidade à medida das necessidades sem ser necessário reiniciar todo o sistema. Um agente pode ser substituído por outro mais atual quando as necessidades de atualizações o exigem.
- Tolerância a Falhas – normalmente os agentes não interagem uns com os outros de forma rígida, pré-definida, mas de forma oportuna, tirando proveito das circunstâncias e dos agentes presentes em cada momento. Em caso de falha de um ou vários agentes, é menos provável que o sistema entre em colapso se comparado com as normas tradicionais (menos flexíveis) de implementação de processos centralizados.
- Redução de Custos e Melhor Desempenho – os custos de software tendem a diminuir dada a decomposição funcional inerente aos sistemas multiagente. Por outro lado, dado que se pode explorar o paralelismo, podem ser utilizados muitos processos simples e mais baratos em vez de alguns muito caros. A exploração do paralelismo também possibilita melhoria no desempenho.

Vantagens de Sistemas Multiagente na Indústria

A aplicação de sistemas multiagente baseado no conceito de inteligência artificial distribuída é uma abordagem de controle promissora para a próxima geração industrial.

Os sistemas são compostos de agentes heterogêneos distribuídos e fazem uso de mecanismos de controle flexíveis para criar e coordenar a sociedade de agentes resultantes (USHER, 2000). Esta sociedade de agentes provê uma arquitetura que possua a capacidade de beneficiar a indústria aumentando a confiabilidade – a habilidade do sistema operar continuamente em casos de falhas dos sistemas; fácil manutenção – possibilidade de mudanças e extensibilidade: a habilidade de mudar os elementos dentro do sistema facilmente ou adicionar um novo componente no sistema; flexibilidade – reconfiguração e adaptabilidade: habilidade de aumentar ou expandir as suas funcionalidades para se manter atualizado com as mudanças tecnológicas; recuperação de erros e estabilidade do sistema, como também provendo meios para planejamento em tempo real.

Nos domínios de automação e manufatura (industriais), um agente é um módulo de software que representa objetos de manufatura e automação como CNCs[19] de máquinas, CLPs[20], robôs e sensores.

Problemas Inerentes aos Sistemas Multiagente

Ao se escolher a abordagem multiagente para a solução de um determinado problema/tarefa é importante levar em consideração alguns aspectos inerentes os sistemas multiagente, como por exemplo, a formulação, descrição, decomposição e alocação de problemas, e agrupamento de resultados entre um grupo de agentes que podem estar distribuídos ou não. Os agentes devem estar capacitados à comunicação, permitindo a troca de informações e o aumento da eficiência nos "serviços" prestados pelo grupo; deve-se levar em consideração as linguagens e protocolos de baixo e alto níveis que serão usadas. Outra preocupação importante para decidir é o que e quando comunicar e assegurar que os agentes vão agir de forma coerente nos vários tipos e níveis de ações/interações, acomodando os efeitos de decisões locais de outros agentes e evitando interações indesejadas. Deve-se capacitar cada agente a representar e raciocinar acerca das ações, planos/intenções e conhecimento dos outros agentes, de forma a agir

[19] CNC – Comando Numérico Computadorizado
[20] CLP – Controlador Lógico Programável

coordenadamente com eles, buscando soluções sobre os estados determinantes de início e finalização de tarefas. Os agentes devem ter a capacidade de reconhecer e lidar com pontos de vista antagônicos e conflitos de planos entre os vários agentes que tentam coordenar suas ações, visando controlar e agir diante de um comportamento global caótico do sistema (WOOLDRIDGE, 1998a), (WOOLDRIDGE, 1998b).

2.1.2 Ontologias

Segundo LABROU *et. Al* (1999b), NOY (2002), CHANDRA *et al.* (2000) e SOUZA *et al.* (2000), todo agente incorpora alguma visão do domínio que ele está aplicado. O termo técnico para este corpo de conhecimento de fundo é *ontologia*. Mais formalmente, uma ontologia é uma conceitualização particular de um conjunto de objetos, conceitos e outras entidades sobre qual o conhecimento é expressado, assim como os relacionamentos existentes entre eles (BOKMA, 2000). Uma ontologia define um vocabulário comum para aqueles que necessitam compartilhar informações de um determinado domínio (neste caso, dispositivos de chão de fábrica), e assim, reduzir ou eliminar a confusões conceituais e terminológicas.

As ontologias têm se estabelecido como um método poderoso para o compartilhamento de conhecimento, e um grande número de aplicações tem se beneficiado do uso delas como meio de alcançar combinação semântica entre sistemas agentificados heterogêneos e distribuídos (LUIGI, 2002). Uma representação adequada do conhecimento não deve perder qualquer conhecimento útil, nem criar redundâncias, possibilitando a transformação do conhecimento, abstrações e compartilhamentos (HEREDIA *et al.*, 1996).

Existem vários motivos para o uso de ontologias, dentre estes, os que merecem maior destaque são (NOY, 2002), (CHANDRA *et al.*, 2000), (SOUZA *et al.* 2000):

- Compartilhar o entendimento comum da estrutura de informação entre os agentes ou mesmo entre pessoas. Pode-se considerar que este é o maior objetivo no desenvolvimento de ontologias. Quando os agentes compartilham e publicam suas informações, outros agentes podem extrair e agregar informações relevantes à

sua área de atuação. Assim é possível agregar informações dos agentes para responder as dúvidas dos "usuários" – agentes ou não – ou na entrada de dados para outras aplicações.

- Possibilitar o reuso do conhecimento, um outro grande esforço por trás das pesquisas mais recentes sobre ontologias. Se um grupo de pesquisadores desenvolve uma ontologia em detalhes, outros pesquisadores podem simplesmente reusar este "conhecimento" nos seus domínios.
- Separar o domínio do conhecimento do domínio operacional é outro uso comum das ontologias. É possível a configuração de uma tarefa ou um produto independente dos produtos ou dos componentes usados. É possível, por exemplo, desenvolver uma ontologia para descrever o domínio de como proceder no armazenamento de vinhos e desenvolver um algoritmo para executar uma tarefa de armazenar estes vinhos. Sendo uma ontologia, o algoritmo de como armazenar os vinhos pode ser utilizado também com o domínio de outras bebidas, como por exemplo, sucos e refrigerantes, ou outro qualquer, pois ele é independente do conhecimento relativo ao produto.

É necessário lembrar que uma ontologia é um modelo de realidade do mundo e os conceitos na ontologia devem refletir esta realidade, proporcionando respostas para que os agentes, ou qualquer outra entidade interessada em enviar e/ou obter informações dos dispositivos industriais, saibam como proceder neste processo.

A representação dos equipamentos industriais em uma CFM tem por objetivo disponibilizar todos os dados técnicos sobre os equipamentos, quais as funções que podem ser executadas por eles e as características das máquinas.

No contexto do compartilhamento de conhecimento, ontologias são especificações na forma de definições de um vocabulário representativo. Um simples caso seria um tipo de hierarquia, especificando as classes e suas relações. Um banco de dados relacional também serve como ontologia através das especificações das relações que podem existir nas tabelas de dados compartilhadas e nas regras de integridade que oferecem suporte

Uma ontologia necessariamente vincula ou inclui algum tipo de "visão geral" referente a um domínio determinado. Esta "visão geral" é freqüentemente concebida como um conjunto de conceitos (por exemplo: entidades, atributos, processos), suas definições e suas inter-relações. Isto se chama uma conceitualização *(conceptualisatio*n). Uma conceitualização pode estar concretamente implementada, como por exemplo, em um componente de software, ou pode ser abstrata, definida em linguagem natural de uma forma não formal (MONTESCO, 2001).

Uma conceitualização pode tomar uma variedade de formas, mas necessariamente ela incluirá um vocabulário de termos e alguma especificação dos seus significados (por exemplo: definições). A maneira como estes vocabulários são criados e especificados tem graus de formalidade que variam consideravelmente (MONTESCO, 2001).

- Altamente informal: expresso livremente em linguagem natural.
- Semi-informal: expresso em uma forma restrita e estruturada em linguagem natural.
- Semi-formal: expressa em uma linguagem artificial definida formalmente.
- Rigorosamente formal: termos meticulosamente definidos com uma semântica formal, teoremas e provas de tais propriedades como completitude e boa qualidade.

2.1.3 KQML como uma Linguagem de Comunicação

Em soluções baseadas em sistemas distribuídos, como é o caso dos sistemas multiagente, os aspectos de comunicação e integração assumem fundamental importância. A comunicação é importante pelo fato das entidades envolvidas devem poder trocar informações entre si. A integração é importante pelo fato das entidades serem completamente heterogêneas entre si e as informações estarem armazenadas em formatos igualmente heterogêneos (RABELO, 1997). Desta forma, para haver comunicação efetiva é necessário dialogar em protocolos comuns e ter as informações de formas acessível.

O KQML é um protocolo para a troca de informações e conhecimento. O KQML foi proposto pela Agencia de Projetos e Pesquisa de Defesa Avançada do Departamento de Defesa dos Estados Unidos (DARPA – *Defense Advanced Research Projects Agency*) no final dos anos 80 com o intuito de desenvolver um protocolo para troca de conhecimento entre sistemas de informação autônomos. O KQML foi concebido como um formato de mensagem e também um protocolo de manipulação de mensagens para sustentar o compartilhamento *run-time* de conhecimento entre agentes ou entre processos quaisquer. O KQML pode ser usado como uma linguagem para um programa de aplicação interagir com um sistema ou para dois ou mais sistemas compartilharem conhecimentos para a solução de problemas de forma cooperativa. O KQML inclui uma série de primitivas que os agentes usam para "comunicar" fatos, fazer perguntas, encontrar grupos de outros agentes, etc. (SINGH, 1998).

BENECH *et al.,* (1997) define as características-chave do KQML:

- As mensagens KQML não somente comunicam sentenças em alguma linguagem, mas mais do que isto, elas comunicam uma "atitude" sobre o conteúdo (afirmação, requisição, pergunta, respostas básicas, etc.).
- As primitivas da linguagem são chamadas de "performativas" (*performatives*). As performativas definem ações possíveis (operações) ou os tipos de interações que os agentes podem usar na comunicação com outros agentes.
- KQML assume que ao nível de agente a comunicação apareça como uma mensagem ponto a ponto.
- Um ambiente onde os agentes "falam" KQML pode ser enriquecido com agentes especiais, chamados facilitadores, que provêm para os agentes funções adicionais para trabalhar em forma de rede (associação de endereços físicos com nomes simbólicos, registro de agentes e/ou serviços oferecidos e requisitados pelos agentes, melhoria nos serviços de comunicação como encaminhamento, difusão, entre outros).

As Camadas KQML

As mensagens KQML são estruturadas em três camadas: a camada de conteúdo, a camada de mensagem e a camada de comunicação (AZEVEDO *et al.* 1999), (DIPIPPO *et al.*, 2001) e (LABROU *et al.,* 1999).

A camada de conteúdo suporta o conteúdo atual da mensagem na própria representação da linguagem do agente. Toda implementação KQML ignora a porção de conteúdo da mensagem, exceto para determinar onde a mensagem termina. Esta camada exibe como característica o conteúdo atual da mensagem em uma linguagem de representação de conhecimento de acordo comum (AZEVEDO *et al.* 1999).

A camada de mensagem é usada para codificar uma mensagem que uma aplicação gostaria de transmitir para outra (AZEVEDO *et al.* 1999).

A camada de comunicação codifica um conjunto de características de mensagens que descreve os parâmetros de baixo nível de comunicação, como a identidade de quem está enviando, o destinatário e um identificador único associado com a comunicação.

A camada de mensagem forma a essência da linguagem KQML e determina os tipos de interações que podem haver com agentes que "falam" KQML. A principal função da camada de mensagem é identificar o protocolo de rede a ser usado para entregar as mensagens, atender as performativas KQML e as performativas KMQL que o emissor anexa ao conteúdo em uma afirmação, uma consulta, um comando ou qualquer performativa existente (LABROU *et al.,* 1999). Esta camada também inclui características opcionais que descrevem o conteúdo da linguagem, a ontologia que ela assume e alguns tipos de descrição de conteúdo, como um descritor nomeando um tópico dentro de uma ontologia. No contexto de compartilhamento de conhecimento, uma ontologia é uma especificação de uma conceitualização. Isto é, uma ontologia é uma descrição (como a especificação formal de um programa) dos conceitos e relacionamentos que podem existir para um agente ou uma comunidade de agentes.

A Figura 5 ilustra como a mensagem KQML está dividida nessas três camadas: a camada de conteúdo, a camada de mensagem e a camada de comunicação.

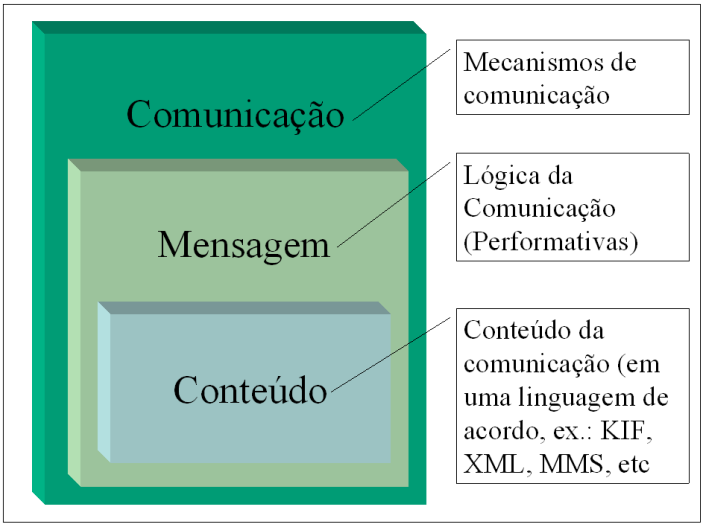


Figura 5 – Camadas do KQML

As Performativas Reservadas do KQML

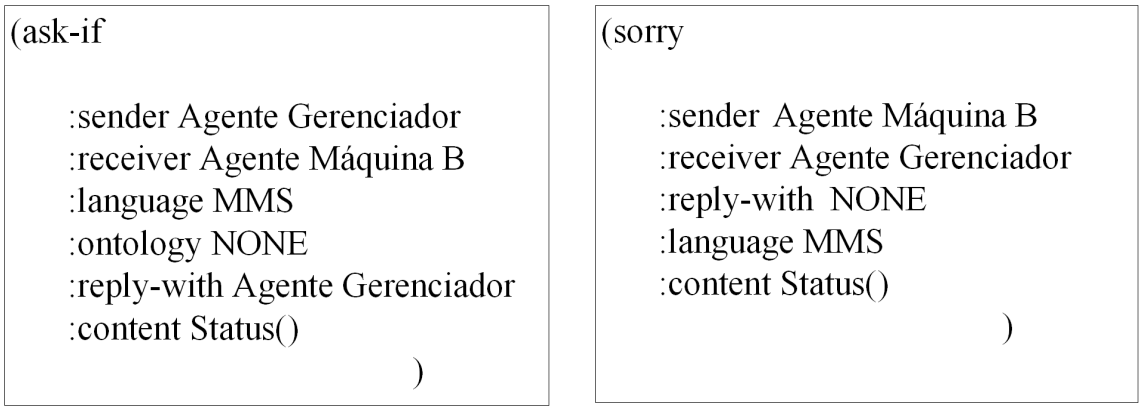
Existem 34 performativas reservadas definidas para a especificação KQML. Uma explicação com detalhes sobre estas performativas pode ser encontrada no Anexo A.

As performativas podem ser classificadas em vários grupos (BENECH *et al.*, 1997):

- Performativas de Discurso: são as performativas usadas no contexto de uma informação e na troca de conhecimento entre dois agentes (*evaluate, ask-if, ask-about, reply, sorry, etc*).
- Performativas de Mecanismos de Conversação e Intervenção: seu papel é ficar entre o curso normal de uma conversação. O curso normal de uma conversação entre agentes é o seguinte: o agente A envia uma mensagem KQML (deste modo iniciando uma conversação) e o agente B responde sempre que ele tiver uma resposta. As performativas desta categoria podem terminar uma conversação prematuramente (*error, sorry*) ou anular este protocolo padrão (*standby, ready, next, rest* e *discard*).
- Performativas de facilidades e Redes: permitem os agentes encontrar outros agentes que possam processar as suas requisições (*register, unregister, forward, broadcast* e *route*).

Portanto, o KQML pode ser considerado como uma linguagem de comunicação para troca de informação e conhecimento entre agentes, através do uso de um conjunto de mensagens padrão.

A Figura 6 a e b ilustra um exemplo de mensagens KQML.



a – Performativa ASK-IF b – Performativa SORRY

Figura 6 – Performativas

No exemplo da Figura 6a, na terminologia KQML, *ask-if* é uma performativa. A performativa configura os parâmetros que são introduzidos por palavras-chave, ou seja, uma performativa identifica o tipo de mensagem. Neste exemplo o agente nomeado "Agente Gerenciador" (*:sender*) interroga o agente "Agente Máquina B" (*:receiver*), em MMS (*:language*), sobre o *status* do "Agente Máquina B" (*:content*). Qualquer resposta para esta mensagem KQML será encaminhada para "Agente Gerenciador" (*:reply-with*). Nenhuma ontologia foi usada para prover informações adicionais relativas a interpretação do campo *:content*.

No caso de o agente "Agente Máquina B" (Figura 6b) ser incapaz de executar a ação requisitada pelo agente "Agente Gerenciador" na sua mensagem, o "Agente Máquina B" irá responder para o "Agente Gerenciador" da seguinte maneira: "Agente Máquina B" (*:sender*) usa a performativa *sorry* para informar ao "Agente Gerenciador" (*:receiver*) que não pode executar a avaliação de *Status()*.

Os projetistas do KQML planejaram as performativas para ter significado independente do conteúdo da linguagem. Assim, a *:language* especificada poderia ser uma cláusula SQL, Prolog, KIF, Lógica de Primeira Ordem ou qualquer outra (WEINSTEIN, 1999).

A seqüência de sintaxes KQML

Uma performativa normalmente é expressa como uma *string* ASCII[21] usando uma sintaxe que foi definida em uma BNF[22], que possui poucas regras e símbolos, facilitando a especificação das mensagens (BNF, 2002; BNF2, 2002; AHO *et al.* 1986). A notação BNF define uma sintaxe para idiomas, sendo tipicamente usada na área da ciência da computação. A notação BNF é uma definição de linguagem construtiva, a qual provê regras (por exemplo um programa de computador ou compilador) onde se pode construir as sentenças válidas de um idioma. Também forma uma maneira humana que torna o idioma legível.

Parâmetros em performativa são indexados por palavras-chave, devem ser seguidos de dois pontos (:) e devem preceder o valor de parâmetro correspondente. Eles são independentes de ordem. A Tabela 3 mostra um resumo das palavras chave de parâmetro reservado e os seus significados.

Com o KQML não é possível fazer com que uma performativa seja enviada de uma vez só para "grupos" de agentes ou todos os agentes e sistemas existentes, é necessário o uso de uma infraestrutura de comunicação para auxiliar nesta tarefa. Também devido ao seu alto grau de generalização, muitos sistemas optam por não utilizar o KQML, pois, para aplicações específicas/de "mundo fechado", os agentes viriam a utilizar apenas um pequeno conjunto de mensagens e não todas as características do KQML. As funcionalidades não utilizadas acarretam em "peso" desnecessário ao protocolo (RABELO, 1997).

Palavra-Chave	Significado
:sender	Remetente atual da performativa
:receiver	Receptor atual da performativa
:reply-with	Campo esperado em uma resposta à mensagem corrente
:in-reply-with	Campo esperado em uma resposta a uma mensagem prévia (o mesmo valor que :reply-with da mensagem anterior)
:language	nome de uma linguagem de representação para o :content
:ontology	nome da ontologia assumida no :content
:content	informação sobre quais as performativas que expressam uma atitude
:from	origem da performativa em :content quando o encaminhamento da mensagem é usado
:to	destino final quando o encaminhamento da mensagem é usado
:force	indica que o :sender nunca irá desmentir o campo :content

Tabela 3 – Campos do KQML

2.1.4 Agentes segundo a FIPA

A FIPA[23] foi criada em 1996 e é uma associação sem fins lucrativos cujo propósito é promover o sucesso de aplicações, serviços e equipamentos que utilizam a tecnologia de agentes (FIPA, 2002). O objetivos da FIPA é construir e disponibilizar especificações para maximizar a interoperabilidade entre sistemas de agentes. A organização FIPA conta com mais de 50 membros e incluem companhias como a IBM, a Sun Mycrosystems, Inc, Motorola, algumas companhias de telecomunicações como a Sonera, Telia AB, British Telecom e também algumas universidades como a Universidade de Helsink e a Universidade de Tecnologia de Tampere.

No padrão FIPA, um agente é uma entidade de software que encapsula seu próprio estado, comportamento, processo de controle de execução e a habilidade de interagir e se comunicar com outras entidades. Esta definição coloca os agentes na mesma família de objetos, funções, e processos, mas também o distingue colocando-o em um nível muito

[21] ASCII – *American Standard Code for Information Exchange*
[22] BNF – *Bakus-Naur Form*
[23] FIPA – *Foundation for Intelligent Physical Agents*

mais alto de abstração. A interação no paradigma de agentes difere do paradigma cliente-servidor pois os agentes podem interagir em níveis iguais, mediando, colaborando e cooperando para atingir os seus objetivos (FIPA, 2002).

A arquitetura da plataforma de agentes FIPA está contida nas especificações normativas da AMS[24] (FIPA, 2002). A plataforma de agentes FIPA provê a infra-estrutura na qual os agentes podem ser desenvolvidos, o que estabelece um modelo de referência lógico para criação, registro, localização, comunicação, migração e retirada dos agentes.

O modelo de referência de gerenciamento do agente é constituído dos seguintes componentes lógicos, segundo FIPA (2002):

- *Agent Identifier* (AID) rotula um agente então ele pode se distinguir dos demais sem risco de ambigüidade no Universo Agente.
- *Directory Facilitator* (DF) provê um serviço de páginas amarelas para os outros agentes. Os agentes podem registrar seus serviços com o DF ou requisitar ao DF que ele encontre serviços oferecidos pelos outros agentes.
- *Agent Management System* (AMS) exerce um controle supervisor de acesso e uso da plataforma de agentes.
- *Message Transport Service* (MTS) é o método de comunicação padrão entre agentes de diferentes plataformas.
- *Internal Platform Message Transport* (IPMT) provê um serviço de expedição de mensagens para agentes numa plataforma particular, a qual deve ser confiável.

Como a descrição sugere, a FIPA é uma organização para especificar padrões para softwares agente. Em particular, o padrão FIPA-ACL[25] tenta identificar os componentes da comunicação e cooperação entre agentes, pela definição de uma linguagem com semântica concisa, formal e com suporte a protocolos de comunicação. De fato, a

[24] AMS – *Agent Management Specification*
[25] FIPA-ACL – *FIPA Agent Communication Language*

principal especificação padrão FIPA (FIPA, 1999) é composta por algumas sub-especificações, dentre elas:

Administração de Agentes – Este documento contém especificações para administração de agente inclusive administração de serviços de agentes, administração de ontologias de agentes e plataforma de transporte de mensagem para agentes. Este documento está principalmente preocupado com definir interfaces de padrão abertas por ter acesso serviços de administração dos agentes. O documento provê uma série de exemplos para ilustrar as funções de administração de agente definidas (FIPA, 2002).

Integração entre agentes – Esta especificação lida com tecnologias possibilitado a integração de serviços providos por softwares não agentes em uma comunidade multiagente. Ela define o relacionamento entre agentes e softwares de quaisquer sistemas (FIPA, 2002). O propósito desta especificação é permitir agentes descrever, agenciar e negociar com qualquer software, e possibilitar que novos serviços de softwares sejam introduzidos dinamicamente na comunidade multiagente. Esta especificação opera no nível de comunicação de agentes e não define qualquer mapeamento para arquiteturas de softwares específicos. Assim os desenvolvedores são capazes de construir pacotes (*wrappers*) para serviços de software os quais são utilizados e/ou controlados por uma comunidade de agentes (FIPA, 2002).

Assistência pessoal para viagem – Este documento estende o padrão de FIPA provendo uma especificação de aplicação para a indústria de turismo. A especificação não está completa, mas os exemplos incluídos ajudam ilustrar o uso de padrão de FIPA e assim acelerar o desenvolvimento e desenvolvimento de reais sistemas. Alguns pontos desta arquitetura foram selecionados como normativo para começar os testes de interoperabilidade. Em resumo, a especificação atende a dois fins: 1 – Continuar testando especificações FIPA técnicas. O contexto de uma aplicação real serve para determinar os pontos fortes e fracos das especificações, 2 – Demonstrar o real valor empresarial e exigência de uma especificação padrão para uma aplicações distribuídas (FIPA, 2002).

Entretenimento e *broadcasting* (difusão) audiovisual – Hoje mais do que nunca, percebe-se a necessidade de meios efetivos para filtrar e recuperar as informações, em

particular, cadeias de radiodifusão digitais. A seleção e armazenamento das preferências pessoais de um espectador de uma série de programações em oferta pode não ser muito prático; tal informação tem que ser provida de uma maneira de customizada, para melhor atender as preferências pessoais do usuário. Para implementar informação que filtra e recupera, o conteúdo semântico e sintático dos fluxos de dados de radiodifusão é necessário um método compatível para permitir a seleção consistente. É crucial que interação humana com tal um sistema seja o mais simples e intuitiva possível (FIPA, 2002). Esta especificação FIPA descreve: 1 – Como filtrar e recuperar a informação; 2 – Customização de informação; 3 – Automatização de casa, educação e entretenimento, e; 4 – Compatibilidade de informação. O centro da especificação FIPA para comunicação entre agentes foi extraído da linguagem ARCOL que foi projetada pela France Télecom (MONTESCO, 2001) e inclui primitivas de comunicação *(performative*s), um modelo formal, e uma linguagem de conteúdo *(Semantic Language*). A linguagem de conteúdo é uma linguagem formal usada para definir a semântica da FIPA-ACL. É quantificada, multimodal com operadores de crença (*belief*), desejo (*desire*) e intenção (*persistent goals*). A especificação FIPA não estabelece nenhuma restrição sobre a(s) tecnologia(s) utilizada(s) na implementação dos agentes.

2.1.4.1 Diferenças e comparações entre FIPA-ACL e KQML

A FIPA-ACL é uma linguagem, como o KQML, baseada em ações de fala. Suas especificações consistem de um conjunto de tipos de mensagem e descrições dos efeitos das mensagens sobre os agentes que a enviam e sobre os que as recebem (FIPA, 1999).

A sintaxe das duas linguagens é idêntica, exceto por alguns nomes diferentes para primitivas. Com isso, foi mantido o conceito de separação do conteúdo que existe em KQML. A linguagem externa (correspondente à parte de mensagem de KQML) define o significado desejado da mensagem e a interna, uma expressão das crenças, desejos e intenções do interlocutor (MONTESCO, 2001).

As linguagens de comunicação entre agentes FIPA-ACL e KQML diferem nas suas respectivas apresentações semânticas. Por causa dessa diferença, seria impossível obter um mapeamento exato entre as performativas do KQML e das primitivas da FIPA-ACL.

Uma diferença entre as duas linguagens de comunicação entre agentes é seu tratamento de primitivas de anúncio e definição de capacidades e de registro. Os agentes precisam desse tipo de primitivas para comunicar-se com os agentes intermediários. Em KQML, tarefas, tais como registro, atualização e anúncio de capacidades são chamadas primitivas de *facilitação*. FIPA ACL não provê, ainda, primitivas de facilitação.

KQML tem sido criticada por usar o termo "*performative*" para se referir às primitivas de comunicação. Em FIPA-ACL, as primitivas de comunicação são chamadas de *ações comunicativas*. Apesar dos nomes diferentes, as performativas e ações comunicativas são o mesmo tipo de entidade (MONTESCO, 2001).

O documento de especificação da FIPA-ACL, assim como KQML, não faz nenhum compromisso com uma linguagem conteúdo em particular. Isto é verdadeiro para a maioria das primitivas. Porém, agentes destinatários devem entender a Linguagem Semântica (MONTESCO, 2001).

A Tabela 4 mostra uma comparação entre as principais características das linguagens de comunicação KMQL e FIPA-ACL.

Elementos/ Linguagem	Formato da Mensagem	Conteúdo Sintático e Linguagem	Conteúdo Semântico
KQML	Estrutura em camadas; separação entre conteúdo, mensagem e comunicação	Knowledge Interchange Format (KIF) Linguagem de Conteúdo não definida	Semântica Informal
FIPA-ACL	Estrutura em camadas; separação entre conteúdo, mensagem e comunicação	Linguagem *Semantic Language*	Semântica Formal baseada em Lógica

Tabela 4 – Comparação entre KQML e FIPA-ACL

2.1.5 MMS

O MMS (*Manufacturing Message Specification*) (SISCO, 1995), (SISCO, 1996), (SISCO, 1998a) corresponde ao protocolo ISO 9506 e é um padrão internacional para troca de mensagens e dados em tempo real e controle supervisório entre dispositivos e aplicações de computadores em rede. Ele atua de uma maneira independente da função da aplicação que está sendo executada, independente do dispositivo de desenvolvimento e da aplicação. O MMS dota as aplicações com um nível de "abstração" onde não é necessário conhecer as particularidades dos diferentes controladores existentes, em outras palavras, as aplicações podem ver os controladores como provedores de serviços homogêneos. A partir desta perspectiva, o MMS pode ser considerado o protocolo de aplicação de padrão mais importante para o contexto industrial (RABELO, 1997, 1998).

A filosofia do MMS trabalha com a noção cliente-servidor. Ou seja, quando um "cliente" deseja algo de um servidor, ele envia uma *mensagem* a este que, após realizar a operação requerida, responde, eventualmente, ao cliente (RABELO, 1997).

Obter informações precisas de dispositivos de sistemas de controle em tempo útil é um dos aspectos mais difíceis na construção de sistemas integrados (SISCO, 1996). Muitas marcas de computadores, dispositivos e controladores e a multiplicidade de mecanismos de comunicação usados por esses sistemas podem formar uma grande barreira para integração. O protocolo MMS é uma das alternativas existentes no mercado para romper essas barreiras e auxiliar na integração industrial.

Uma implementação de um servidor MMS deve prover o mapeamento entre o modelo abstrato das funcionalidades do dispositivo industrial e suas funcionalidades reais. A Figura 7 ilustra esta implementação:

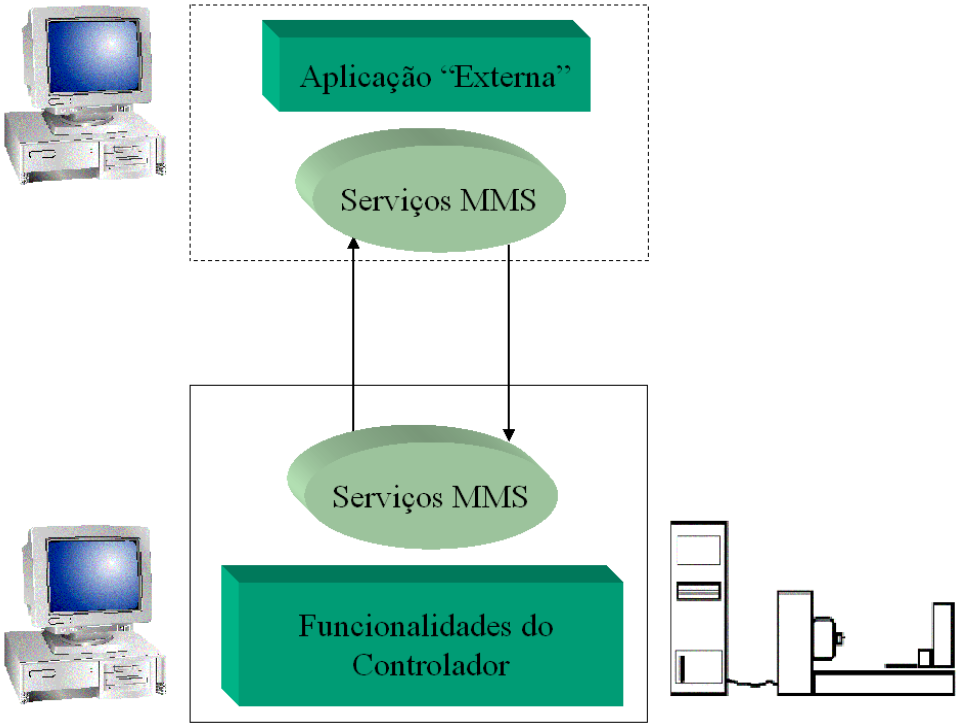


Figura 7 – Implementação MMS

O padrão MMS é dividido em duas partes principais. A parte 1 é a especificação do serviço. Esta especificação contém uma definição do Dispositivo Virtual de Manufatura (VMD[26]), os serviços (mensagens) trocados entre os nós da rede, os atributos, e os parâmetros associados com o VMD. A parte 2 é a especificação do protocolo. Esta especificação define as regras de comunicação, que incluem o sequenciamento das mensagens através da rede, o formato (codificação) das mensagens e as interações da camada MMS com outras camadas do modelo OSI (SISCO, 1998a).

Benefícios do MMS

O MMS proporciona benefícios, baixando o custo de construir e usar sistemas automatizados (SISCO, 1998a). O MMS é apropriado para qualquer aplicação que requeira um mecanismo de comunicação comum para executar tarefas relacionadas com acessos à tempo real, distribuição de processos e controle supervisório de dados. Os

[26] VMD – *Virtual Manufacturing Device*

serviços de mensagens providos pelo MMS são genéricos o suficiente e se tornam apropriados para uma vasta gama de dispositivos, aplicações e indústrias (SISCO, 1998a), (RABELO, 1997).

Aqui se destacam as três maiores vantagens do uso do MMS na indústria:

- Interoperabilidade – permite que duas ou mais aplicações em rede possam trocar informações de controle supervisório sem a necessidade do usuário da aplicação criar um ambiente particular de comunicação.
- Independência – a interoperabilidade pode ser alcançada independentemente do:
  - *Desenvolvedor da Aplicação*. Os módulos de comunicação são usualmente específicos a uma marca particular (ou mesmo modelos) de aplicações ou de dispositivos. O MMS é definido por padrões internacionais independentes e com a participação dos principais especialistas e fabricantes.
  - *Conectividade em Rede* – MMS se torna à interface para a rede de aplicações, isolando assim, a aplicação da maioria dos aspectos que não pertencem ao MMS e de como as mensagens são transferidas pela rede de um nó ao outro.
  - *Execução de Funções* - MMS provê um ambiente de comunicação independente da função a ser executada.
- Acesso a Dados – permite que as aplicações conectadas em rede obterem as informações requeridas por uma aplicação.

O MMS também provê definições, estruturas e significado das mensagens, o que permite a duas aplicações desenvolvidas independentemente operarem juntas. O MMS tem um conjunto de características que facilitam a distribuição em tempo real de dados e funções de controle supervisório através da rede em um ambiente cliente/servidor.

O padrão MMS também provê definições para:

- Objetos – O MMS define um conjunto de objetos comuns (ex.: variáveis) e define os atributos de rede visíveis para aqueles objetos (ex.: nome, valor, tipo).

- Serviços – O MMS define um conjunto de serviços de comunicação (ex.: ler, escrever) para acessar e gerenciar estes objetos em um ambiente de rede.
- Comportamento – O MMS define um comportamento visível da rede que um dispositivo pode exibir quando está processando este serviço.

Objetos MMS

O MMS define classes contendo um determinado número de objetos que, por sua vez, são caracterizadas através de atributos (SILVEIRA, 1991). Esses atributos descrevem as características visíveis externamente dos objetos de uma determinada classe.

- **O Objeto VMD**

A característica chave do MMS é o modelo de Dispositivo Virtual de Manufatura (VMD). O modelo VMD especifica como os dispositivos MMS, também chamados de servidores, se comportam através do ponto de vista de uma aplicação externa (MMS ou não) (SISCO, 1998a). O MMS permite qualquer aplicação ou dispositivo prover ambas as funções de cliente e servidor ao mesmo tempo.

O VMD pode ser visto como um objeto para o qual todos os outros objetos MMS são subordinados (variáveis, domínios e outros estão contidos dentro do VMD). Em outras palavras, o VMD tem como objetivo proporcionar ao cliente acessar objetos do servidor por meio de um pedido de serviços MMS. Desta forma, é necessário que o servidor torne disponível os recursos e funcionalidades ligados ao serviço MMS.

O modelo VMD apenas especifica os aspectos visíveis da rede para as comunicações (SISCO, 1995). Os detalhes internos de como um dispositivo real implementa um modelo VMD não são especificados pelo MMS. Focando apenas nos aspectos visíveis de um dispositivo, o modelo VMD é especificado para prover um alto grau de interoperabilidade.

Em geral, o modelo VMD define os objetos contidos no servidor, os serviços que um cliente pode usar para acessar e manipular estes objetos e o comportamento do

servidor nos serviços requisitados pelos clientes (LEITÃO *et al.*, 1996), como ilustra a Figura 8.

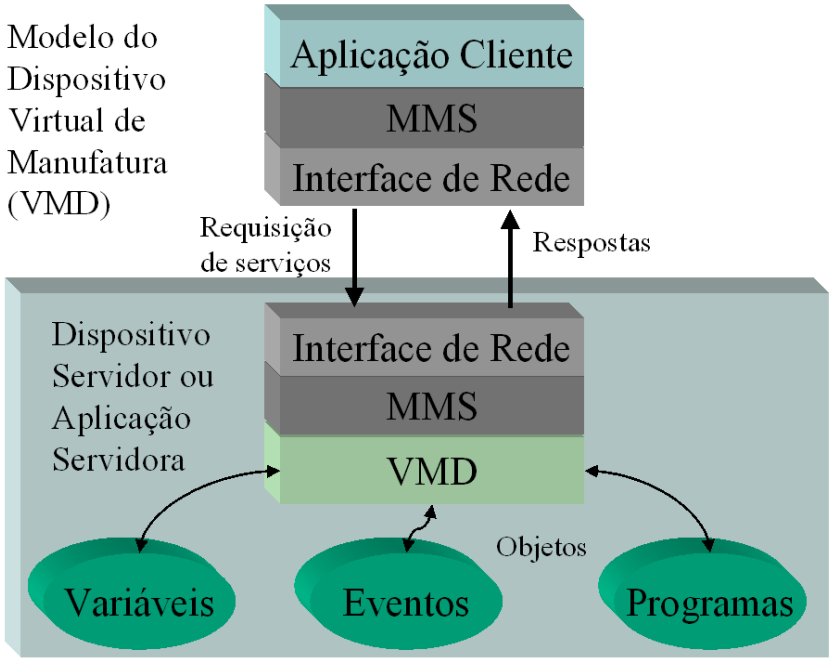


Figura 8 - VMD

O modelo VMD proporciona uma visão consistente e bem definida para as aplicações cliente dos objetos contidos no VMD. Clientes usam serviços MMS para acessar e manipular estes objetos. O MMS requer que todos os serviços se comportem de acordo com o modelo VMD.

- **Objetos de Semáforo**

Semáforos MMS são objetos que podem ser usados para controlar o acesso a outros recursos e objetos dentro do VMD. Por exemplo, um VMD que controla o acesso para um *setpoint* (uma variável) para o controle de *loop* poderia usar semáforos somente para permitir que apenas um cliente por vez será capaz de mudar o *setpoint*.

- **Objetos de Controle de Programas (Domínios e Invocação de Programas)**

O modelo de execução VMD define dois objetos para controlar a execução de programas dentro do VMD. Um domínio MMS é definido como um objeto que

representa um recurso dentro do VMD (ex.: a memória na qual o programa está locado/armazenado). Uma invocação de programa é definida como um grupo de domínios e a sua execução pode ser controlada e monitorada.

- **Variáveis MMS**

Uma variável é um elemento de um tipo de dado que está contido dentro de um VMD. Uma variável MMS é um objeto virtual que representa um mecanismo para os clientes MMS acessarem uma variável real.

O MMS descreve dois tipos de objetos virtuais para descrever variáveis de acesso (SISCO, 1998a):

Objetos de variáveis não nomeadas – Um objeto de variável não nomeada descreve o acesso à variável real pelo uso de um dispositivo de endereço específico. O MMS inclui objetos de variáveis não nomeadas principalmente para compatibilidade com serviços legados que não são capazes de suportar nomes. Uma variável não nomeada é um mapeamento direto para variável real subjacente que está localizada um endereço específico. Os atributos de variáveis não nomeadas são:

- "*Address*": este é o atributo chave das variáveis não nomeadas. Define onde a variável está localizada.
- "*MMS Deletable*": este atributo é sempre falso para um objeto de uma variável não nomeada. Objetos de variáveis não nomeadas não podem ser excluídos porque eles são uma representação direta para a variável "real" localizada em um endereço específico no VMD.
- "*Type Description*": este atributo descreve o tipo (formato e valores possíveis) da variável.

Objetos de variáveis nomeadas – Os objetos de variáveis nomeadas descrevem os acessos para a variável real usando um objeto de nome MMS. Clientes MMS precisam apenas saber o nome do objeto para o acessar. Um objeto de variável nomeada tem os seguintes atributos:

- "*MMS Deletable*": este atributo indica se o acesso a uma variável pode se excluído.
- "*Type Description*": este atributo descreve o tipo (formato e valores possíveis) da variável.
- "*Access Method*": se o método de acesso é público, isto significa que o endereço subjacente do objeto de variável nomeada é visível para o cliente MMS. Neste caso, a mesma variável pode ser acessada como um objeto de variável não nomeada.

Gerenciamento de Contexto MMS

O MMS provê serviços para gerenciamento de contexto das comunicações entre dois nós MMS na rede. Estes serviços são usados para estabelecer e finalizar aplicações associadas.

O protocolo MMS é uma solução para aplicações que requerem um controle dos recursos, ou quando a integração envolve um número de recursos com complexibilidade e heterogeneidade.

Desvantagens ao empregar o MMS

Apesar de todos os esforços em busca de uma padronização no que diz respeito a comunicação em ambientes fabris, a maioria dos dispositivos programáveis de manufatura instalados ainda implementam padrões incompatíveis. Algumas razões são apontadas como determinantes da situação atual. Primeiro, a implementação de algumas características básicas da especificação MMS baseadas no modelo OSI tem sido muito lenta e até esquecida. Isto gera uma certa hesitação por parte do pessoal envolvido neste processo em termos de investir recursos financeiros em uma especificação incompleta. Segundo, a tecnologia tem um custo elevado. Apenas as grandes organizações têm condições de adquiri-la. Outra consideração que deve ser feita diz respeito à falta de interesse que as empresas que trabalham nesta área (vendendo soluções prontas ou proprietárias) têm em abrir os seus sistemas temendo a perda de mercado (SISCO, 1998a).

2.1.6 CORBA

CORBA (*Common Object Request Architecture*) (CHAFFEE, 1999), (KEAHEY, 2001), (SEIGEL, 2000) é uma arquitetura padrão para sistemas com objetos distribuídos proposto pela OMG[27]. A especificação CORBA permite que uma coleção heterogênea e distribuída de objetos interoperem (SIEGEL, 2000), (CHAFFEE, 1999). Isto significa dizer que a arquitetura CORBA permite aplicações se comunicarem com outras sem se preocupar onde eles estão localizados ou como essas aplicações são projetadas, como em um barramento global.

A arquitetura CORBA é uma plataforma independente que permite a interoperabilidade com outras aplicações e outras plataformas de forma transparente. Via um objeto chamado ORB os objetos das aplicações podem ser encontrados "automaticamente" e os respectivos métodos acessados onde quer que eles se encontrem. O protocolo entre ORBs genérico (GIOP[28]) especifica uma sintaxe de transferência genérica e um conjunto de formatos de mensagens para comunicação entre ORBs. O protocolo entre ORBs para Internet (IIOP[29]) especifica como mensagens GIOP são trocadas usando conexões TCP/IP. Ele especifica um protocolo de interoperabilidade padrão para *Internet*.

IDL

As interfaces IDL são uma linguagem declarativa. Elas definem "ligações entre linguagens" para muitas linguagens de programação diferentes. Isto permite ao implementador de um objeto escolher a linguagem de programação apropriada para o objeto. Similarmente, isto permite ao desenvolvedor do cliente escolher a linguagem de programação apropriada, e possivelmente diferente, para o lado cliente.

Usando-se uma IDL são descritas características de interface, sem se preocupar com a linguagem de programação, para obter:

---

[27] OMG – *Object Management Group* – http://www.omg.org
[28] GIOP – *Generic Inter-ORB Protocol*
[29] IIOP – *Internet Inter-ORB Protocol*

- Interfaces de objetos modulares;
- Operações e atributos que um objeto suporta;
- Exceções criadas por uma operação e;
- Os tipos de dados que uma operação retorna, seus parâmetros e os atributos dos seus objetos.

Uma interface IDL descreve os tipos de dados, serviços e objetos que um cliente pode requisitar e que o servidor pode disponibilizar para a implementação de um dado objeto.

Como exemplo, um arquivo IDL poderia descrever uma interface VMD que define os serviços *CreateProgramInvocation, DeleteProgramInvocation, Start, DataExchange* e *Status* (que são serviços previstos no protocolo MMS). Uma aplicação cliente que fizesse uso desta IDL poderia fazer chamadas para todas as operações disponibilizadas e especificadas na IDL, enquanto que um servidor que poderia satisfazer as requisições de um cliente deveria estar capaz de executar o trabalho associado com as operações descritas na IDL. A Figura 9 mostra um exemplo de definição de uma interface IDL escrita na sintaxe IDL.

```
interface VMD {
      void CreateProgramInvocation();
      void DeleteProgramInvocation();
      void Start();
      void DataExchange(in long pos);
      string Status();                  };
```

Figura 9 – Exemplo de IDL

A Figura 10 ilustra duas operações no lado cliente e seus métodos relacionados no lado servidor.

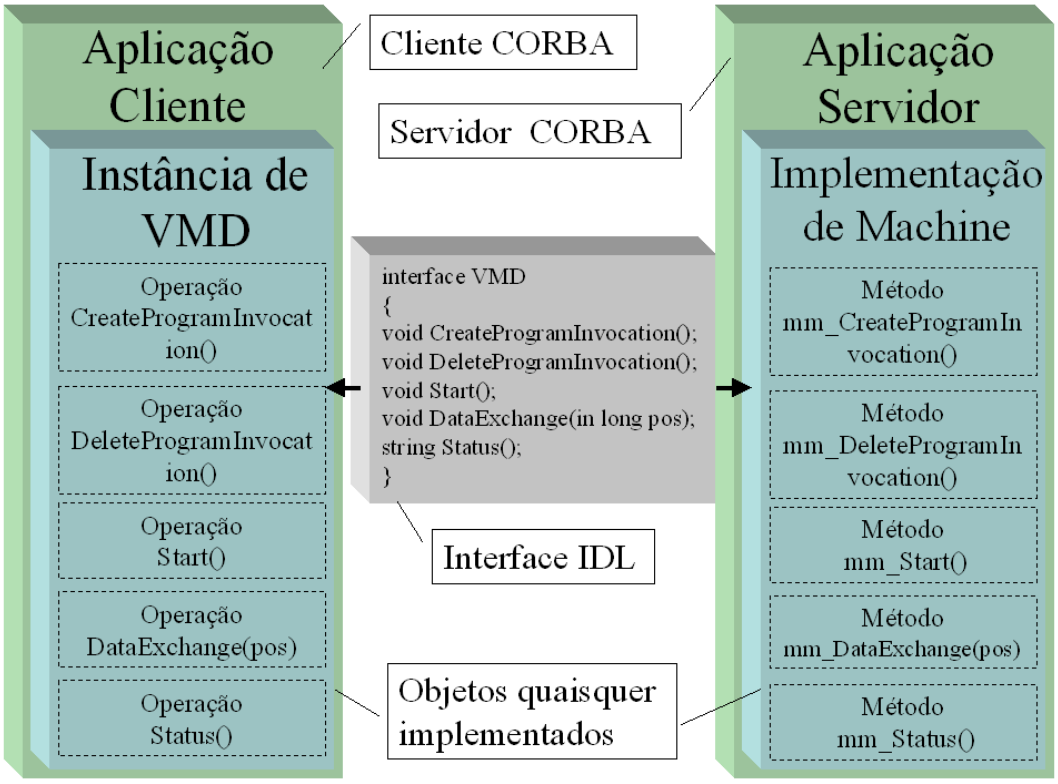


Figura 10 – Iteração Cliente/Servidor via CORBA/IDL

- A operação *CreateProgramInvocation()* e o seu método relacionado mm_*CreateProgramInvocation()* tem como objetivo iniciar o ambiente MMS para as chamadas recebidas através do VMD.
- A operação *DeleteProgramInvocation()* e o seu método relacionado *mm_DeleteProgramInvocation()* tem como objetivo finalizar o ambiente MMS para as chamadas recebidas através do VMD.
- A operação *Start()* e o seu método relacionado *mm_Start()* tem como objetivo deixar o dispositivo industrial pronto para iniciar uma operação qualquer que esteja disponibilizada no VMD.
- A operação *DataExchange(in long pos)* e o seu método relacionado *mm_DataExchange(pos)* tem como objetivo mudar a posição de um dispositivo industrial, por exemplo uma mesa posicionadora, através do parâmetro *pos*.
- A operação *Status()* e o seu método relacionado *mm_Status()* obtém o status físico do dispositivo industrial.

O ORB

Um dos mais importantes componentes CORBA é o ORB – *Object Request Broker*. O ORB é o serviço distribuído que implementa as requisições dos objetos remotos. Ele é o mecanismo responsável por interceptar as chamadas e por encontrar os objetos que podem implementar as requisições, passar os parâmetros, invocar os seus métodos e retornar os resultados implementando a transparência de localização (CHAFFEE, 1999), (PIRES, 1997). O ORB é um *middleware* que estabelece a relação cliente-servidor entre objetos. O cliente não precisa se preocupar onde o objeto está localizado, a sua linguagem de programação, o seu sistema operacional ou qualquer outro aspecto do sistema que não faça parte da interface do objeto. Por exemplo, na Figura 11 o objeto de referência é descrito pela interface *GetMachineStatus*. O ORB entrega a requisição para um objeto e retorna os resultados para o cliente. A Figura 12 ilustra uma requisição. Na figura, a requisição *GetStatus* retorna um objeto de referência descrito pela interface *Status*.

```
Interface GetMachineStatus

{…
  Status GetStatus(in long is_status);
 …}
```

Figura 11 – Interface de uma IDL

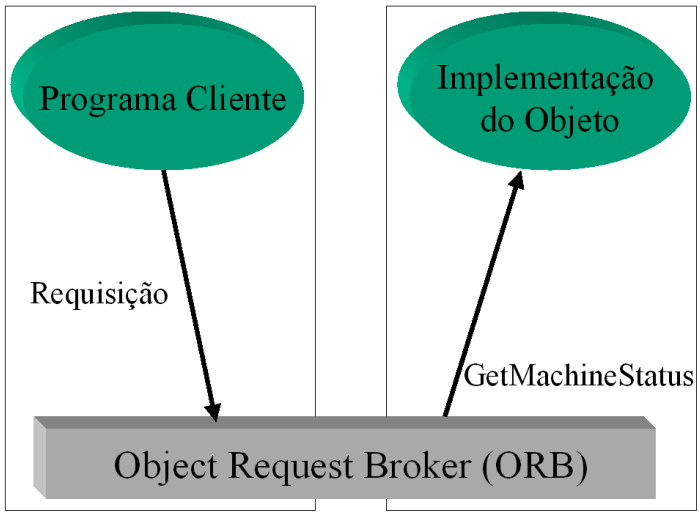


Figura 12 – Troca de Mensagens via CORBA

Em aplicações Cliente/Servidor típicas, os desenvolvedores utilizam um padrão qualquer para definir como será a troca de dados. A definição de protocolos/padrões para troca de dados depende das linguagens de implementação e mecanismos de transporte de rede. O ORB simplifica este processo através do uso de interfaces através da implementação de uma IDL (KEAHEY, 2001). Isto permite aos programadores escolherem o sistema operacional mais apropriado, o ambiente de execução e mesmo a linguagem de programação a ser usada em cada componente em construção. Mais importante ainda, permite a integração de componentes heterogêneos existentes. Em uma solução baseada em ORB os desenvolvedores simplesmente modelam os componentes legados usando as IDLs; desta forma os códigos são encapsulados e são traduzidos entre as interfaces legadas e disponibilizadas de maneira comum.

Com CORBA os usuários ganham o acesso às informações de forma transparente sem precisar saber em que plataforma de *software* ou *hardware* elas, as informações, residem, ou se estão localizadas na rede da empresa ou não.

Dificuldades no emprego do CORBA

A arquitetura CORBA é uma tecnologia que envolve conceitos avançados de programação orientada a objetos e exige experiência no desenvolvimento de aplicações distribuídas (FILHO, 2000).

Muitos dos produtos comerciais existentes provêm apenas as funcionalidades básicas do ORB e, em alguns casos, serviços como o de nomes e o de eventos.

2.1.7 XML

XML (*Extensible Markup Language*) (MORRISON, 2000) (PIMENTEL *et al.*, 2000) é uma linguagem derivada do SGML[30], que por sua vez é a padronização da

[30] SGML – *Standard Generalized Markup Language*

linguagem GML[31]. A GML foi criada nos anos 60 pela IBM com o propósito de estruturar documentos de forma padronizada para facilitar a troca e a manipulação de dados internos, publicações, relatórios e outros tipos de documentos.

Em 1986 o SGML tornou-se um padrão ISO. Embora extremamente poderoso, o SGML era muito complexo e pesado. Devido a isso o SGML não foi adotado para representação de hipertexto no início da Internet. Em 1989 surge o HTML, baseado no SGML, que acabou se tornando o padrão para o formato de informação na Web. Em 1998 o W3C[32] (W3C, 2000) cria o XML, como uma forma de introduzir o poder e a flexibilidade do SGML nos domínios *Web.* Isso visava fornecer três benefícios significantes que estavam faltando no HTML: extensibilidade, estrutura e validação dos dados. De fato, o XML não é nada mais que um formato de texto padronizado para representar informações estruturadas na *Web*, não sendo, portanto, uma versão estendida do HTML. Enquanto o HTML é uma linguagem de marcadores específica que é usada para codificar as informações para apresentação em *Web browsers,* o XML é uma especificação para projetar linguagem de marcações, ou seja, XML é uma *metalinguagem*.

O XML está se estabelecendo como uma proposta para representar as informações e a troca de dados entre sistemas, não apenas no domínio da Internet, mas também como ferramenta de suporte para a interoperabilidade entre aplicações heterogêneas entre as empresas (OSÓRIO *et al.* 2000).

A construção básica de um documento XML é a entidade, que contém dados analisados ou não gramaticalmente (dados "parseados" ou "não parseados"). Dados "parseados" consistem de caracteres que são considerados dados de caracteres ou marcadores que são processados por um processador XML. Dados "não parseados" são vistos como um texto "cru" que não é processado.

[31] GML – *Generalized Markup Language*
[32] W3C – *World Wide Web Consortium*

Os marcadores são usados para descrever a estrutura de armazenamento de um documento (entidades) e estruturas lógicas (elementos). O XML permite a imposição de limitações no formato e na estrutura de um documento, especificando os relacionamentos entre os componentes marcadores.

As aplicações XML empregam os processadores XML para ter acesso ao conteúdo e à estrutura contida nos documentos XML. Para os usuários finais, uma aplicação XML e um processador XML provavelmente são indistinguíveis. Contudo, a distinção é muito importante para os desenvolvedores de aplicações XML porque os processadores XML estão prontamente disponíveis como componentes de software que podem ser utilizados pelas aplicações para fazer o trabalho de processamento do XML – verificar se os documentos XML são bem formados e válidos. Um documento bem-formado é um documento que segue as regras de sintaxe do XML, enquanto que um documento válido é um documento bem-formado que também segue a um DTD. É importante lembrar que o XML *parser* é um componente do processador XML que analisa as marcações XML e determina a estrutura dos dados do documento.

Os documentos XML são textos formados por caracteres e dados de informações de marcação explicitamente separados. Informações de marcação incluem comentários, referências a caracteres e entidades, delimitadores de seção CDATA, elementos, instruções de processamento e DTDs. Todo o resto constitui caracteres de dado. Quando um documento é processado, algumas marcações são transformadas em caracteres de dados.

Como já mencionado, documentos XML consistem de dados de caracteres e marcadores. São os seguintes diferentes componentes de marcadores em XML.

Elementos de marcadores (*tags*) – Os *tags* são os componentes mais comuns na sintaxe XML e são usados para descrever elementos. Um elemento é caracterizado por qualquer cadeia que aparece entre os caracteres delimitadores < e >, desde que não esteja contido em um comentário ou em uma seção *CDATA*. O *tag* inicial <qualquer elemento> marca o início de um elemento e o *tag* final </qualquer elemento> marca o fim do elemento. A Figura 13 ilustra uma mensagem XML. Os campos <*name_ent*> e

</name_ent> são exemplos de marcadores, *Software & Software LTDA* é um *dado* (não "parseada"). É interessante lembrar que um elemento é um segmento de uma marcação, enquanto que um *tag* se refere a um texto específico usado para representar um elemento em um documento XML.

```
<!-- Doumento XML que identifica uma empresa-->

<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "message.dtd">

<message>
        <code_enterprise>234</code_enterprise>
        <name_ent>Software &amp;amp; Software LTDA</name_ent>
        <complemento/>
        <contact>
                <contact_name>João</contact_name>
                <contact_fax>5455665</contact_fax>
                <email>joao@softsoft.com</email>
                <function>sales</function>
                <department>comertial</department>
        </contact>
</message>
```

Figura 13 – Exemplo de Documento XML

Diferentemente do HTML, o XML exige que a todo elemento de abertura seja associado um elemento de fechamento, mesmo que nenhum dado esteja delimitado por tais elementos. Uma alternativa de representação é a utilização do elemento vazio, o qual assume ambas as funções, de elemento de abertura e de fechamento. O elemento vazio deve terminar com /> em vez de apenas >, por exemplo o campo <complemento/> da Figura 13.

Referências a entidades – São marcações que são substituídas com caracteres de dados após a análise do documento. As Referências as entidades são blocos de construções dos documentos XML. Essencialmente, uma entidade se refere a um único

nome de um segmento de dados do XML. As Referências as entidades são as chamadas a estas entidades e são feitas através do caracter "&". Toda vez que um documento XML quiser se referenciar e utilizar o caractere "&", por exemplo, ele pode fazer a referência a esta entidade como ilustrado na Figura 13 no campo <name_ent>Software *&* Software LTDA</name_ent>

Comentários – Os comentários são usados em um documento XML para apresentar informações que não fazem tecnicamente parte do conteúdo do documento. Como nas linguagens de programação, comentários em XML são usados para proporcionar descrições de dados de documentos para o usuário. Os *parsers* e as aplicações XML geralmente ignoram os comentários. Os comentários iniciam com <!-- e terminam com -->. O único caracter que não é permitido nos comentários é o caracter "-", pois gera conflito com os caracteres da sintaxe de comentários, como ilustra Figura 13 <!- - Documento XML que identifica uma empresa-->

Instruções de processamento – A sintaxe XML não lida apenas com dados de caracteres e marcadores. Existem também instruções de processamento que atuam como um mecanismo de inserção de informações explícitas em um documento destinadas a alguma aplicação. Os *parsers* XML não interpretam tais informações, assim como não o fazem para comentários; eles simplesmente as repassam para a aplicação. Sintaticamente uma instrução de processamento é uma cadeia de caracteres que começa com a configuração <? e termina com ?>. A cadeia de caracteres deve começar com um nome XML seguido por espaço e por dados. Exemplo: <?xml version="1.0"?> Figura 13.

CDATA – Normalmente o texto que aparece entre os delimitadores < e > são considerados marcações. Uma exceção é feita aos textos entre delimitadores de seção *CDATA*, que são considerados caracteres de dados. Os delimitadores de abertura e fechamento da seção são, respectivamente, *<![CDATA[e]]>*. A única seqüência de caracteres que não pode aparecer em uma seção *CDATA* é *]]>*. As seções *CDATA* são úteis quando se deseja que todos os caracteres de um texto sejam interpretados como caracteres e não como elementos de marcação. Exemplos são textos contendo os caracteres <, >, &, etc., comuns em trechos de código de programas ou em textos

descrevendo XML. A seguir segue um trecho de programa protegido por uma seção *CDATA*. Exemplo: <![CDATA[if A > B then MAIOR = A else MAIOR = B;]]>

Declarações do tipo de documentos

As declarações do tipo de documentos são usadas em XML para especificar informações sobre um documento, incluindo o elemento raiz e o a Definição do Tipo de Documento (DTD).

A declaração do tipo de documento para um documento é muito importante para estabelecer se o documento é válido ou apenas bem-formado. As três principais tarefas da declaração do tipo de documento são:

- Especificar o elemento raiz do documento;
- Definir elementos, atributos e entidades específicas para o documento (interno ao DTD).
- Identificar um DTD externo para um documento.

Na Figura 13 é ilustrada uma declaração do tipo de documento usado <!DOCTYPE message SYSTEM “message.dtd”>

O elemento raiz para o documento é o elemento *message*, que é claramente especificado na declaração do tipo de documento. O DTD externo para o documento *message.dtd* é também claramente referenciado na declaração do tipo de documento.

DTD - Document Type Definition

O DTD é um conjunto de regras que define quais tipos de dados e entidades farão parte de um documento XML. Estas regras são utilizadas para que o *parser* possa verificar se o documento foi gerado de forma correta ou se apresenta erro.

Documentos DTD permitem representar uma nova dimensão de um documento. Ao restringir os elementos e atributos que podem ser utilizados na caracterização do documento, e também ao definir a ordem, encadeamento, aninhamento de elementos, entre outras regras, que devem ser respeitadas, DTDs permitem melhor estabelecer a estrutura de documentos válidos e viabilizam a construção de aplicações que exploram tais disciplinas. As aplicações podem associar significados a trechos dos documentos, o que lhes permitem processá-los automaticamente.

Um DTD pode estar definido dentro do próprio arquivo XML ou em um arquivo à parte, com extensão ".dtd", que deve ser mencionado no código XML. A utilização do DTD pode padronizar um documento XML estabelecendo padrões de marcação para as mais diversas áreas, possibilitando que todas as empresas de um mesmo ramo de atividade (ou até mesmo de um outro) possam se comunicar de maneira unificada, criando vários tipos de linguagem de marcação a partir da linguagem XML.

Os DTDs são caracterizados pela definição de elementos <!ELEMENT nome_elemento > e as suas listas de atributos <!ATTLIST nome_elemento nome_atributo TIPO_ATRIBUTO QUALIFICADOR> . Na definição dos elementos, estes ainda podem conter uma lista de elementos, por exemplo <!ELEMENT nome_element (elem_1+, elem_2?)>, que são aninhados àquele elemento.

- **DTD Interno**

O DTD interno, apesar de não ser a forma mais utilizada, pode ser útil quando surgir a necessidade de se estabelecer regras para um único documento XML sem que seja necessário utilizar as mesmas regras para elaborar outros documentos. A declaração do tipo de documento interno pode definir todas as regras ou parte delas, podendo também atuar em conjunto com um outro DTD, sendo este "externo".

O DTD interno é mais maleável do que o DTD externo, pois mesmo que um elemento seja declarado, ele pode ser utilizado no documento, fato que não ocorre em um DTD externo, cujas regras de construção são severamente analisadas. A seguir segue a

construção de um DTD interno por meio da declaração de elementos que o documento poderá conter:

```
<? Xml version="1.0"?>
<! DOCTYPE TEXTO [
<! ELEMENT TEXTO ANY>
<! ELEMENT FRASE (#PCDATA)>
]>
<TEXTO>
      <FRASE> Primeiro exemplo de DTD interno</FRASE>
</TEXTO>
```

Tabela 5 – Exemplo de DTD Interno

A parte do documento referente ao DTD interno aparece em negrito e está delimitada pelos caracteres:

<!DOCTYPE, TEXTO [, o formato aplicado é: <!DOCTYPE- nome do elemento – *root* [e deve ser sempre seguido de uma declaração interna. No caso, o elemento *root* utilizado no documento, deve possuir obrigatoriamente o nome TEXTO (em maiúsculas), já os outros nomes não são escritos.

A linha <!ELEMENT TEXTO ANY> , está indicando que o elemento (ELEMENT) TEXTO, isto é, a *tag* TEXTO que em nosso caso é o próprio elemento *roo*t, poderá conter qualquer nome (ANY), isto é, quaisquer elementos filhos, sejam eles outras *tag*s, entidades ou atributos. Sempre que necessário realizar a declaração de uma *tag* no DTD, a palavra reservada "ELEMENT", deve ser utilizada,

A linha <!ELEMENT FRASE (#PCDATA)>, indica que a *tag* FRASE ao aparecer no documento XML poderá conter um texto (#PCDATA), entretanto, ela não deverá conter outros elementos, apenas texto.

- **DTD Externo**

Caso o DTD esteja armazenado em um arquivo, ele é considerado externo ao documento XML. Neste caso deve-se usar a seguinte sintaxe:

<!DOCTYPE elemento-raiz SYSTEM "nome de arquivo” >

```
<? Xml version="1.0"?>
```

```
<DOCTYPE nota SYSTEM "nota.dtd">
<nota>
<para>Joao</para>
<de>Jose</de>
<cabecalho>lembrete</cabecalho>
   <corpo> Nossa reuniao esta marcada para as 3 horas</corpo>
   </nota>
```

Tabela 6 – Exemplo de DTD Externo

Neste caso o Arquivo "nota.dtd" conteria:

```
<!ELEMENT nota ( de, para, cabecalho, corpo )>
<!ELEMENT para ( #PCDATA )
<!ELEMENT de ( #PCDATA )
<!ELEMENT cabecalho( #PCDATA )
<!ELEMENT corpo( #PCDATA )
```

XML Schema

O *XML Schema Definition Language* provê um super conjunto dos recursos disponibilizados por DTDs. Utiliza-se o termo *instância de documento* como referência a um documento XML que está em conformidade com um *Schema XML* particular.

Uma instância de um documento consiste do elemento principal o qual possui sub-elementos os quais, por sua vez, contêm outros sub-elementos. Assim, os elementos contêm hierarquicamente outros elementos até um ponto em que os elementos no nível mais baixo dessa hierarquia contêm datas, números ou outras formas de cadeias de caracteres. No caso da definição com DTDs, os elementos nesse nível mais baixo são definidos com #PCDATA.

No caso de XML Schema, elementos que contêm outros elementos, ou que têm atributos, são ditos terem “tipos complexos”, enquanto que elementos contendo apenas cadeias de caracteres são ditos conterem “tipos simples”. Um XML Schema consiste de um elemento e uma variedade de sub-elementos, em particular element, complexType e simpleType, que permitem especificar com mais rigor o conteúdo possível nas instâncias dos documentos correspondentes. A diferença básica entre tipos simples e tipos complexos é que apenas estes últimos podem conter elementos e podem ter atributos associados. Outra distinção importante é entre definições que criam tipos novos e aquelas

que permitem o uso de instâncias de elementos ou atributos com nomes e tipos específicos. Em ambos os casos os tipos podem ser simples ou complexos.

Tipos complexos novos são definidos através do elemento xsd:complexType. Tais definições normalmente contêm um conjunto de declarações, referências a elementos e declarações de atributos. As declarações não são tipos em si e sim associações entre o nome correspondente e as restrições impostas pelo esquema. Elementos e atributos são declarados com os elementos xsd:element e xsd:attribute respectivamente.

Ao invés de declarar um elemento pela associação de um novo nome à definição de um tipo existente, é possível utilizar um elemento já existente, como no caso da declaração, que faz referência a um elemento declarado em algum outro ponto do esquema. Regra geral, o nome de um atributo precisa referenciar um elemento global declarado no primeiro nível do esquema, e não como parte de um tipo complexo. Tanto elementos como atributos podem ser declarados globalmente.

Desvantagens ao empregar o XML

Como em todas as tecnologias existentes, o emprego do XML traz algumas desvantagens.

O XML toma muito espaço para representar os dados que poderiam ser similarmente modelados usando um formato binário ou um arquivo de texto simples. A razão para isto é simples: este é o preço que se paga para ter informações legíveis pelos seres humanos, e uma plataforma neutra e estruturada (ZAPTHINK, 2001). Vale lembrar que esta diferença de espaço não é insignificante. Os documentos XML podem ser de três a vinte vezes maiores quando comparados com arquivos binários ou substituídos por representações em arquivos texto. Os efeitos deste “espaço” não podem ser subestimados. É possível que 1 GigaByte de informações pode resultar em mais de 20 GigaBytes de informações codificadas em XML.

Ainda levando-se em consideração a questão de espaço, não se pode esquecer que os computadores precisam processar as informações. Grandes documentos XML devem ser carregados na memória antes do processamento; alguns documentos XML com os

valores GigaBytes no seu tamanho podem tornar o seu processamento lento (ZAPTHINK, 2001).

Toda estrutura/DTD XML corre o risco de se tornar um formato "proprietário" devido à natureza do XML, que permite aos desenvolvedores escolherem um conjunto de *tags* e a representação dos seus dados. Esta "abertura" do XML permite o mesmo dado ter representações diferentes em diferentes documentos, mesmo seguindo um contexto comum, assim, cada conjunto de atores, em um determinado contexto, pode especificar os mesmos dados de maneira diferente conforme a sua necessidade.

Todos os atores envolvidos na troca de mensagens XML devem conter o(s) DTD(s) utilizado(s) pela(s) mensagem(ns) trocada(s). Os DTDs devem ser acordados e detalhados previamente pelos atores que irão utilizá-los, e esta tarefa nem sempre é fácil.

Os documentos XML não são adequados para representar valores binários. Imagens, vídeos, sons e outros formatos são representados no formato binário. Como o XML trabalha com a representação em texto, transmitir este tipo de dado se torna ineficiente.

2.2 Aplicações Isoladas das Tecnologias Apresentadas:

2.2.1 Aplicações fazendo uso do MMS

Existem muitas empresas e trabalhos relacionados com o uso do MMS como uma interface padrão para a comunicação com dispositivos industriais. Dentre algumas empresas podemos citar: Boeing, Digital Equipment Corporation e SISCO.

A Boeing (BOEING, 1998) utiliza a linguagem de comunicação MMS para permitir um computador Boeing "de mais alto nível" questionar os controladores dos dispositivos dados de status.

A Digital Equipment Corporation (DIGITAL EQUIPMENT CORPORATION, 1998) desenvolveu um software chamado DEComni MMS, um produto para

proporcionar uma solução para conexão e gerenciamento ou controle de dispositivos e sistemas. O DEComni MMS implementa as especificações ISO/IEC 9506-1 e ISO/IEC 9506-2. Quando combinado com pré requisitos de Hardware e Software, DEComni MMS também pode interoperar com outros sistemas seguindo as especificações ISO.

A SISCO aborda o MMS em vários trabalhos onde são propostas várias soluções ou interfaces para o acesso aos dispositivos industriais, como SISCO AX-S4 MMS – "*access for MMS*" (SISCO, 1998b), – ferramenta que permite a integração da área de trabalho (*desktop*) com dados em tempo real para aplicações compatíveis para o Sistema Operacional MS-Windows, permitindo aos usuários interagirem e visualizarem os dispositivos e objetos MMS, o DWG (SISCO, 1996) – o Device GateWay é um dispositivo que permite integrar dispositivos como CNCs, robôs e outros controladores que usam protocolos de comunicação comuns ou outros métodos baseados em MMS. Aplicações baseadas em MMS podem se comunicar com DWG se forem compatíveis com os dispositivos MMS conectados na rede e o MMS-EASE Lite (SISCO, 1998c) – pacote de código para MMS atuar em sistemas embarcados que suporta todos os serviços MMS para funções de aplicações clientes e servidores, permitindo o processamento de mensagens síncronas e assíncronas, dependendo dos requisitos das aplicações.

Em PINHO (1998) são estudadas e avaliadas as dificuldades e a complexidade nos aspectos de comunicação de sistemas informáticos distribuídos nas aplicações industriais em termos de desempenho, disponibilidade, confiabilidade e flexibilidade. Visando a solução destes "problemas", a interconexão de diversos equipamentos na indústria de manufatura é abordada utilizando o protocolo MAP, com atenção especial ao estudo e serviços do protocolo MMS. Estes serviços visam disciplinar a operação corporativa entre diferentes equipamentos programáveis e, desta forma, minimizar os problemas relacionados com a interoperabilidade entre dispositivos heterogêneos. Uma solução neste nível obtém a interoperabilidade de equipamentos heterogêneos e possibilita a integração das mais diversas atividades componentes do ciclo industrial.

WILLRICH (1991) propõe um modelo de implementação do padrão de comunicação MMS numa arquitetura Mini-MAP, de acordo com o padrão da MMSI[33], visando o compartilhamento de informações nos diversos níveis do processo de manufatura para sua melhor coordenação, aumentando a sua eficiência e conseqüentemente a sua produtividade. Para auxiliar nesta tarefa, foram abordados os principais conceitos existentes no ambiente CIM, que conduzem a tomada de decisões importantes relativas a implementações da MMSI.

Em SILVEIRA (1991) é estudada a utilização dos serviços propostos na camada MAP para especificar as mensagens de manufatura (MMS) e também é proposta uma metodologia para auxiliar o implementador na utilização do padrão MMS em aplicações na área de automação do chão de fábrica. As operações de uma célula de manufatura simuladas permitem compreender o comportamento da célula e inferir a respeito dos seus requisitos de comunicação. Neste trabalho ficam claras as vantagens do uso dos serviços MMS destinados à comunicação de uma célula e sobre como é possível fazer a modelagem dos diversos dispositivos industriais através dos objetos existentes no MMS. Por outro lado, são atacadas apenas as funcionalidades de como o MMS trabalha com o dispositivo industrial, não se tendo a preocupação de como trabalhar uma melhor integração das interfaces MMS (através do VMD) para o mundo exterior, ou de que maneira os clientes não-MMS poderiam ter acesso as informações e dados na célula.

Em RABELO (1997) o protocolo MMS é usado em um módulo onde um cliente qualquer pode se comunicar com os recursos de produção existentes em uma empresa. Nesta abordagem foi implementado um "servidor MMS", ou um VMD, sobre cada um dos recursos de produção. Os clientes que querem acessar as funcionalidades do chão de fábrica devem possuir uma interface MMS para poder se comunicar com o VMD, e este, por sua vez, faz a ligação entre os serviços MMS e os dispositivos industriais e os seus sistemas legados. Nesta abordagem, todos os clientes que querem comunicar com o dispositivo industrial devem, no mínimo, conhecer a interface MMS proposta, caso contrário não é possível o acesso às informações.

[33] MMSI – *Manufacturing Message Specification Interface*

2.2.2 Integração do MMS com CORBA

O uso do MMS com o CORBA é explorado em GUYONNET *et al.* (2001), onde é proposta a união destas duas tecnologias como um ambiente promissor para aplicações de controle do processo de manufatura. Foi explorado um ambiente para combinar as vantagens funcionais do padrão MMS e as características do CORBA. Na arquitetura MAP a estruturação é feita em camadas. Desta forma, o VMD se torna um objeto CORBA e as suas interfaces (do VMD) correspondem aos serviços que foram implantados e declarados em uma IDL CORBA. Como a maioria dos serviços MMS são confirmados – recebem o retorno de reconhecimento, os serviços MMS são mapeados em métodos de invocação síncronos. O ORB é usado para acessar o objeto VMD remoto, usando as facilidades de transporte do CORBA. Quando um cliente requisita a execução de um serviço MMS, o serviço de *binding* encontra a referência ao objeto requisitado e invoca o método correto. Os resultados são retornados ao cliente no final do processamento. Quando um servidor MMS quer relatar uma informação não solicitada ele usa o ORB para acessar o cliente remoto, porém não são enviadas mensagens de "resultado" e o servidor MMS (agora no papel de cliente) não fica bloqueado. O MMS em conjunto com CORBA oferece um ambiente orientado a objetos para controle de processos em manufatura que proporciona um ambiente uniforme para resolver a integração de aplicações heterogêneas. Nesta abordagem o servidor MMS deve conhecer todos os seus clientes, pois quando uma mensagem não solicitada for gerada ele deve saber para quem enviar, quando tratamento de exceções ou erros. Também nesta abordagem não é possível o tratamento de mensagens assíncronas por parte dos clientes – os clientes devem sempre esperar o final do processamento das mensagens para "retomar" os seus serviços. Vale lembrar que o MMS, originalmente, não foi concebido para trabalhar com TCP/IP e comunicação assíncrona.

Em ARIZA (1998a e 1998b) e ARIZA *et al.* (2001a e 2001b) é apresentado um modelo de integração entre os serviços MMS e CORBA para a comunicação entre dispositivos e contribuir para a construção de sistemas de manufatura de forma distribuída. A arquitetura CORBA permite modelar/dividir o sistema em objetos e a comunicação destes objetos pode ser feita levando em conta os eventos de gerenciamento

do VMD. Um VMD geral foi criado e ele pode ser herdado por outras classes, permitindo a construção de um VMD específico para qualquer dispositivo de manufatura. Já o MMS não permite herança, mas esta característica é alcançada através das características do CORBA. O objetivo principal é construir classes que podem ser usadas como base para implementação de objetos VMD que representam outros dispositivos. Neste sentido, o desenvolvimento nesta plataforma é auxiliado pois o ORB permite aos servidores – objetos MMS – serem tratados como objetos. O servidor MMS é visto como um objeto CORBA que pode ser acessado através do ORB. Os serviços providos pelo objeto são aqueles especificados no padrão MMS para os servidores, e a descrição dos objetos é especificada usando IDLs. Reusabilidade e modularidade são as características alcançadas neste tipo de abordagem. Isso permite ter uma base para implementar novos clientes usando herança, e o CORBA como uma plataforma de comunicação para executar as interações entre clientes e servidores, facilitando a criação do controle de sistemas distribuídos. Porém, assim como em (GUYONNET *et al*. 2001), há a necessidade do servidor MMS ter um mapeamento dos seus clientes para quando uma mensagem não solicitada for gerada no servidor MMS ele saber para quem mandar esta mensagem.

2.2.3 Integração do KQML com CORBA

Em BENECH *et al.,* (1997) é explorada uma proposta para suprir os requisitos de comunicação entre agentes com o objetivo de aumentar a escalabilidade e flexibilidade das soluções em ambientes distribuídos. Como suporte à comunicação no alto nível entre os agentes foi usado o KQML, no papel de um protocolo de interação, e a arquitetura CORBA, como o protocolo de transporte. Nesta proposta, a plataforma CORBA adiciona naturalmente a transparência de locação, controle de acessos e serviço de nomes, facilitando todas as performativas e os conceitos KQML. Quando o agente (aplicação) necessita/quer se comunicar com outros agentes ele deve utilizar uma linguagem de comunicação entre agentes, o KQML. Desta forma o agente gera as performativas que irão representar os seus requisitos e ações. O interfaceamento do KQML com CORBA é feito através das interfaces IDL, onde dentro das IDLs existem os métodos referentes a cada performativa KQML. A grande desvantagem dessa abordagem é que os agentes

devem incorporar ao seu código todas as performativas existentes na IDL, mesmo que muitas destas performativas nunca venham a ser usadas.

2.2.4 Uso dos Agentes e Sistemas Multiagente

A seguir são citados alguns trabalhos que utilizam agentes e sistemas multiagente.

Em BARBER *et al.* (1998) são apresentadas algumas vantagens de projetar sistemas de controle em manufatura baseados em agentes, onde os agentes têm o papel de proporcionar o dinamismo, adaptabilidade e autonomia ao sistema. São abordados assuntos como os níveis de autonomia do(s) agente(s) no controle de chão de fábrica (como, quando e onde cooperar) e plano de ações e resolução de conflitos, visando vencer os desafios para a automação de tarefas industriais, como por exemplo mudanças nos programas de produção, quebra de máquinas e atualização dos equipamentos no chão de fábrica. Por outro lado, no referido artigo, não existe a preocupação de como transmitir as decisões tomadas, sejam elas em grupo, seja individualmente, aos dispositivos de chão de fábrica propriamente ditos.

CHANDRA *et al*. (2000) apresenta uma proposta baseada em Sistemas Multiagente para auxiliar os princípios do suporte às tecnologias do Gerenciamento de Cadeias de Produção. Nos sistemas multiagente os agentes cooperam para alcançar um objetivo comum, tornando este tipo de implementação uma das mais promissoras para o suporte e o gerenciamento de redes de cadeias de fornecedores. Os agentes representam a diminuição dos custos de operação (negociação); conseqüentemente a redução do tempo para a realização destas tarefas é evitando, para as cadeias de produção em rede, "gargalos" operacionais. Neste trabalho o problema da interoperabilidade entre os agentes não foi abordado de maneira exaustiva, não propondo soluções ou apontando os problemas existentes, desde a simples comunicação entre os agentes até os protocolos de negociação.

Em LEITÃO *et al.* (2001) são discutidas as oportunidades de usar a tecnologia multiagente em processos de automação e sistemas de manufatura distribuída. Uma aplicação para o controle de uma célula de manufatura foi descrita. Os resultados foram

comparados com outras aplicações de controle desenvolvidas usando as abordagens tradicionais nas células e mostraram que o uso de sistemas multiagente traz as vantagens de reusabilidade de código – uma vez os agentes desenvolvidos eles podem ser "reaproveitados" em outros "mundos"/cenários, distribuição e autonomia – cada agente tem autonomia, controle sobre o seu comportamento local e conhecimento sobre a comunidade, podendo solucionar problemas localmente sem a intervenção de um operador humano ou não, porém, o desenvolvimento de sistemas multiagente não se adapta a implementações de pequenas aplicações ou aplicações específicas devido à alta complexidade do desenvolvimento dos agentes e a pouca reusabilidade de código quando desenvolvido para aplicações de domínios restritos.

2.2.5 Exemplos do uso do XML no "intercâmbio" de dados

CULLEN (2001) traz no seu artigo a explanação das vantagens do XML trabalhando como um "modelo intermediário" ou uma "ponte" para interoperabilidade de dados legados. O XML permite o intercâmbio dos dados, que na maioria das vezes, tem estruturas de dados hierárquicos diferentes. Os dados legados devem ser convertidos diretamente para o formato XML. Uma vez estes dados convertidos, existe um enorme número de opções para receber e processar informações codificadas em XML. Além do mais, o XML traz as habilidades de publicar os dados legados como dados XML diretamente em um *browser* na Web.

FRICK (2000) apresenta uma infraestrutura para uma "Casa de Projetos Virtuais" visando o suporte e a colaboração de engenheiros no desenvolvimento de processos distribuídos, especialmente nos relacionamentos da manufatura e seus fornecedores. O projeto se concentra no compartilhamento de dados de *workflows* através dos limites da organização. O projeto é composto por um grupo distribuído de "Servidores de Projetos" e aplicações cliente "Browser de Projeto". A interface de integração entre as diferentes organizações está baseada em padrões de Internet, como o XML (serialização de dados) e o HTTP (comunicação). Algumas possíveis extensões podem ser feitas no projeto utilizando ferramentas para o gerenciamento dos dados e implementando cenários adicionais de *workflow*.

Em seu artigo RABELO (*et al.* 2002) descreve um sistema de suporte a decisões multiagente chamado *SC*^{2} [34], desenvolvido para auxiliar as empresas na coordenação de negócios distribuídos. Através do SC^2 é possível ter um ambiente integrado para melhor gerenciar as cadeias de produção, distribuição e vendas. As informações entre os membros da cadeia de produção são trocadas através de documentos XML. Como infraestrutura de comunicação é utilizada a arquitetura CORBA. Estas duas tecnologias adotadas em conjunto garantiram o intercâmbio entre os dados trocados na cadeia de produção, sendo os nós da cadeia instalados em diferentes países.

2.2.6 Troca de dados no ambiente industrial

Uma plataforma para a troca de informações industriais distribuída é proposta por VERÍSSIMO *et al.* (1996) com o objetivo de integrar todo o fluxo de informações, desde os níveis de supervisão até o nível de comandos e controle, abordando os sistemas industriais como naturalmente distribuídos. Neste trabalho são usados vários protocolos padrão com o objetivo de facilitar a integração das informações. Também é abordado o "tratamento" dos dados no fluxo chão de fábrica – supervisão, e vice-versa. Para isso, é usada a plataforma NavCim, que oferece funcionalidades para supervisão de processos, controle industrial e suporte para o gerenciamento de tarefas. Esta é uma abordagem que explora a integração e o tratamento das informações nos níveis industriais, porém as diversas funcionalidades disponíveis pelos sistemas multiagente não são empregadas. Estas funcionalidades poderiam auxiliar nos vários níveis existentes na proposta.

O trabalho de CARVALHO *et al.* (1996) está voltado para a integração de várias áreas dentro de uma indústria, usando diferentes padrões de comunicação, com o objetivo de integrar as "ilhas de informações" existentes dentro das indústrias e ainda preservar os antigos investimentos em tecnologia. Isso quer dizer que o método adotado foi de "encapsular" os equipamentos antigos e criar um padrão para a troca de informações. Os dados do chão de fábrica são trocados através do protocolo MAP/MMS e armazenados em uma base de dados. As consultas a esta base de dados são feitas através de SQL,

[34] SC^2 – *Supply Chain Smart Coordination* – http://www.inescporto.pt/~damascos/

permitindo a atualização dos dados da manufatura nos demais níveis "acima". Existe uma aplicação "mestre" (*Master Production Planning*) responsável pela tomada de decisões e responsável por "distribuir" as informações.

Em SMITH *et al.* (1995) o trabalho é focado na criação de um software para controle do chão de fábrica usando a metodologia de ferramentas CASE[35] capaz de proporcionar a integração dos diferentes níveis dos sistemas de manufatura. Para isso uma arquitetura de controle foi descrita seguindo o modelo de três níveis hierárquicos. Os níveis são introduzidos para reduzir a complexidade limitando a autoridade; cada nível tem um planejamento distinto. A proposta estende arquiteturas existentes proporcionando definições formais aos componentes de arquiteturas individuais. No desenvolvimento de um sistema específico para o controle do chão de fábrica permite a descrição de sistemas de manufatura sem ambigüidades, proporcionando assim, uma estrutura genérica pela qual uma implementação específica pode ser descrita. Esta descrição formal serve como base para o desenvolvimento do controle, incluindo funções de planejamento, escalonamento e execução. Trabalhando com a separação da execução do escalonamento e planejamento, é possível o desenvolvimento de módulos de execução genéricos. Estes módulos são desenvolvidos usando tarefas genéricas derivadas de características físicas dos controladores individuais e são representadas usando modelos formais do controlador. Através desta estrutura é possível a implantação independente da linguagem de implementação a ser adotada. As tarefas de desenvolvimento e manutenção são simplificadas com a adoção de descrições formais e o desenvolvimento de metodologias.

2.3 Modelos Tradicionais de Comunicação com o Chão de Fábrica

As exigências de comunicação entre unidades para integração flexível dos sistemas de automação traduzem a necessidade de uma especificação de redes locais para aplicações industriais diferente daquela adotada em automação de escritório.

[35] CASE – *Computer Assisted Software Engineering*

A arquitetura de redes de comunicação industrial deve integrar sistemas heterogêneos de diferentes fabricantes, suportando tanto a operação de chão de fábrica quanto as funções de apoio à produção. A definição de padrões de protocolos de comunicação e a sua adoção por diferentes fabricantes tem como objetivo permitir a conexão dos vários dispositivos de chão de fábrica de fabricantes diferentes. Algumas das iniciativas para a padronização das redes industriais são: Projeto FIELDBUS, projeto PROWAY e projeto IEEE 802.

2.3.1 Projeto FIELDBUS

O Fieldbus é uma solução de comunicação para os níveis hierárquicos mais baixos dentro da arquitetura fabril, interconectando os dispositivos primários de automação instalados na área de campo e os dispositivos de controle de nível imediatamente superior (FIELDBUS, 2003a).

O Fieldbus é uma linha de comunicação serial, digital, bidirecional para interligação dos dispositivos primários de automação instalados na área de campo e os dispositivos de controle de um sistema integrado de automação e controle de processos (WILLRICH, 1991).

Segundo FIELDBUS (2003b) a adoção do Fieldbus traz as vantagens econômicas: a redução de fiação pela utilização de um meio físico compartilhado; redução do número de canais de comunicação com o processo; redução do tempo e complexidade do projeto de layout; técnico-operacionais: facilidade de instalação e manutenção, pela manipulação de um menor número de cabos e conexões; facilidade de detecção, localização e identificação de falhas através de funções de monitoração automática; maior modularidade no projeto e instalação, aumentando a flexibilidade e expansão de funções e módulos; sistêmicas: aumento da consciência e da confiabilidade da informação advinda dos instrumentos de campo, através da digitalização e pré-processamento; maior facilidade de interconexão entre níveis hierárquicos diferentes de automação;

mercadológica: desacoplamento do software de supervisão da dependência de um fornecedor específico do Hardware.

Enquanto que os segmentos de banda base do MAP permitem a realização de tempos de resposta de cerca de 100ms, sistemas do tipo Fieldbus se propõem a reduzir este tempo para a faixa de 1 a 10ms, como requerido para o controle e supervisão de importantes grandezas envolvidas na automação (FIELDBUS, 2003a).

Um aspecto essencial na definição de protocolos de comunicação para Fieldbus é a redução a um mínimo estritamente necessário das informações adicionais incluídas nas mensagens, de forma a permitir uma performance adequada em tempo real.

Em função do tipo de aplicações que se propõem atender, um conjunto de requisitos básicos são impostos ao Fieldbus:

- Necessidade de elevado desempenho para atender as aplicações com requisitos de tempo críticos;
- Método e meio de transmissão simples e de preço acessível;
- Necessidade de consistência nos dados;
- Necessidade de serviços compatíveis com redes dos níveis hierárquicos superiores – compatibilidade com serviços MAP.

Dentre os sistemas Fieldbus atualmente em discussão, os mais fortes candidatos à normalização são o PROFIBUS[36] e o FIP[37]. Nos Estados Unidos existe o grupo de estudos SP-50 da ISA[38] para a consolidação das propostas FIP e FIELDBUS.

Ao lado das questões de normalização permanecem ainda abertas à discussão questões referentes ao espectro de aplicações a ser atendido pelo Fieldbus. Por um lado é desejada uma boa compatibilidade e interconectividade com os níveis hierárquicos superiores de automação, por outro lado devem ser considerados os requisitos técnicos e econômicos para a conexão de componentes simples e de baixo custo.

---

[36] PROFIBUS – *Process Filedbus*, proposta alemã
[37] FIP – *Factory Instrumentation Protocol*, proposta francesa
[38] ISA – *Instrumentation Society of America*

Enquanto soluções que permitam uma compatibilidade com a definição da camada de aplicação adotada no sistema MAP através da definição de subconjuntos do MMS são preferidas, permanecendo aberta a questão de adequação de sistemas para o acoplamento direto de sensores e atuadores em processos com dinâmica elevada.

Um dos fatores que impedem a ampla adoção do Fieldbus é a não definição de um padrão internacionalmente aceito. Definido este padrão o Fieldbus poderá revolucionar o setor de instrumentação, permitindo que a inteligência das aplicações seja totalmente distribuída pelo campo e favorecendo o surgimento de dispositivos com capacidade locais de processamento mais sofisticadas.

2.3.2 Projeto PROWAY

O projeto PROWAY – Process Data Highway, teve por objetivo a normalização de redes de comunicação para controle de processos. Os seus trabalhos foram iniciados em 1975 em um grupo da IEC[39]. Em função dos trabalhos de padronização internacional o projeto passou por diversas fases (Proway A, C e C). Proway A e B utilizavam o protocolo HDLC[40] da ISO na camada de enlace, adequando-se assim apenas a sistemas tipo mestre/escravos. Proway C adotou a técnica de *token-passing*. Sua arquitetura é composta de quatro camadas do modelo de referência ISO/OSI, denominadas *Line* – camada física, *highway* – camada de enlace, *network* – camada de rede e *application* – camada de aplicação (ISA, 2003).

2.3.3 Projeto IEEE 802

O IEEE[41] iniciou em 1980 o projeto conhecido pelo número 802 (IEEE, 2003) e define originalmente uma série de normas para as camadas Física e Enlace do modelo de

[39] IEC – *International Electrotechnical Comission*
[40] HDLC – *High Level Data Link Control*
[41] IEEE – *Institute of Electrical and Electronics Engineers*

referência OSI. Na proposta do IEEE, a camada de Enlace é subdividida em duas camadas: LLC[42] e MAC[43]. A norma é hoje conhecida sob a designação ISO/IEC 8802.

O projeto ISO/IEC 8802 define uma norma com doze partes que tratam de vários aspectos ligados a redes de computadores as normas de maior relevância são:

- Norma IEEE 802.3 (CSMA/CD[44]) – descrição da sub-camada MAC e camada Física para as redes com topologia em barramento e método de acesso ao meio CDMA/CD;
- Norma IEEE 802.4 (barramento com ficha – *Token Bus*) – descrição da sub-camada MAC e camada Física para as redes com topologia em barramento e método de acesso ao meio baseado em *token-passing* (Token Bus);
- Norma IEEE 802.5 (anel com ficha – *Token Ring*) – descrição da sub-camada MAC e camada Física para as redes com topologia em anel e método de acesso ao meio baseado em *token-passing* (Token Ring);
- Norma IEEE 802.11 (redes sem fio – *Wireless Netoworks*) – padroniza LANs[45] com MAC sem fio (*Wireless*) e a camada física correspondente.

Dentre todos os modelos tradicionalmente apresentados, o fluxo de informações ocorre em dois sentidos: dados de monitoração, informações de status do chão de fábrica sobem para os níveis mais altos da hierarquia, em cada nível, o volume de informação é processado, gerando sinais de controle e ajustes que descem para os níveis mais baixos. Os níveis mais baixos são caracterizados por mensagens freqüentes e curtas, transmitidas sobre distâncias curtas, como por exemplo atribuição de controle. As mensagens nos níveis mais altos incluem normalmente a transferência de grandes arquivos e interações humanas de entrada e saída (WILLRICH, 1991).

2.3.4 Abordagens Centralizadas *versus* Distribuídas

[42] LLC – *Logical Link Control*
[43] MAC – *Medium Access Control*
[44] CSMA/CD – *Carrier Sense Multiple Access.*
[45] LAN – *Local Area Network*

A abordagem tradicional de suporte à integração de informações industriais trabalhando com a filosofia CIM define uma hierarquia organizacional objetivando a robustez, heterogeneidade e escalabilidade dos sistemas de manufatura. O modelo padrão da ISO para automação industrial divide a empresa em seis níveis hierárquicos (KUSIAK, 1989):

- Nível Empresa: gerenciamento da corporação, assuntos financeiros, pesquisas, desenvolvimentos, entre outros.
- Nível de Instalação: implementação das funções do nível empresa, como por exemplo o planejamento da produção, planejamento de processos.
- Nível de Seção: coordenação da produção através do gerenciamento de tarefas e recursos controlando a produção e atividades de suporte que serão executadas pelos controladores de células.
- Nível de Célula: controle de execução do plano de produção.
- Nível de estação: cada controlador de célula monitora um conjunto de dispositivos que são responsáveis pela seqüência de operações necessárias para completar o plano de trabalho.
- Nível de Equipamento: dispositivos responsáveis pela execução dos comandos que lhe foram transmitidos pelo controlador de estação.

Até meados dos anos 80 os supervisores das células industriais eram centralizados em uma única máquina, tipicamente um computador supervisor. Em contraste com o processamento e/ou planejamento centralizado de sistemas, uma abordagem descentralizada naturalmente divide o problema global complexo em problemas menores e específicos, que são tratados como unidades independentes na busca da sua solução (AZEVEDO *et al*. 1996). Com a evolução da tecnologia dos sistemas distribuídos, muitos dos serviços que eram centralizados em um único supervisor foram "separados" em vários supervisores distribuídos. Isto teve o intuito de melhorar o desempenho, a segurança, a flexibilidade e escalabilidade das aplicações dos sistemas industriais.

Na verdade, a adoção de abordagens distribuídas vem na direção de buscar soluções mais eficientes para os problemas das abordagens centralizadas. Os principais problemas associados com as abordagens centralizadas segundo LEITÃO *et al.* (2001) são:

- Reconfiguração: ajusta-se às aplicações que apresentam uma estrutura organizacional rígida. Entretanto quando é necessário mudar a estrutura organizacional surgem grandes problemas.
- Aprendizado e perturbação do conhecimento: é difícil e complexo introduzir inteligência nas aplicações para otimizar suas execuções e gerenciar as perturbações e avisos.
- Distribuição e descentralização: este modelo não suporta uma distribuição e descentralização das entidades de forma eficiente.

Nas abordagens dos sistemas atuais, normalmente centralizadas, é possível alcançar um certo "grau" de flexibilidade, porém este nível alcançado muitas vezes está aquém daqueles desejados e/ou necessários. Isto se deve ao fato de limitações naturais dos sistemas centralizados, como por exemplo, a disponibilidade do sistema no processamento dos dados e tomada de decisões é igual a disponibilidade do computador responsável por esta tarefa, não existe a divisão de tarefas; também quando se pensa na ampliação de arquiteturas centralizadas (adição de novos componentes ou máquinas) é exigido um grande ajuste ou até mesmo uma reprogramação total nos sistemas responsáveis por controlar os dispositivos; isso tudo sem levar em consideração que se o "computador central" parar, todo o sistema parará também. Assim estas limitações não impossibilitam que os sistemas trabalhem, mas trazem um grande peso na manutenção e não permitem agilidade e flexibilidade, características necessárias para as empresas que querem competir no mercado atual.

A Figura 14 ilustra como é a maneira tradicional de um controlador de uma célula de manufatura usando uma abordagem centralizada.

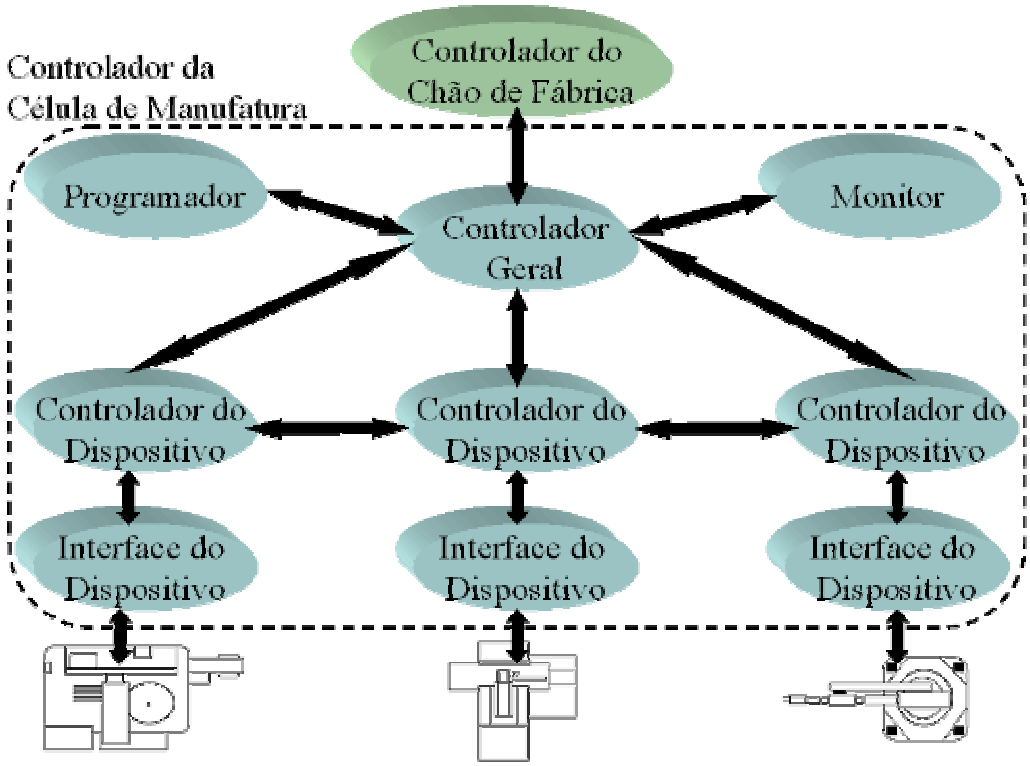


Figura 14 – Controlador de Célula de Manufatura Usando uma Abordagem Centralizada

Os sistemas industriais distribuídos são compostos por muitos elementos (computadores e outros dispositivos), enquanto que existe apenas um elemento de tomada de decisão nos sistemas centralizados (BRITO *et al*. 2000). Algumas vantagens dos sistemas distribuídos sobre os sistemas centralizados são:

- Distribuição inerente: existem aplicações que são inerentemente distribuídas, por exemplo, aplicações de manufatura.
- Robustez: sistemas centralizados são dependentes de um único componente; diferentemente, os sistemas distribuídos são tolerantes a faltas; isto é, se uma máquina parar, isto não quer dizer que o sistema todo irá parar.
- Escalabilidade: o poder de computação pode ser aumentado, quando necessário, aos poucos.

Em contrapartida a estas vantagens, algumas importantes desvantagens, segundo BRITO (*et al*. 2000) são:

- Heterogeneidade: os componentes conectados são usualmente de diferentes fabricantes e suas funções são diferentes.
- Comunicação: a interconexão dos componentes do sistema deve ser tratada, e como resultado, novas camadas de comunicação devem ser adicionadas.
- Software: desenvolver software para estes sistemas pode se tornar uma tarefa complexa, principalmente devido à heterogeneidade dos componentes de sistemas e as camadas mais baixas de comunicação.

Os novos requisitos apresentados para o controle de sistemas são principalmente a distribuição e descentralização das entidades, conhecimento e habilidades, cooperação entre entidades distribuídas, evolutividade de entidades e componentes, mecanismos de transparência da comunicação para uma melhor integração, cooperação de entidades distribuídas e interação com dispositivos físicos (LEITÃO *et al.* 2001).

Sistemas Multiagente têm sido vistos como uma adequada tecnologia de informação para atuar em ambientes distribuídos, pois eles naturalmente proporcionam um conjunto de características que facilitam a implementação e a operação de sistemas distribuídos complexos (AZEVEDO *et al.*, 1996) e (RABELO, 1998). A tecnologia de Sistemas Multiagente para manufatura e automação de sistemas visa atender da melhor forma possível os requisitos de autonomia – um agente pode operar sem a intervenção direta de entidades externas e tem algum controle sobre o seu comportamento; cooperação – os agentes interagem uns com os outros para alcançar um objetivo comum; reatividade – os agentes percebem o ambiente e respondem de forma rápida às mudanças que nele ocorrem; proatividade – os agentes não simplesmente agem em resposta ao seu ambiente, mas são capazes de tomar iniciativas e controlar o seu ambiente; e descentralização – os agentes podem ser organizados em estruturas descentralizadas conduzindo a rápidas adaptações ao sistema e permitindo o reaproveitamento de código para outras aplicações, aumentando a flexibilidade e permitindo o desenvolvimento de softwares modulares. De fato, um conjunto de agentes comunicando e cooperando entre si pode solucionar problemas mais complexos e aumentar a funcionalidade e a eficiência dos processos em que eles estão envolvidos se comparado com arquiteturas centralizadas.

3. Proposta e Modelo Conceitual

Este trabalho propõe um ambiente global de integração para sistemas multiagente industriais, aberto, robusto, padronizado e transparente nos vários níveis de integração requeridos. Mais especificamente, um ambiente no qual convirjam os esforços feitos sobre os protocolos e esquemas de integração do CORBA, MMS, KQML e XML, estendendo os trabalhos de RABELO (1997) e ARIZA *et al.* (2001b) como o objetivo harmonizar e encapsular a comunicação dos sistemas multiagente o máximo possível com os atores com os quais é necessário comunicar/envolver, mantendo-se as propriedades essenciais de cada protocolo dentro dos limites para os quais foram desenvolvidos. Portanto, significa permitir que uma aplicação externa, que necessita acessar dados do chão de fábrica, não precise conhecer a fundo a sintaxe e detalhes de todos os protocolos (dos vários envolvidos nos vários níveis) e as particularidades de cada controlador de máquina que se queira comunicar.

A arquitetura proposta é independente da aplicação do chão de fábrica com a qual serão feitas as interações. Este "canal de comunicação" proposto pode ser usado por qualquer dispositivo industrial ou um conjunto de dispositivos industriais que podem estar agrupados, como por exemplo, em uma célula de manufatura.

A Figura 15 ilustra a composição das entidades envolvidas e suas possíveis interações:

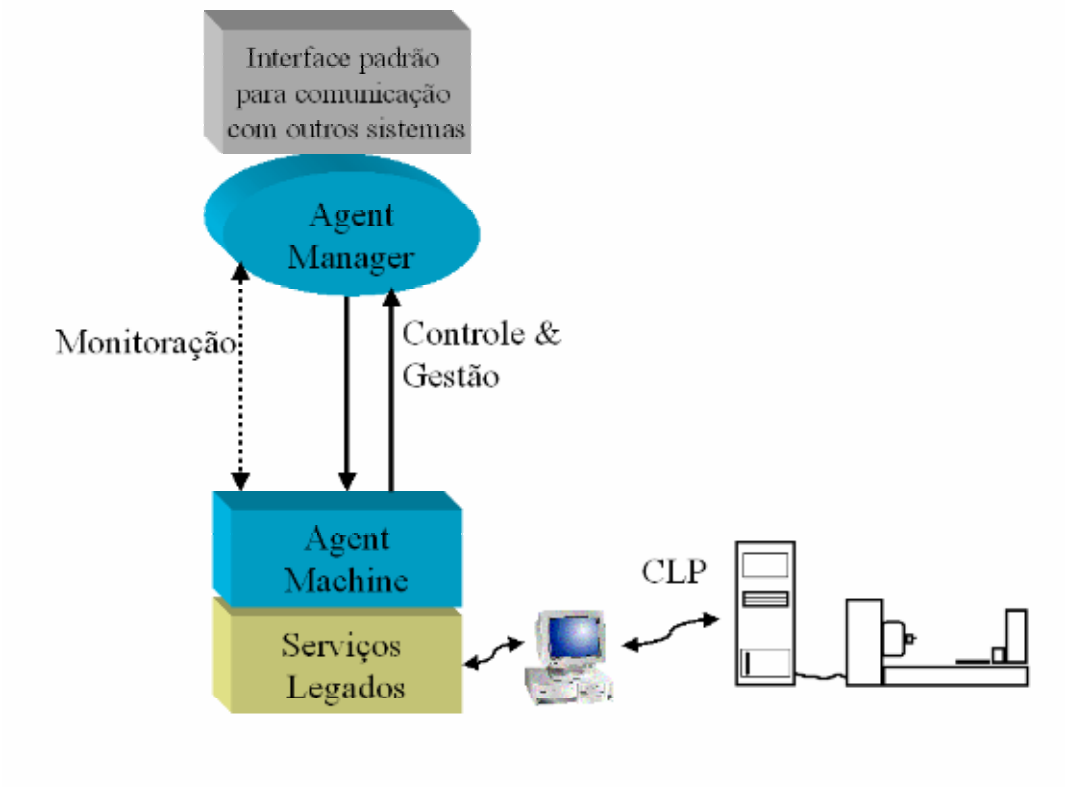


Figura 15 – Entidades Envolvidas na Proposta

Os agentes que estão em contato com o "mundo externo", *AgentManager*, são responsáveis por interfacear a comunicação com os agentes que conversam com os dispositivos do chão de fábrica.

Os supervisores distribuídos e/ou seus sistemas legados, *AgentMachine*, podem ser "agentificados" e trabalhar de forma cooperativa/"inteligente" para a solução dos seus problemas, como aumentar a robustez e a segurança das operações e das informações que trafegam no chão de fabrica. A "agentificação" de sistemas legados foi primeiramente abordada por RABELO (1994) e consiste em encapsular um dispositivo em camada(s) de software para transformar o dispositivo integrável ao sistema, assim este dispositivo pode ser representado como um agente ou conversar com um agente.

3.1 As Classes de Agentes da Proposta

Esta tarefa é complexa e implica na integração de diferentes formatos e diferentes representações de dados e mensagens nos diversos níveis industriais. Estas incompatibilidades são barreiras para uma integração eficiente, rápida e confiável. Assim, propõe-se uma solução para a problemática da interoperabilidade em sistemas multiagente industriais. Isto significa como criar um ambiente de interação com um outro ambiente completamente diferente visando a comunicação e o acesso aos dados de toda a indústria. Para isso, se propõe uma arquitetura híbrida de comunicação entre os agentes, ou seja, parte hierárquica e parte distribuída. Do ponto de vista do mundo externo, as aplicações são distribuídas e também é distribuída a tomada de decisões e coordenação dos dispositivos do chão de fábrica, porém dentro da arquitetura proposta, a troca de dados e mensagens ocorre de forma hierárquica, onde apenas um agente tem acesso a apenas um dispositivo industrial do chão de fábrica. Esta abordagem híbrida tenta minimizar os problemas e aproveitar o máximo das qualidades existentes nas duas abordagens. Nesta proposta existem quatro grande classes envolvidas na comunicação dos agentes com os dispositivos do chão de fábrica conforme ilustra a Figura 16.

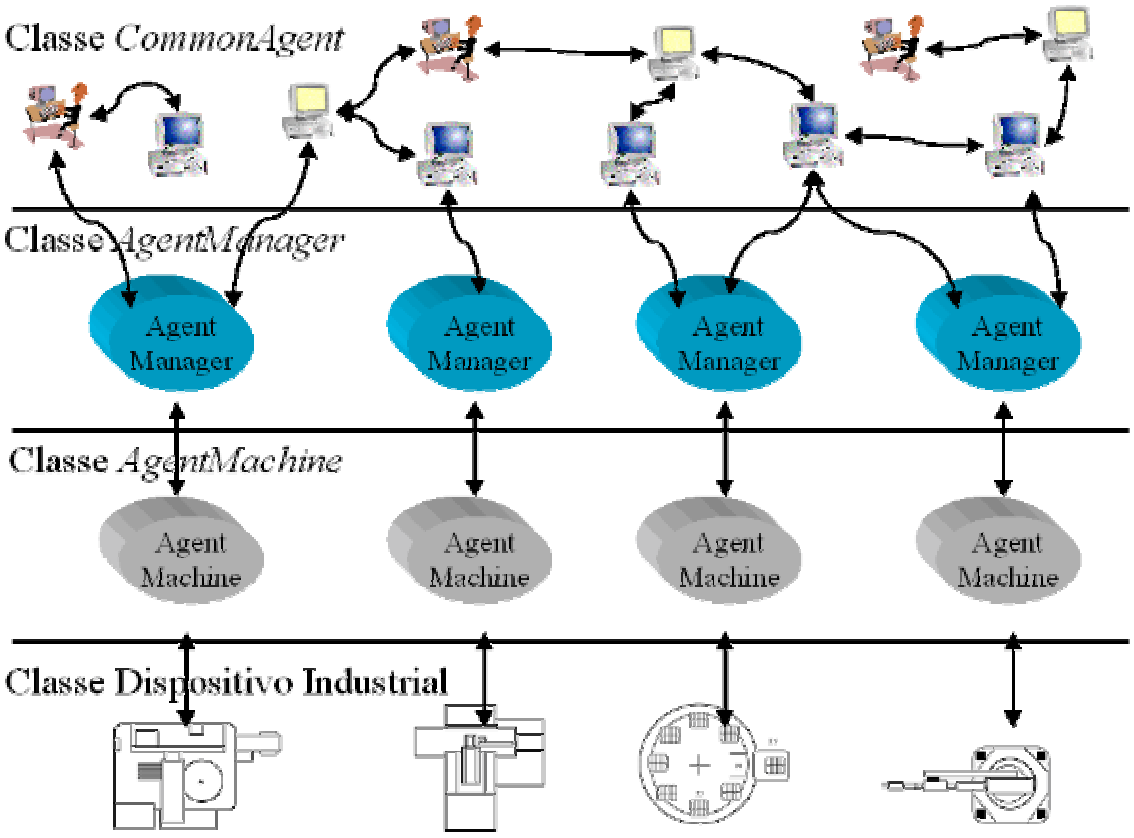


Figura 16 – Classes dos Agentes

Cada classe de agente é responsável por um determinado papel na comunicação com os dispositivos do chão de fábrica. A classe *CommonAgent* representa todo e qualquer sistema, seja ele agentificado ou não, que quer obter/enviar informações sobre/para os dispositivos de chão de fábrica, ou seja, o "mundo externo". A classe *AgentManager* representa, perante a comunidade multiagente, os agentes associados aos dispositivos de chão de fábrica mas não sabem "exatamente" como se comunicar com eles. Já a classe *AgentMachine* representa os agentes que realmente sabem como atuar diretamente com os equipamentos de chão de fábrica. A classe dispositivo industrial representa qualquer dispositivo no chão de fábrica.

A Figura 17 ilustra o fluxo de informações entre estas classes de agentes.

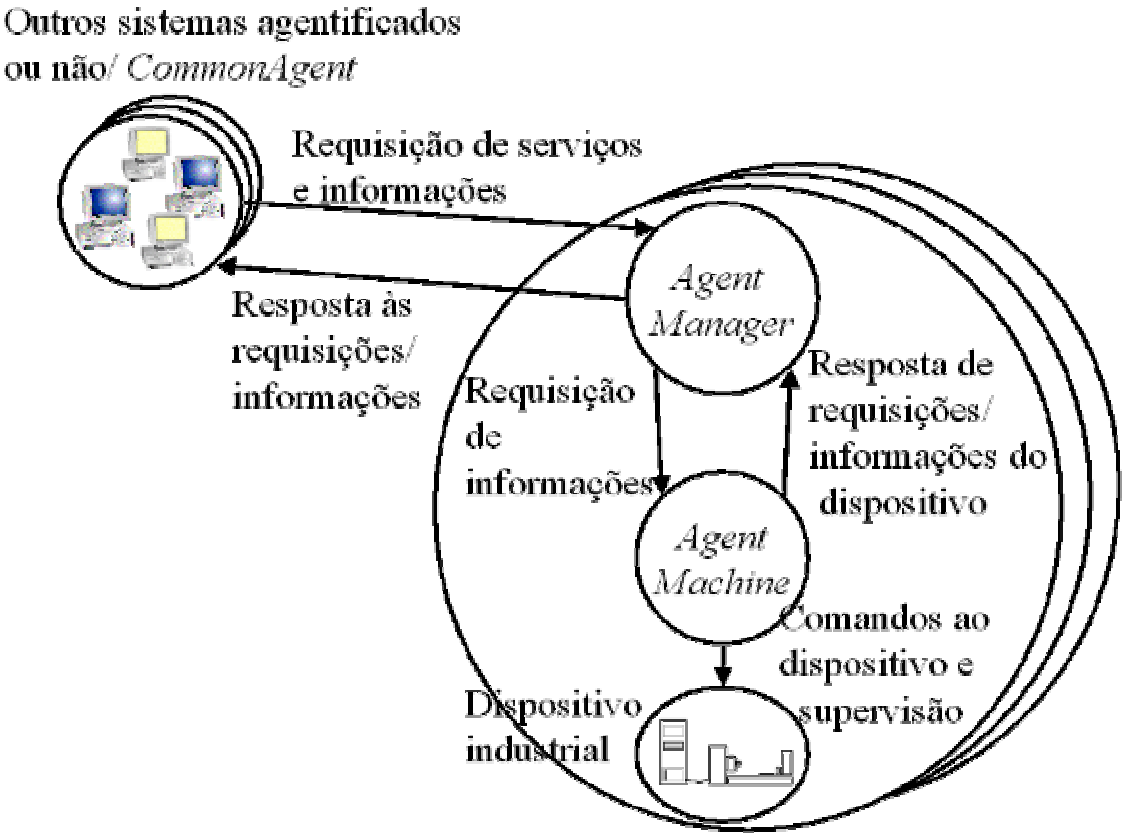


Figura 17 – Fluxo de Informações Entre as Classes de Agentes

Fez-se necessária a divisão das funções dos agentes entre as classes *AgentManager* e *AgentMachine* devido a algumas características importantes. Os agentes pertencentes à classe *AgentMachine* devem estar sempre monitorando/verificando o estado dos dispositivos industriais para o caso de algum erro ou quebra do equipamento, ou até mesmo para saber do final da execução de uma tarefa. Além disso, esta classe tem por objetivo "encapsular" os sistemas legados, muitas vezes com arquiteturas de comunicação heterogêneas, com o objetivo de tornar a sua camada de comunicação comum aos agentes da classe *AgentMachine*. Em contrapartida, os agentes da classe *AgentManager* devem estar preocupados com a "interface" com os demais agentes; por exemplo, com funções de responder/fornecer informações sobre qual dispositivo de chão de fábrica eles representam ou se o dispositivo está sendo usado, etc.

Os agentes propostos irão executar diferentes funções de acordo com os seus objetivos. Seguindo o princípio básico da divisão de trabalho, cada agente irá executar

um determinado conjunto de funções ou, em outras palavras, cada agente terá um "papel" no ambiente que ele pertence.

Os *AgentManager* são os agentes responsáveis pela comunicação de alto nível dos dispositivos industriais com os outros agentes do sistema multiagente, com outros sistemas agentificados ou não, e com os usuários (pessoas) propriamente ditos, aqui considerados *CommonAgent* (ver a seguir). O *AgentManager* também é responsável pelos processos ditos "inteligentes", como a autorização de quem pode ver o quê ou quem pode atuar de determinada forma no dispositivo industrial; por exemplo, ligar a máquina, parar o processo, etc.

Os *CommonAgent* são os agentes comuns que não conhecem as funcionalidades do chão de fábrica propriamente dito, porém sabem como se comunicar com os *AgentManager*. Todas as vezes que os *CommonAgent* quiserem obter informações sobre os dispositivos industriais ou mandar informações para estes, eles terão suas interações com os dispositivos industriais geridas através do *AgentManager*. Os *CommonAgent* fazem uso do protocolo KQML para a troca de mensagens de alto nível com os *AgentManager*.

As mensagens trafegam do *AgentManager* para o *AgentMachine* (conhecedor do MMS), empacotadas em uma mensagem KQML que está sendo enviada através do XML. Estas mensagens KQML trazem no campo *content* os comandos MMS referentes às requisições recebidas pelo *AgentManager*.

O *AgentMachine* é o agente responsável por desempacotar os comandos MMS "empacotados" (campo *content*) nas mensagens KQML encapsulada no XML e atuar diretamente com o sistema legado do dispositivo através da interface MMS (VMD, seta contínua da Figura 15). Em (SHANG *et al*. 2002), (RABELO, 1997), (SISCO, 1998b) e (ARIZA *et al.*, 2001b) ficam claras as vantagens da utilização do MMS através do VMD pelos *AgentMachine* pela interoperabilidade e independência proporcionadas, quais sejam: 1) Interoperabilidade, o modelo VMD em MMS provê todos os tipos de controle do dispositivo ao "usuário" (*AgentMachine*) sem expor os detalhes de como as funções disponibilizadas são mapeadas para os dispositivos reais ou implementadas internamente

nos dispositivos; 2) Independência, o protocolo de comunicação MMS proporciona independência do dispositivo que está sendo virtualizado e é independente também do meio de acesso (RPC[46], CORBA, etc.).

O *AgentMachine* também é responsável por avisar ao *AgentManager* quando qualquer problema ocorrer durante o funcionamento do dispositivo industrial; desta forma o *AgentManager* pode atuar de forma ágil na solução dos problemas de manufatura. Com este "serviço de informações" sobre o problema o *AgentManager* atua com o *AgentMachine* somente quando é necessário, evitando assim que ocorra uma sobrecarga de mensagens na rede (seta pontilhada), conforme Figura 15 e Figura 18.

Os *AgentMachine* se comunicam apenas com os seus respectivos *AgentManager* e não com os demais agentes da comunidade. Os *AgentManager* se comunicam com o mundo externo (*CommonAgents*) e com o(s) seu(s) *AgentMachine*. Não existe troca de mensagens/dados entre os *AgentManager* e entre os *AgentMachine*.

Uma abordagem semelhante a esta é mostrada em RABELO (1997). Nesta, os agentes podem trabalhar numa filosofia de Produção Integrada por Computador – CIM para o escalonamento e supervisão em tempo real do chão de fábrica. O conceito de CIM foca na interconexão de funções técnicas e de gerenciamento do projeto de manufatura. Estas funções devem compartilhar informações preocupadas com o mercado, processos e produtos.

Os agentes das classes *AgentMachine* e *AgentManager* não trocam informações entre si pois eles servem apenas como um "canal de comunicação" usado pelos agentes responsáveis pela tomada de decisão, *CommonAgent*, isto quer dizer que os mecanismos de tomada de decisões estão distribuídos na "camada de aplicação".

O *AgentManager* na Figura 18 representa uma abordagem de "agentificação" (RABELO, 1998 e 2001), onde se coloca uma "camada de software" (*wrapper*) sobre o controlador, ou o seu serviço legado – normalmente não normalizado e fora dos padrões

[46] RCP – *Remote Procedure Call*

internacionais (como os protocolos proprietários) – a fim de virtualizar o respectivo dispositivo físico e seus serviços, que passarão a disponibilizar serviços. A agentificação visa trabalhar com os recursos de manufatura como eles são, e integrá-los de forma transparente em uma arquitetura global.

As classes apresentadas foram assim concebidas por melhor separarem os serviços e as funcionalidades exigidas quando um sistema agentificado quer comunicar com os dispositivos industriais do chão de fábrica. Como já citado no texto, os agentes da classe *AgentMachine* são especialistas no envio e recebimento de comandos e mensagens (integração) e monitoração dos dispositivos industriais. Os agentes da classe *AgentManager* são especialistas em "traduzir" os comando vindos do mundo externo e em controlar o fluxo de mensagens aos *AgentMachine*; também são eles quem fornecem uma "interface"/informações dos dispositivos industriais do chão de fábrica para os agentes da classe *CommonAgent*.

O MMS tem o papel de fazer, através do VMD, com que os muitos dispositivos industriais que não "falam" MMS, mas sim outros protocolos ou protocolos proprietários, tenham uma camada de comunicação comum com o mundo externo. Este protocolo foi escolhido devido às suas características para sistemas de manufatura. Além de possuir um conjunto de comandos que podem ser executados em muitos servidores, o MMS é um padrão ISO. A padronização é muito importante para se obter portabilidade e independência para as camadas de comunicação mais baixas e para os hardwares. Outro fator importante na escolha do MMS é a abstração dos dispositivos industriais; através do VMD é possível acessar as funcionalidades destes dispositivos sem se preocupar como eles foram implementados/desenvolvidos.

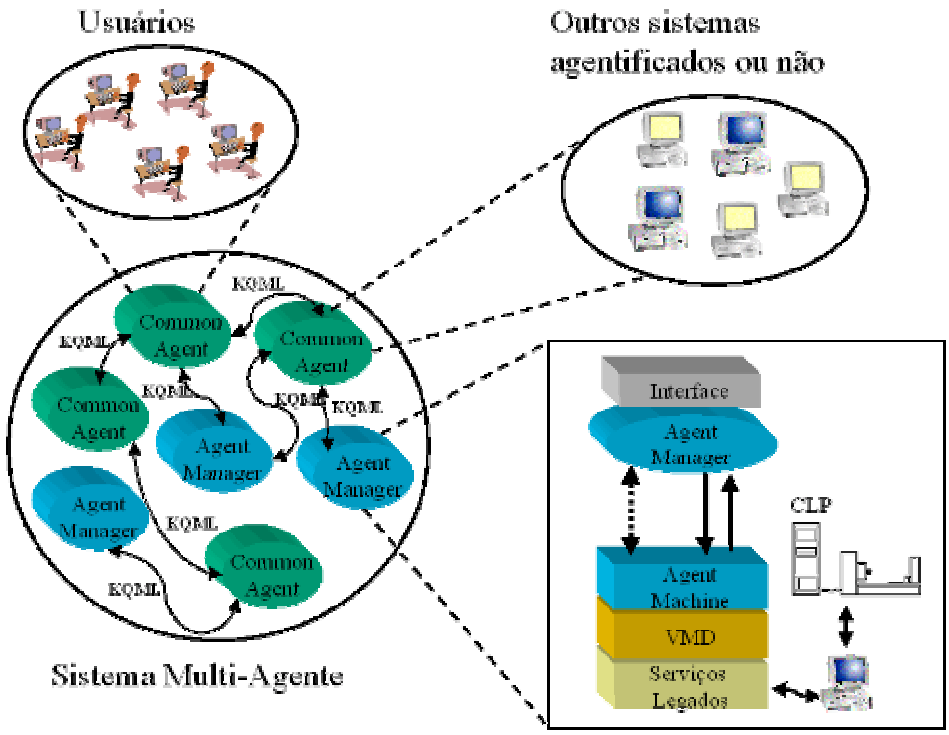


Figura 18 – Interação dos Demais Agentes com o Sistema

O que está sendo proposto é um módulo de comunicação que pode ser "acoplado"/adaptado a sistemas multiagente industriais legados já em funcionamento/operação ou implementado em novos sistemas, a fim de que estes possam ter um nível de harmonização na comunicação com o chão de fábrica. Isso significa que para um sistema multiagente qualquer se comunicar com os dispositivos de chão de fábrica, através desta proposta, seria necessário apenas "conversar" com a classe de agentes *AgentMachine*. A classe *AgentMachine* acaba por encapsular todo o módulo de comunicação com o dispositivo de chão de fábrica tornando a troca de informações transparente. A implementação desse processo e suas considerações são explicadas no capítulo 4.

3.2 Arquitetura dos Agentes

A arquitetura genérica dos agentes é composta por dois módulos, como mostra a Figura 19: o Módulo de Tomada de Decisões e o Módulo de Comunicação.

O módulo de tomada de decisão é o módulo responsável pela "inteligência" dos agentes. Nele são implementados os comportamentos dos agentes, ou seja, as funcionalidades de tomada de decisão. Por exemplo, quais as atitudes e as ações a serem tomadas na chegada de uma mensagem, seja ela de qualquer ator (agente ou não) envolvido no processo.

Este módulo é responsável, por exemplo, por empacotar e desempacotar (interpretador KQML) as mensagens KQML que chegam ou partem "embutidas" em documentos XML ou, no caso dos *AgentMachine*, comandar a troca de mensagens com os dispositivos industriais.

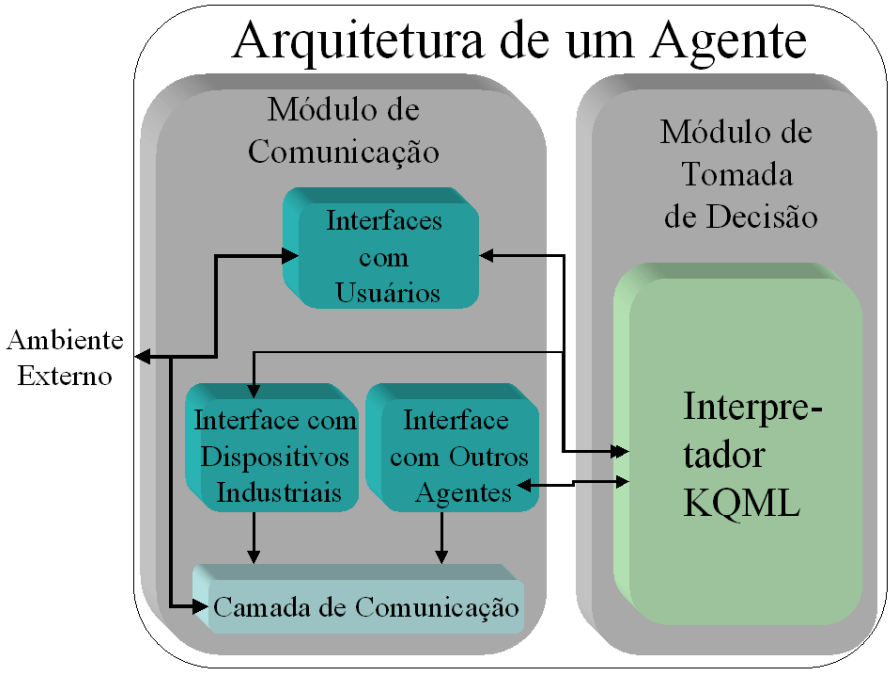


Figura 19 – Arquitetura Genérica do Agente Proposto

O módulo de comunicação trabalha com a interação com os usuários, como interfaces com outros agentes e como interfaces com os dispositivos industriais (quando for o caso). Este módulo permite o agente receber e enviar mensagens de/para outros agentes, expressas em KQML, mensagens dos usuários, e mensagens das máquinas, expressas em MMS. A infraestrutura de comunicação entre os agentes e os dispositivos industriais utilizada é o CORBA.

O interfaceamento com os usuários é feito através de uma interface gráfica, onde um usuário comum pode mandar mensagens para os agentes – estas mensagens podem ser no formato KQML (empacotado XML) ou mensagens em outros formatos quaisquer. O interfaceamento com outros agentes é tem o suporte por uma ontologia que define o vocabulário e os termos para a troca de conhecimento/mensagens entre os agentes. O interfaceamento com os dispositivos industriais e/ou seus sistemas legados é suportado pelo protocolo MMS. Os canais de comunicação são estabelecidos na inicialização do agente e tem como objetivo suportar toda a comunicação com todos os atores envolvidos no processo de comunicação.

A seguir segue a descrição de como os agentes se interrelacionam/trocam informações nos vários protocolos:

Quando um *CommonAgent* quer ordenar o início de um trabalho ou requisitar uma informação de um dispositivo do chão de fábrica ele deve enviar uma mensagem KQML-XML para o *AgentManager,* que representa este dispositivo. O *AgentManager* "interpreta" a performativa e traduz o comando (campo *content* do KQML) que ele recebeu para uma mensagem MMS, então *AgentManager* empacota esta mensagem MMS em uma mensagem KQML e encaminha para o *AgentMachine.* O *AgentMachine* verifica se o dispositivo do chão de fábrica pode atender/executar o comando MMS que chegou. Se o dispositivo industrial pode executar o serviço, ele o faz, assim que este serviço é executado o *AgentMachine* responde ao *AgentManager* com a performativa *Reply* que o serviço foi executado com sucesso. Por outro lado, se não é possível executar o serviço ou ocorreu um erro durante a execução, o *AgentMachine* encaminha ao *AgentManager* uma mensagem KQML com a performativa *Sorry* indicando que o serviço não pode ser executado com sucesso ou que um ocorreu um erro.

Neste capítulo foi apresentado o modelo conceitual da plataforma de comunicações proposta, evidenciando os agentes necessários, os protocolos envolvidos, os seus níveis de utilização e a questão da integração com sistemas legados. No capítulo a seguir é apresentada a componente de implementação da plataforma e dos experimentos de validação realizados.

4. IMPLEMENTAÇÃO E VALIDAÇÃO

Em um sistema de manufatura o gerenciamento da produção envolve todos os elementos associados com o processo de produção, principalmente tarefas e máquinas. Dentre os objetivos de um sistema de manufatura pode-se destacar alocar e supervisionar eficientemente tarefas para as máquinas durante determinados períodos de tempo, procurando satisfazer paralelamente alguns critérios de desempenho e também produzir produtos de qualidade a um custo menor.

4.1 O Dispositivo Industrial

A integração de vários recursos de máquina com protocolos e funcionalidades diferentes é a chave para uma CFM[47], aumentando a flexibilidade da produção e o controle de qualidade (LEITÃO *et al.*, 1996). O maior passo na integração é conectar as máquinas em uma rede de comunicação, permitindo a troca de informações entre as máquinas e os sistemas externos.

A arquitetura proposta por SILVEIRA (1991) de uma célula flexível de manufatura foi adotada como base para a validação do modelo apresentado. Para representar os dispositivos industriais de chão de fábrica foram desenvolvidos softwares que emulam o seu funcionamento. Assim, é possível criar um cenário para simular a troca de mensagens entre os agentes e os dispositivos do chão de fábrica fazendo uso dos protocolos apresentados na proposta. A razão desta adoção e emulação foi a não existência de uma CFM real para a realização de testes. De qualquer forma, os dispositivos industriais são representados/implementados como processos computacionais completamente distribuídos e autônomos, o que dá ao cenário desenvolvido uma situação relativamente realística.

---

[47] CFM – Célula Flexível de Manufatura

A descrição da CFM á apresentada na forma de uma especificação funcional informal, que objetiva um conhecimento básico sobre os dispositivos que compõem a célula e algumas atividades requeridas durante o processo de produção.

A CFM descrita comporta os seguintes elementos:

- Um ou mais centros de usinagem;
- Sistema de transporte e armazenamento de peças (mesa posicionadora);
- Dispositivo de manipulação e armazenamento de ferramentas (robô) e
- Sistema de controle (agentes).

Na verdade para o fim a que se destina, não é relevante conhecer o fluxo completo de materiais/informações dessa CFM. O que se deseja é ter um ambiente onde os equipamentos de manufatura sejam agentificados e possam se comunicar com outras entidades de forma ágil.

A Figura 20 mostra o layout da CFM adotada como modelo.

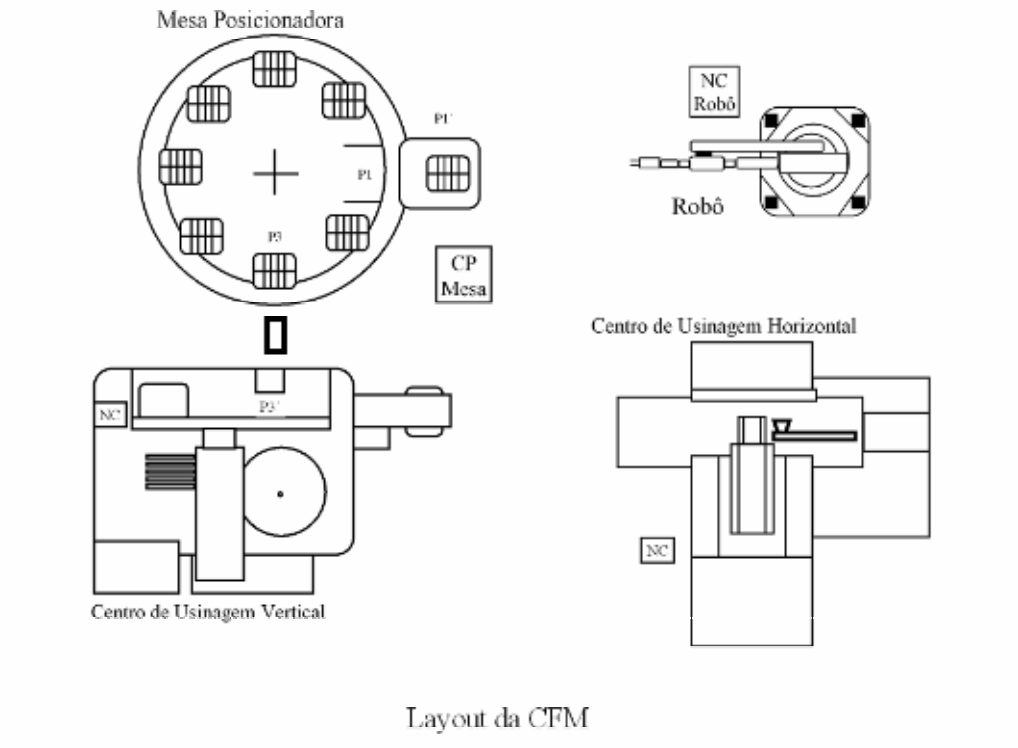


Figura 20 – Layout da Célula Flexível de Manufatura

O padrão MMS é especificado como um protocolo genérico da camada de aplicação destinado à transmissão de mensagens no ambiente industrial, permitindo o acesso aos dados e a execução e controle de uma vasta gama de sistemas e equipamentos no chão de fábrica.

As organizações de padronização especializadas são responsáveis pela elaboração dos padrões contendo detalhes e aspectos específicos das aplicações de cada equipamento (não cobertos pelo padrão genérico). Tais documentos são referidos como *Companion Standards*[48] na ISO-9506 (padrões associados ao MMS) (LEITÃO *et al.* 1996).

A especificação do padrão baseia-se numa representação em objetos, onde os elementos são descritos através de objetos abstratos (objetos MMS), contendo características externamente visíveis (atributos MMS) e operações sobre estes objetos (serviços MMS).

Cabe a cada um dos *Companion Standards* criar serviços específicos e particularizar aqueles definidos pelo padrão genérico quando necessário.

4.2 Os Agentes

Todos os agentes têm a mesma "estrutura", seja qual for a classe a que eles pertencem. A diferença entre os agentes que pertencem à classe *CommonAgent*, *AgentManager* e *AgentMachine* está na implementação dos métodos descritos nas classes. A Figura 21 ilustra o diagrama de classes dos agentes – UML[49] (BOOCH *et al.* 2000) e (LEE, 1997).

[48] *Companion Standards* – padrões contendo os detalhes e aspectos específicos de aplicação para cada equipamento não coberto pelo padrão MMS genérico.
[49] UML – *Unified Modeling Language*

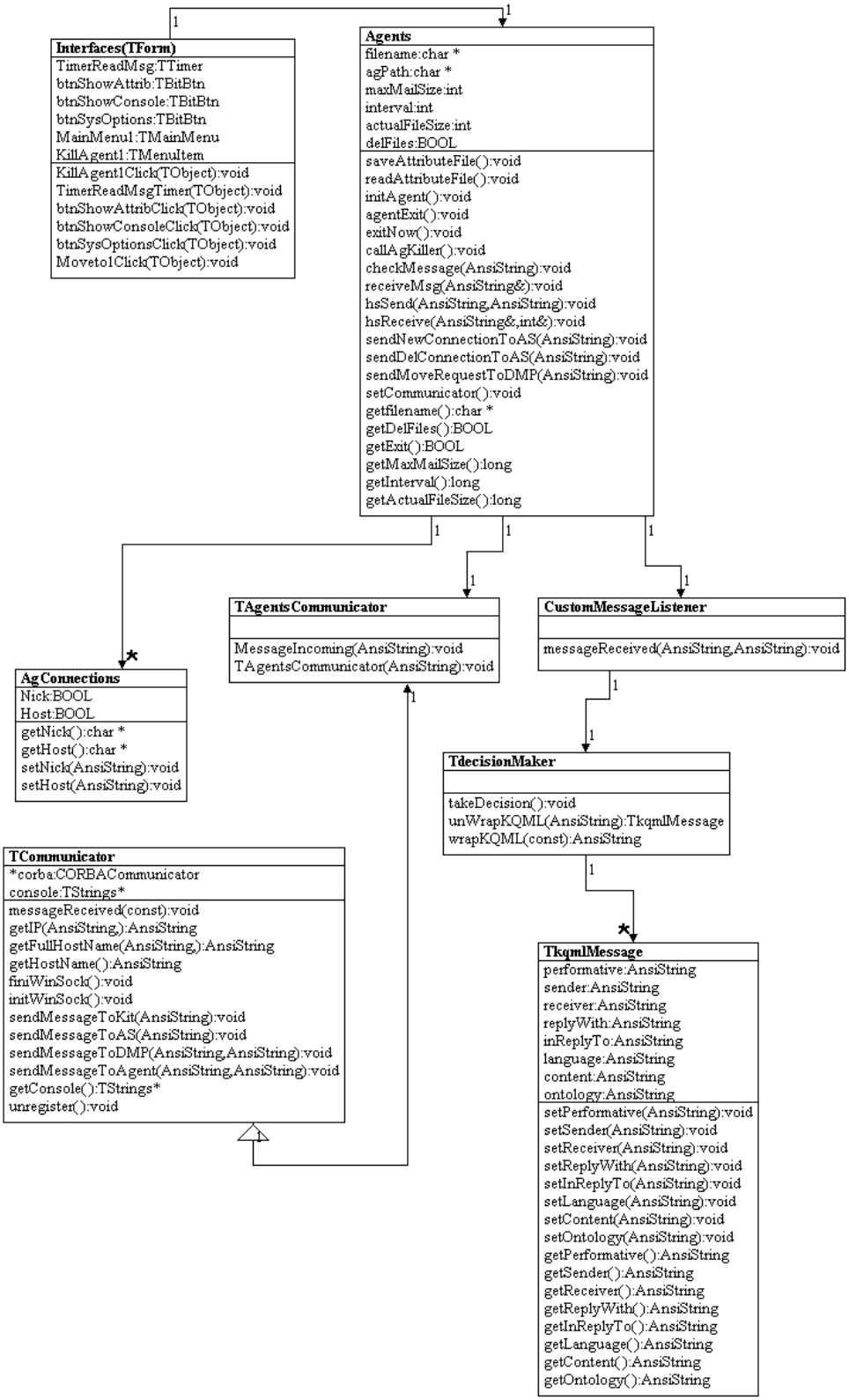


Figura 21 – Diagrama de Classes dos Agentes

A classe *Interface(TForm)* é responsável pela interface dos agentes com os usuários, permitindo aos usuários (pessoas) interagir com o sistema.

A classe *CustomMessageListener* é responsável por receber as mensagens externas (definidas pelos usuários/agentes) que chegam aos agentes. Esta classe que invoca os métodos para a tomada de decisão.

Cada agente possui uma lista de *agConnections* que representa as ligações com outros agentes do sistema, ou seja, esta lista informa quais os agentes é possível a troca de mensagens no sistema multiagente.

A plataforma de comunicação usada no *Massyve Kit 3.0* é implementada na classe *Tcommunicator*, esta classe é responsável pelo suporte/protocolos de comunicação. Os agentes derivam essa classe e implementam os métodos específicos para a sua comunicação na classe *TAgentsCommunicator*.

A classe *Agents* é o "corpo" principal do agente. Nesta classe estão implementadas todas as funções para o funcionamento destes.

Se a classe *Agents* é considerada o "corpo" do agente, a classe *TdecisionMaker* pode ser considerada o "cérebro". A classe *TdecisionMaker* é o local onde é codificada a "inteligência" do agente. Nela são implementas as tomadas de decisões e o agente "sabe" quando e como se comunicar com os demais agentes ou com os dispositivos do chão de fábrica.

Na classe *TkqmlMessage* são implementados os métodos e os atributos utilizados para manipular o objeto *kqmlMessage* dentro do agente (ver seção 4.9).

Quando um sistema multiagente ou um agente qualquer quiser utilizar os benefícios de se comunicar com os agentes que representam os dispositivos de chão de fábrica, ou seja, se comunicar com os agentes da classe *AgentManager*, é necessário incluir as classes *CustomMessageListener*, *TdecisionMaker* e *TkqmlMessage*. Com estas qualquer agente saberá como enviar e receber mensagens aos *AgentManager* sem a necessidade de fazer grandes alterações nos agentes que não pertencem ao contexto industrial.

4.3 Os *AgentMachine*

Os *AgentMachine* são os Agentes responsáveis pela comunicação dos *AgentManager* com o VMD de cada um dos dispositivos industriais citados a seguir, tornando a comunicação no chão de fábrica transparente. A comunicação do *AgentMachine* com o VMD (que pode estar em um sistema legado ou diretamente acoplado ao dispositivo industrial) é feita através das IDLs CORBA que servem como interface de mapeamento entre o agente propriamente dito e o VMD através da arquitetura CORBA.

Um arquivo IDL descreve os tipos de dados, operações e objetos que um cliente pode requisitar e que o servidor pode disponibilizar para a implementação de um dado objeto.

Como exemplo, um arquivo IDL poderia descrever uma interface *VMD* que define os serviços MMS *CreateProgramInvocation, DeleteProgramInvocation, Start, DataExchange* e *Status*. Uma aplicação cliente que faz uso desta IDL estaria capaz de fazer requisições para todas as operações disponibilizadas e especificadas na IDL, e um servidor que poderia satisfazer as requisições de um cliente estaria apto a executar o trabalho associado com as operações descritas na IDL.

4.4 Definição dos VMDs

Apenas as aplicações que possuem caráter servidor (fornecem serviços) necessitam ser modeladas por VMDs. Analisando a descrição da CFM podemos considerar que os equipamentos da célula (centros de usinagem, robô e mesa posicionadora) têm papel de servidores e conseqüentemente serão modelados por um VMD.

A Figura 22 (a), Figura 22 (b) , Figura 22 (c) e , Figura 22 (d) ilustram os VMDs e os Objetos contidos nestes VMDs.

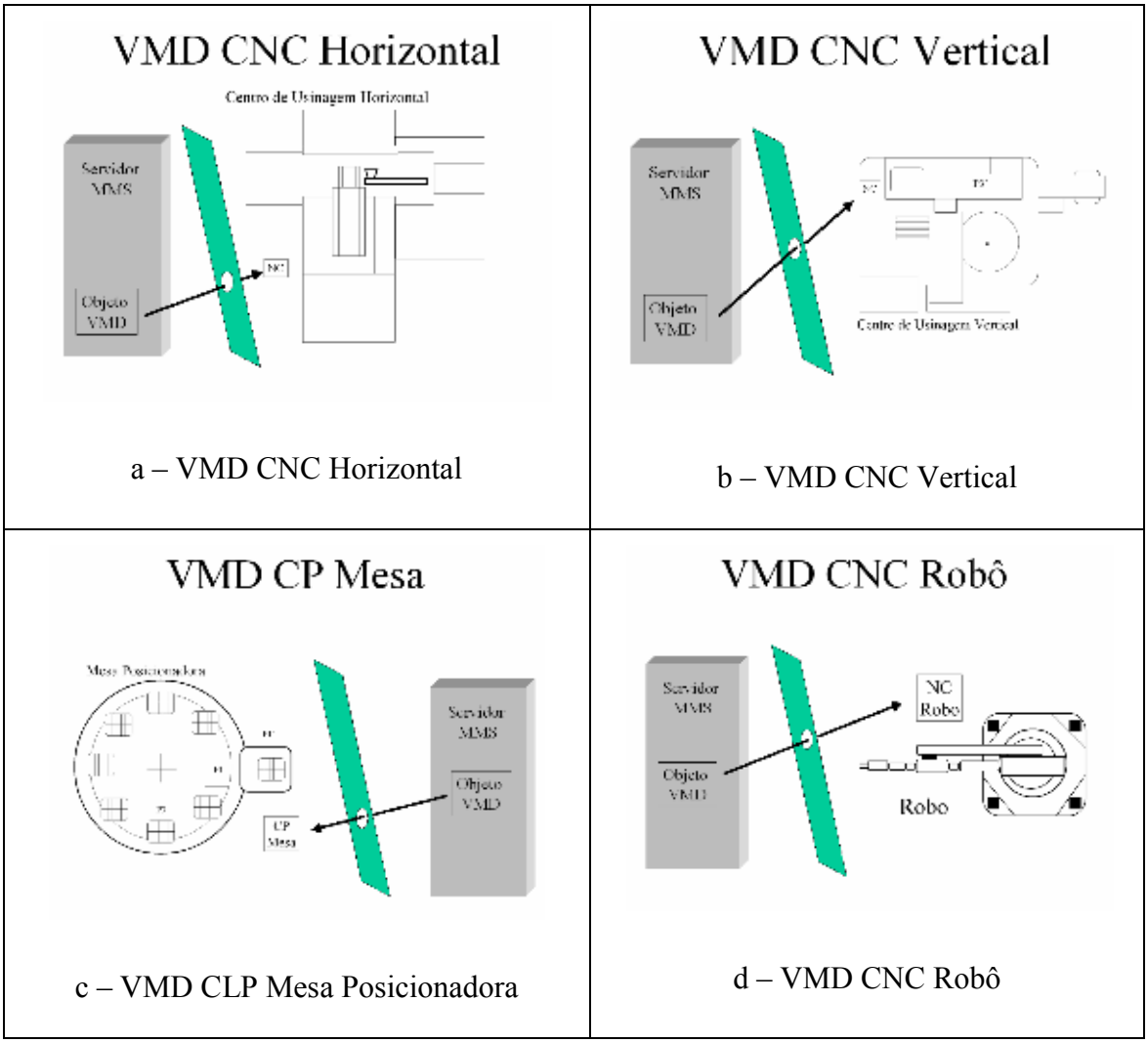


Figura 22 – VMDs

4.5 Centros de Usinagem

A CFM possui dois centros de usinagem, um destinado à usinagem vertical e outro destinado a usinagem horizontal. Cada centro de usinagem é acionado por um CNC que efetua o armazenamento e execução dos programas-peça (que definem o sequenciamento das operações de usinagem sobre uma peça). O CNC também é responsável pelos processos de medição *in-process*, cujo objetivo é detectar erros na usinagem, aferir elementos de corte e detectar erros no posicionamento da peça.

4.5.1 Definição dos Objetos MMS contidos no VMD dos Centros de Usinagem

A modelagem dos Objetos MMS nos VMDs é específico a cada equipamento, e baseia-se nas considerações do Padrão MMS genérico e nas especificações adicionais dos *Companion Standards*.

Neste exemplo, são necessários os serviços de transferência de dados (dados de ferramentas) e de resposta a inspeções pedidas pelos supervisores – *AgentMachine*. Isso permite definir que os CNCs devem pertencer à subclasse *Full Unattended NC* (SILVEIRA, 1991). Nesta classe poderão ser criados e excluídos remotamente (pelo *AgentMachine*) domínios, além de permitir também o controle da execução destes domínios através de condições de eventos.

A seguir são apresentadas as principais considerações para se definir os objetos necessários ao VMD dos centros de usinagem (na classe *Full Unattended NC*).

4.5.2 Objetos de Domínio

O conteúdo de um Objeto de Domínio consiste em programas executáveis, dados e outros objetos subordinados. Quando executados pelo CNC são geradas ações de máquina para executar algum serviço.

Os domínios utilizados aqui são:

- *Machine Program*, que contém uma seqüência de passos de execução. O nome padrão para este tipo de domínio é N_PRG_ seguido pelo nome do programa máquina (definido pelo usuário). Cada CNC da célula conterá ao menos um Domínio deste tipo denominado *N_PRG_x* (onde "x" identifica o tipo da peça a ser usinada pelo programa).
- *Tool Data Table*, o nome padrão é *N_TLD_* seguido pelo nome da tabela de ferramentas necessárias à usinagem de uma peça (definido pelo usuário). Nesta aplicação o Domínio será identificado pelo nome *N_TLD_x* (onde "x" identifica a tabela de ferramentas usadas para fabricação da peça tipo "x"). Este Domínio

servirá para a identificação do conjunto de ferramentas necessárias à usinagem de uma peça e o ajuste da ferramenta.

- *Statical Data*, que contém dados coletados dinamicamente. O nome padrão é *N_SPD_* seguido pelo nome da aquisição definida pelo usuário. Cada CNC conterá um objeto Domínio deste tipo, que será denotado pelo nome *N_SPD_Med* e armazenará as medidas obtidas na medição *in-process*.

Todos os objetos Domínios deste VMD são considerados dinâmicos, devendo ser carregados pelo supervisor em tempo de inicialização ou manutenção de programa-peça.

4.5.3 Objetos de Invocação de Programa

Os objetos Invocação de Programa são usados apenas para controlar remotamente a execução de um programa-peça. Existem, portanto, apenas objetos Invocação de Programa do tipo *Active Program*, que são usados para ativação de domínios do tipo *Machine Program* (SILVEIRA, 1991). O controle de um programa-peça é feito através dos serviços MMS *Start, Stop, Resume, Kill* e *Reset* sobre o objeto Invocação de Programa. O nome padrão para este tipo de objeto é *N_ACT_* seguido pelo nome do programa-peça (*N_ACT_x*).

4.5.4 Objetos *Machine Control*

Este objeto adicionado pelo *Companion Standard* é uma representação abstrata das chaves lógicas providas para permitir ativação remota (*true* ou *false*) de características da máquina. A estrutura de um objeto *Machine Control* deve ser mapeada sobre um objeto Variável Nomeada pré-definido no VMD (SILVEIRA, 1991). São objetos da classe *Machine Control*:

- *N_MachinePower*, definida para ligar/desligar remotamente a alimentação da máquina CNC. Através de um pedido *Write* sobre esta variável MMS, o usuário transfere o estado da máquina CNC de *IDLE* para *READY* ou vice-versa.

- *N_ControlLocal*, ativa e desativa o modo remoto. Através de um pedido de *Write* sobre esta variável MMS, o usuário pode colocar a máquina em modo local ou remoto.

4.5.5 Objeto VMD para os Centros de Usinagem

Finalmente pode-se definir a forma padrão para o objeto VMD dos centros de usinagem. O objeto VMD descreve os atributos disponibilizados pelo servidor (dispositivo industrial), através das mensagens MMS, para o cliente (*AgentMachine*), como por exemplo o fabricante do dispositivo (*Vendor Name*), o modelo do produto (*Model Name*), status lógico (*Logical Status*), status físico do dispositivo (*Physical Status*), os domínios a serem instanciados para operar o dispositivo (*List of Domains*), etc.

```
Object: VMD

     Vendor Name = (específico da implementação)
     Model Name = (específico da implementação)
     Revision = (específico da implementação)
     Logical     Status     =     {STATES_CHANGES_ALLOWED,     NO_
STATES_CHANGES_ALLOWED, LIMITED_SERVICES_SUPPORTED}
     Physical    Status    {OPERATIONAL,    PARTIALLY_OPERATIONAL,
INOPERABLE, NEEDS_COMMISSIONING}
     List of Program Invocations = [N_ACT_x,]
     List of Domains = [N_PRG_x, N_TLD_x, N_SPD_Med]
     List of Machine Controls = [M_MachinePower, N_ControlLocal]
     List of Executable Programs = [N_ACT_x]
```

Tabela 7 – Objeto VMD para o Centro de Usinagem

4.5.6 Inicialização dos Centros de Usinagem

A seguir são descritos os procedimentos básicos, por parte dos supervisores *AgentMachine*, para a inicialização e operação dos centros de usinagem.

4.5.6.1 Inicialização dos CNCs dos Centros de Unsinagem

- Atribuir à variável *N_ControlLocal* o valor *false* (serviço MMS *Write*), para colocar o CNC em modo de operação remoto;

- Atribuir à variável *N_MachinePower* o valor *true* (serviço MMS *Write*), para energizar o equipamento;
- Carregar o Domínio *N_SPD_Med* responsável pelo armazenamento dos dados da medição *in-process*. Estes objetos serão compartilhados por todos os programas-peça. O carregamento destes domínios é feito através dos serviços da seqüência de carregamento (serviços MMS *Initiate_Down_Load_Sequence, Down_Load_Segment* e *Terminate_Down_Load_Sequence*);
- Criar a Invocação de Programa *N_ACT_Prog1* (suposto programa de usinagem) responsável pela inicialização do centro de usinagem (serviço MMS *Create_Program_Invocation*);
- Executar o procedimento *N_ACT_Prog1* (serviço MMS *Sart*) para colocar o Centro de Usinagem em funcionamento.

4.5.6.2 Mensagens MMS Implementadas no VMD dos Centros de Usinagem

Algumas mensagens MMS foram escolhidas, com o caráter ilustrativo, para a implementação dos Centros de Usinagem Horizontal e Vertical que melhor exemplificam a interação com estes dispositivos. Dentre as muitas mensagens MMS existentes, foram escolhidas:

- mm_Write(string VaribaleName, boolean var) – *VariableName* (*N_ControlLocal* ou *N_MachinePower*) representa a variável que receberá o valor booleano contido em *var* (*true* ou *false*) conforme descrito anteriormente;
- mm_Initiate_Down_Load_Sequence(*DomainObject*), mm_Down_Load_Segment(*DomainObject*), mm_Terminate_Down_Load_Sequence(*DomainObject*) – carrega os domínios de objetos (*DomainObject*): *N_SPD_Med* responsável pelo armazenamento dos dados de medição durante o processo, *N_PRG_Proc1* que contém uma seqüência de passos de execução no Centro de Usinagem, *N_TLD_Tool1* responsável por indicar ao dispositivo a tabela de ferramentas necessárias à usinagem de uma peça, ou outros domínios a serem implementados no futuro;

- mm_Create_Program_Invocation(*N_ACT_Prog1*) - cria a invocação do programa de usinagem *N_ACT_Prog1,* definido pelo usuário, responsável pelos procedimentos do Centro de Usinagem;
- mm_Delete_Program_Invocation(*N_ACT_Prog1*) termina a invocação do programa de usinagem *N_ACT_Prog1*;
- mm_Start(*N_ACT_Prog1*) – executa o procedimento *N_ACT_Prog1*;
- mm_Stop(*N_ACT_Prog1*) – termina o procedimento *N_ACT_Prog1*;
- mm_Status() – retorna o status físico do dispositivo industrial.

```
// Mapeamento das mensagens do VMD dos Centros de Usinagem
module Machine{

   exception ProgramInvocation_ex {};
   interface VMD {

      const string ProgramInvocation = "N_ACT_Prog1";
      struct Identify_Response {
            string vendorName;
            string modelName;
            string revision;      };
      boolean Write(in string variable, in boolean state);
      boolean InitiateDownLoadSequence(in string loadSeq)
raises(ProgramInvocation_ex);
      boolean       DownLoadSegment(in       string       loadSeq)
raises(ProgramInvocation_ex);
      boolean    TerminateDownLoadSequence(in    string    loadSeq)
raises(ProgramInvocation_ex);
      boolean       CreateProgramInvocation       (in       string
ProgramInvocation) raises(ProgramInvocation_ex);
      boolean       DeleteProgramInvocation       (in       string
ProgramInvocation) raises(ProgramInvocation_ex);
      boolean Start (in string ProgramInvocation);
      boolean Stop (in string ProgramInvocation);
      string Status ();

      };

};
```

Tabela 8 – Mensagens do VMD dos Centros de Usinagem – Mapeamento na IDL

4.5.7 Mapeamento das Mensagens MMS para IDL utilizada na comunicação dos *AgentMachine* com os Centros de Usinagem:

O mapeamento dos objetos VMD através de IDLs CORBA pode ser feito de forma dinâmica pela herança de um VMD genérico e a partir deste VMD construir/montar o VMD específico ao dispositivo industrial como explorado em (ARIZA *et al.*, 2001a) e (ARIZA, 1998a). Porém, na implementação aqui descrita, cada VMD foi mapeado manualmente pois o VMD padrão proposto em (ARIZA *et al.*, 2001a) e (ARIZA, 1998a) traz inúmeras funcionalidades que não são utilizadas/implementadas neste modelo.

Abaixo é mostrado a IDL responsável por mapear o VMD dos Centros de Usinagem e os principais métodos utilizados na operação destes dispositivos.

A Tabela 9 mostra a relação entre as chamadas de métodos da IDL os serviços MMS correspondentes a estas chamadas:

Métodos da IDL	Mensagens MMS Correspondente
boolean Write(in string variable, in boolean state); "variable" podem ser as variáveis *N_ControlLocal* ou *N_MachinePower* e "state" recebe o valor booleano *true* ou *false*	mm_Write(string VaribaleName, boolean var) *VariableName* corresponde as Variáveis N_ControlLocal(false) e N_MachinePower(true)
boolean InitiateDownLoadSequence (in string loadSeq) raises(ProgramInvocation_ex); "loadSeg" são os objetos de domínio que podem ser carregados: *N_SPD_Med*, *N_PRG_Proc1* e *N_TLD_Tool1*. Toda vez que um erro ocorrer ele será tratado pela exceção gerada e descrita na IDL (*raises(ProgramInvocation_ex)*)	mm_Initiate_Down_Load_Sequence(*DomainObject*) DomainObject corresponde aos Objetos N_SPD_Med, N_PRG_Proc1 e N_TLD_Tool1.
boolean DownLoadSegment (in string loadSeq) raises(ProgramInvocation_ex);	mm_Down_Load_Segment(*DomainObject*)
boolean TerminateDownLoadSequence (in string loadSeq) raises(ProgramInvocation_ex);	mm_Terminate_Down_Load_Sequence(*DomainObject*)
boolean CreateProgramInvocation (in string ProgramInvocation) raises(ProgramInvocation_ex);	mm_Create_Program_Invocation(*N_ACT_Prog1*)

"ProgramInvocation" corresponde a invocação do programa de execução de tarefas - *N_ACT_Prog1* definido previamente pelo usuário.	
boolean DeleteProgramInvocation (in string ProgramInvocation) raises(ProgramInvocation_ex);	mm_Delete_Program_Invocation(*N_ACT_Prog1*)
boolean Start (in string ProgramInvocation);	mm_Start(N_ACT_Prog1)
boolean Stop (in string ProgramInvocation);	mm_Stop(N_ACT_Prog1)
String Status ();	mm_Status()

Tabela 9 – Mapeamento das Mensagens MMS nos métodos da IDL – Centros de Usinagem

4.6 Mesa Posicionadora

Este dispositivo tem como função armazenar as peças que serão usinadas no centro vertical (vindos do armazém de peças brutas ou do centro de usinagem horizontal) e peças que já foram usinadas neste centro (aguardando um processamento no centro de usinagem horizontal, ou colocação no armazém de peças prontas).

A mesa é acionada por um Controlador Lógico Programável (CLP), que executa os procedimentos de movimentação da mesa, que deverá ser ativado pelo *AgentMachine*.

4.6.1 Definição dos Objetos MMS contidos no VMD da Mesa Posicionadora

As operações da mesa devem ser efetuadas a partir da chamada de procedimentos remotos (classe *Interlocked Control* definido no *Companion Standard* para CLPs (SILVEIRA, 1991).

Nesta configuração, um cliente pede ao servidor a execução de uma operação de aplicação, que após executá-la, informa ao cliente o resultado da operação.

Há dois aspectos importantes quanto ao uso da classe *Iterlocked Control*; a sincronização dos programas de aplicação cliente e servidor, e a troca de dados entre eles

(que ocorre no ponto de sincronização). O *Companion Standard* para CLPs define o objeto adicional *Gate* e o serviço *Data_Exchange* para tratar estes aspectos.

4.6.2 Objetos Domínio e Invocação de Programa

As funções básicas de movimentação da mesa estarão estaticamente alocadas na memória do controlador programável, e serão mapeadas em um objeto Domínio estático. Desta forma, o supervisor ativará a mesa através de um determinado objeto Invocação de Programa. O fato do objeto Domínio estar alocado estaticamente, se deve à característica dedicada de suas funções, que não necessitam ser alteradas durante o processo de produção.

4.6.3 Objeto *Gate*

Cada operação básica da mesa será representada por um objeto *Gate* que por sua vez implementará as chamadas de procedimento remoto. Cada um dos comandos da mesa será considerado uma subrotina do Domínio *CP_MESA*.

4.6.4 Objeto VMD para a Mesa Posicionadora

O objeto VMD descreve os atributos disponibilizados pelo servidor (dispositivo industrial), através das mensagens MMS, para o cliente (*AgentMachine*), como por exemplo o fabricante do dispositivo (*vendor name*), status físico do dispositivo (*Physical Status*), os domínios a serem instanciados para operar o dispositivo (*List of Domains*), etc.

A Tabela 10 mostra o Objeto VMD da Mesa posicionadora:

```
Object: VMD
      Vendor Name = (específico da implementação)
      Model Name = (específico da implementação)
      Revision = (específico da implementação)
      Logical Status = {STATES_CHANGES_ALLOWED,
                        NO_ STATES_CHANGES_ALLOWED,
                        LIMITED_SERVICES_SUPPORTED}
      Physical Status = {OPERATIONAL,
                         PARTIALLY_OPERATIONAL,
                         INOPERABLE,
                         NEEDS_COMMISSIONING}
      List of Program Invocations = [CP_MESA]
      List of Domains = [CP_MESA]
```

Tabela 10 – Objeto VMD para a Mesa Posicionadora

4.6.5 Inicialização da Mesa Posicionadora

A seguir são descritos os procedimentos básicos, por parte do supervisor *AgentMachine*, para a inicialização e operação da mesa operadora.

4.6.5.1 Inicialização do CLP da Mesa Posicionadora

- Criar a invocação do programa *CP_MESA* (serviço *Create_Program_Invocation*) responsável pela inicialização da mesa posicionadora.
- Executar o procedimento *CP_MESA* (serviço *Start*) para colocar a mesa posicionadora em operação (esperando a chamadas remotas de procedimentos).
- Pedido para a mesa rotacionar para uma posição qualquer (serviço *Data_Exchange*).

4.6.5.2 Mensagens MMS Implementadas no VMD da Mesa Posicionadora

Algumas mensagens MMS foram escolhidas, com o caráter ilustrativo, para a implementação da Mesa Posicionadora que melhor exemplificam a interação com ela. Dentre as muitas mensagens MMS existentes, foram escolhidas:

- mm_Create_Program_Invocation (*CP_MESA*) – cria uma invocação do programa CP_MESA responsável pela inicialização da mesa posicionadora;
- mm_Delete_Program_Invocation (*CP_MESA*) – termina a invocação do programa CP_MESA;
- mm_Start(*CP_MESA*) – executa o procedimento CP_MESA para colocar a mesa posicionadora em operação;
- mm_Data_Exchange (int pos) – rotaciona a mesa para a posição onde o *pallet* será manipulado, pelo robô ou pelo centro de usinagem vertical;
- mm_Status() – retorna o status físico do dispositivo industrial.

4.6.6 Mapeamento das Mensagens MMS para IDL utilizada na comunicação do Agent Machine com a Mesa Poscionadora

A Tabela 11 mostra o mapeamento das mensagens MMS através das IDLs utilizadas pelo *AgentMachine* responsável por comunicar com os serviços legados da mesa posicionadora através dos VMD.

```
// Mapeamento das mensagens do VMD da Mesa Posicionadora

module Machine{
   exception ProgramInvocation_ex {};

   interface VMD {

      const string ProgramInvocation = "CP_MESA";
      struct Identify_Response {
            string vendorName;
            string modelName;
            string revision;
      };

      boolean       CreateProgramInvocation       (in       string
ProgramInvocation) raises(ProgramInvocation_ex);
      boolean       DeleteProgramInvocation       (in       string
ProgramInvocation) raises(ProgramInvocation_ex);
      boolean Start (in string ProgramInvocation);
      boolean DataExchange (in long pos);
      string Status ();
      };
};
```

Tabela 11 – Mensagens do VMD da Mesa Posicionadora – Mapeamento na IDL

Cada serviço descrito na IDL CORBA corresponde diretamente a uma mensagem MMS que irá acessar o VMD; por exemplo, toda vez que o *AgentMachine* (no papel de cliente) for criar a invocação do programa CP_MESA (responsável por implementar as chamadas de procedimentos remotos) da Mesa Posicionadora, o serviço *boolean CreateProgramInvocation()* na IDL será o responsável por chamar a mensagem *mm_Create_Program_Invocation()* no objeto VMD (servidor) como é mostrado abaixo.

Métodos da IDL	Mensagens MMS Correspondente
boolean CreateProgramInvocation (in string ProgramInvocation) raises(ProgramInvocation_ex); "ProgramInvocation" corresponde a invocação do programa de execução da mesa operadora – *CP_MESA*	mm_Create_Program_Invocation(*CP_MESA*)
boolean DeleteProgramInvocation (in string ProgramInvocation) raises(ProgramInvocation_ex);	mm_Delete_Program_Invocation(*N_ACT_Prog1*)
boolean Start (in string ProgramInvocation);	mm_Start(N_ACT_Prog1)
boolean Stop (in string ProgramInvocation);	mm_Stop(N_ACT_Prog1)
String Status ();	mm_Status()
boolean DataExchange (in integer pos) "pos" corresponde ao grau de rotação que a mesa de girar	mm_Data_Exchange (int pos)

Tabela 12 – Mapeamento das Mensagens MMS nos métodos da IDL – Mesa Posicionadora

4.7 Robô

A função do Robô é transportar peças entre os vários pontos da célula. Ele é acionado por um Comando Numérico, que armazenará um programa de transporte de peças.

O *AgentMachine* do robô comandará o robô através de uma chamada de procedimento de transporte, descrita no VMD na lista de programas de invocação, que conterá os parâmetros necessários para definição da trajetória e do tipo de peça a

transportar, possibilitando proceder a manipulação de acordo com as características mecânicas de cada peça.

4.7.1 Objetos de Domínio

Existe neste VMD a necessidade de quatro objetos Domínio, responsáveis pelo modelo das características físicas e operacionais do robô.

O primeiro representa o recurso braço do robô e é padronizado com o nome *R_ARM* (*Robot Arm*). Este recurso é composto pela garra de manipulação, pelos servomecanismos (um para cada grau de liberdade) e pelo sistema de planejamento de rota, responsável pela conversão de uma trajetória desejada em comandos para os servomecanismos. Cada um destes componentes adiciona um número de atributos específicos ao Domínio *R_ARM* de forma a modelar por completo a cinemática e o controle do robô.

Os demais Domínios modelam os subsistemas que não fazem parte do braço do robô e são chamados pelo *Companion Standard* de objetos *Auxiliary Devices* (todos mapeados em objetos Domínio). São eles os Domínios *R_SAFE* (*Robot Safeguard*), associado aos dispositivos de proteção. *R_CAL* (*Robot Calibration*), que contém os procedimentos de calibração das diversas junções do braço do robô; e o último *R_TRANS*, movimento do robô na célula.

4.7.2 Objetos Invocação de Programa

O VMD do Robô necessita de três objetos Invocação de Programa, o primeiro está permanentemente associado ao Domínio *R_ARM* e tem por finalidade o controle direto do braço do robô. Outro objeto Invocação de Programa, está associado ao Domínio *R_CAL* e controla a execução do procedimento de calibração das junções. A última Invocação de Programa deve ser carregada dinamicamente pelo usuário, para controlar a execução dos procedimentos de movimentação. Ela está associada não somente ao Domínio *R_TRANS* mas também ao Domínio *R_SAFE*, garantindo assim a segurança do equipamento durante um procedimento de movimentação.

4.7.3 Objeto VMD para o Robô

```
Object: VMD
      Vendor Name = (específico da implementação)
      Model Name = (específico da implementação)
      Revision = (específico da implementação)
      Logical Status = {STATES_CHANGES_ALLOWED,
                        NO_STATES_CHANGES_ALLOWED,
                        LIMITED_SERVICES_SUPPORTED}
      List of Capabilities = (específico da implementação)
      Physical Status {OPERATIONAL,
                       PARTIALLY_OPERATIONAL,
                       INOPERABLE,
                          NEEDS_COMMISSIONING}
      List of Program Invocations = [TRANS, R_ARM, R_CAL]
      List of Domains = [TRANS, R_ARM, R_CAL, R_SAFE]
      Robot VMD State = {ROBOT_IDLE, ROBOT_EXECUTING, ROBOT_READY
                         ROBOT_LOADED, ROBOT_PAUSED,
                         MANUAL_INVERVENTION_REQUIRED }
```

Tabela 13 – Objeto VMD para o Robô

4.7.4 Inicialização do Robô

A seguir são descritos os procedimentos básicos, por parte do supervisor *AgentMachine*, para a inicialização e operação do robô.

4.7.4.1 Inicialização do CNC do Robô

- Ativar o procedimento de auto-calibração usando o serviço *Start* sobre o objeto Invocação de Programa *R_CAL*.
- Carregar o Domínio contendo o código do programa de transporte *TRANS*. O carregamento destes domínios é feito através dos serviços da seqüência de carregamento (*Initiate_Down_Load_Sequence, Down_Load_Segment* e *Terminate_Down_Load_Sequence*).
- Criar a Invocação de Programa *TRANS* (serviço *Create_Program_Invocation*).
- Relacionar a Invocação de Programa *TRANS* com a Invocação de Programa *R_ARM* para permitir que o procedimento de transporte tenha acesso às

rotinas de movimentação do robô. Esta relação é conseguida através do serviço *Select*.

4.7.4.2 Mensagens MMS Implementadas no VMD do Robô

Algumas mensagens MMS foram escolhidas, com o caráter ilustrativo, para a implementação do Robô, que melhor exemplificam a troca de mensagens entre um supervisor e o robô propriamente dito.

- mm_Start() – ativa o processo de auto-calibração sobre o objeto Invocação de Programa R_CAL;
- mm_Initiate_Down_Load_Sequence(*TRANS*), mm_Down_Load_Segment(*TRANS*), mm_Terminate_Down_Load_Sequence(*TRANS*) – carrega o domínio *TRANS* responsável pelo movimento do robô na célula.
- mm_Create_Program_Invocation(*TRANS*) – a Invocação de Programa *R_ARM* para permitir que o procedimento de transporte tenha acesso às rotinas de movimentação do robô
- mm_Select(*R_ARM*) – relaciona a invocação do programa de transporte *TRANS* com o procedimento de transporte *R_ARM* para a movimentação do robô.

4.7.5 Mapeamento das Mensagens MMS para IDL utilizada na comunicação do *AgentMachine* com o Robô

```
// Mapeamento das mensagens do VMD do Robô
module Machine{
exception ProgramInvocation_ex {};
   interface VMD {
      const string ProgramInvocation1 = "TRANS";
      const string ProgramInvocation2 = "R_ARM";
      const string ProgramInvocation3 = "R_CAL";
      struct Identify_Response {
            string vendorName;
            string modelName;
            string revision;
      };
      boolean          InitiateDownLoadSequence(in          string
ProgramInvocation1) raises(ProgramInvocation_ex);
```

```
      boolean     LoadSegment(in     string     ProgramInvocation1)
raises(ProgramInvocation_ex);
      boolean          TerminateDownLoadSequence(in          string
ProgramInvocation1) raises(ProgramInvocation_ex);
      boolean       CreateProgramInvocation       (in       string
ProgramInvocation1) raises(ProgramInvocation_ex);
      boolean       DeleteProgramInvocation       (in       string
ProgramInvocation1) raises(ProgramInvocation_ex);
      boolean Start (in string ProgramInvocation3);
      boolean Select(in string ProgramInvocation2)
      string Status ();
      };
};
```

Tabela 14 – Mensagens do VMD do Robô – Mapeamento na IDL

A tabela abaixo mostra a relação entre as chamadas de métodos da IDL os serviços MMS correspondentes a estas chamadas:

Métodos da IDL	Mensagens MMS Correspondente
boolean InitiateDownLoadSequence (in string *ProgramInvocation1*) raises(ProgramInvocation_ex); "ProgramInvocation1" é o objeto *TRANS* responsável pelo mecanismo de transporte	mm_Initiate_Down_Load_Sequence(*TRANS*)
boolean DownLoadSegment (in string *ProgramInvocation1*) raises(ProgramInvocation_ex);	mm_Down_Load_Segment(*TRANS*)
boolean TerminateDownLoadSequence (in string *ProgramInvocation1*) raises(ProgramInvocation_ex);	mm_Terminate_Down_Load_Sequence(*TRANS*)
boolean CreateProgramInvocation (in string ProgramInvocation1) raises(ProgramInvocation_ex);	mm_Create_Program_Invocation (*TRANS*)
boolean DeleteProgramInvocation (in string ProgramInvocation1) raises(ProgramInvocation_ex);	mm_Delete_Program_Invocation(*TRANS*)
boolean Start (in string ProgramInvocation3);	mm_Start(*R_CAL*)
boolean Select (in string ProgramInvocation2);	mm_Stop(*R_ARM*)
String Status ();	mm_Status()

Tabela 15 – Mapeamento das Mensagens MMS nos métodos da IDL – Robô

4.8 Mensagens KQML trocadas

Os *AgentManager* são os agentes responsáveis pelo processo de tomada de decisão e comunicação com o mundo exterior, tornando os *AgentMachines* desconhecidos para os demais agentes e sistemas. Para que esta transparência exista é necessária a troca de informações entre estes dois tipos de agentes. A troca de mensagens é feita através da linguagem KQML.

A seguir são mostradas as mensagens KQML trocadas entre o *AgentManager* e o *AgentMachines* dos dispositivos industriais.

4.8.1 Mensagens KQML trocadas entre os *AgentManager* e os *AgentMachine*

Nesta implementação foram usadas apenas as performativas *Ask-if*, *Tell*, *Reply* e *Sorry*. As demais performativas não são usadas pois as mensagens trocadas entre o *AgentManager* e o *AgentMachine* são apenas de caráter informativo – performativa *Tell* que é usada para comunicar uma ação ou como informações genéricas, como por exemplo a mudança do posicionamento da mesa através do comando *DataExchange(Pos)* – ou para pesquisas básicas – performativa *Ask-if* que é usada para requisitar o estado físico da pesa posicionadora através do comando *Status()* – performativas *Reply* e *Sorry* são usadas para respostas simples (resultado de uma consulta) ou informar um erro ocorrido com alguma requisição.

O conjunto de performativas é extensível. Existe um conjunto de performativas reservadas que tem um significado bem definido. Para dois ou mais agentes se comunicarem, não é necessário um conjunto mínimo de performativas (UNIVERSITY OF MARYLAND, 1994).

Quando o *AgentManager* quer saber o estado físico do dispositivo físico (por exemplo do Centro de Usinagem) ele "pede" ao *AgentMachine* esta informação através da mensagem MMS *Status()* usando a performativa *ASK-IF* (Tabela 16).

```
Performativa/Código
(ASK-IF
  :sender AgentManager
  :receiver AgentMachine
  :reply-with AgentManager
  :in-reply-to NONE
  :language MMS
  :content Status ()
  :ontology NONE )
```

Tabela 16 – ASK-IF

Sempre que o *AgentManager* quer comunicar ao *AgentMachine* uma determinada ação, ele usa a performativa *TELL* contendo a mensagem MMS equivalente ao seu "desejo" no campo *content*, por exemplo, a mensagens MMS *Start()* (Tabela 17) diz para o *AgentMachine* iniciar o processo de usinagem de uma peça.

```
Performativa/Código
(TELL
  :sender AgentManager
  :receiver AgentMachine
  :reply-with AgentManager
  :in-reply-to NONE
  :language MMS
  :content Start()
  :ontology NONE )
```

Tabela 17 – TELL

Para responder as questões feitas pelo *AgentManager* através da performativa *ASK-IF* o *AgentMachine* usa a performativa *REPLY* para indicar a resposta. A Tabela 18 mostra a resposta de uma performativa *ASK-IF* perguntando sobre o status físico do Centro de Usinagem, o campo *content* traz a resposta da requisição.

```
Performativa/Código
(REPLY
  :sender AgentMachine
  :receiver AgentManager
  :reply-with NONE
  :in-reply-to AgentManagerUsinagem – ASK-IF
  :language MMS
  :content OPERATIONAL
  :ontology NONE )
```

Tabela 18 – REPLY

Toda vez que o *AgentMachine* não pode executar um comando enviado pelo *AgentManager* ele informa ao *AgentManager* que o houve problema ao executar o comando MMS referido usando a performativa *SORRY* (Tabela 19).

Performativa/Código
(SORRY :sender AgentMachine :receiver AgentManager :reply-with NONE :in-reply-to AgentManager :language MMS :content Start() :ontology NONE)

Tabela 19 – SORRY

4.9 Mensagens KQML empacotadas em XML

As mensagens KQML trocadas entre os agentes aqui propostos estão encapsuladas, ou melhor, empacotadas em XML. As mensagens KQML em XML são previamente definidas em um DTD.

A opção, de adotar o formato KQML como um padrão para as mensagens entre os agentes, foi tomada para manter um formato comum nas mensagens trocadas e pela maneira clara de como são especificados os dados de "alto nível". O uso das características do KQML nas mensagens (documentos XML) ajuda a identificar qual ação/atitude está sendo tomada/comunicada pelo agente e ainda trazer detalhes as ações e atitudes.

Outra vantagem de "empacotar mensagens" no formato KQML em XML é a facilidade que o XML traz às aplicações. Os agentes não geram documentos XML que não sejam válidos e bem formados (seção 2.1.7). Ainda, o XML permite qualquer aplicação que saiba "ler" um documento XML interpretar os campos contidos na mensagem.

O XML atua como uma "linguagem universal" eliminando as restrições, como por exemplo, as incompatibilidades entre os diversos sistemas (agentificados ou não)

existentes. Basta saber qual DTD usar para formatar ou interpretar a mensagem e o seu conteúdo.

Para as mensagens trocadas entre os agentes, apenas um único DTD é necessário, todas as mensagens KQML têm o mesmo formato e devem conter sempre os mesmos campos, assim não se faz necessário mais de um DTD. Este DTD especifica todos os campos KQML, e possibilita o uso facultativo de alguns dos campos que não são fundamentais para a compreensão das mensagens. Todas as mensagens "KQML" empacotadas em XML devem obrigatoriamente usar este DTD. A adoção de apenas um DTD traz a vantagem de que todas as mensagens obedecem a um padrão único e os agentes necessitam tratar diferentes tipos de mensagens, mas sim, apenas o seu conteúdo.

A Tabela 20 mostra o DTD padrão das mensagens KQML.

```
<!—DTD KQML DTD padrao para as mensagens KQML -->

<!ELEMENT KQML (Performative)>
     <!ELEMENT Performative (sender, receiver, reply-with, in-
reply-to, language, content, ontology, from?, to?, force?)>
     <!ATTLIST Performative name CDATA #REQUIRED>
          <!ELEMENT sender (#PCDATA)>
          <!ELEMENT receiver (#PCDATA)>
          <!ELEMENT reply-with (#PCDATA)>
          <!ELEMENT in-reply-to (#PCDATA)>
          <!ELEMENT language (#PCDATA)>
          <!ELEMENT content (#PCDATA)>
          <!ELEMENT ontology (#PCDATA)>
          <!ELEMENT from (#PCDATA)>
          <!ELEMENT to (#PCDATA)>
          <!ELEMENT force (#PCDATA)>
```

Tabela 20 – DTD Padrão para as Mensagens KQML

Neste DTD é definido o elemento *Performative* que contém o atributo *name* correspondente ao tipo da mensagem KQML que está sendo enviada/recebida. Este elemento tem uma lista de "sub-elementos" ou elementos filhos. Os elementos filhos *sender, receiver, reply-with, in-reply-to, language*, etc., são os campos da mensagem KQML.

Toda vez que uma mensagem KQML empacotada no XML chega, o agente que está recebendo a mensagem verifica se ela está de acordo com o DTD a ela associado,

como já citado anteriormente. Caso a mensagem recebida não esteja de acordo com o DTD, o agente receptor da mensagem "errada" responde ao emissor que a sua mensagem é inválida e deve seguir o padrão estabelecido no DTD. Porém, se o KQML empacotado no XML recebido está de acordo com o DTD, o agente receptor irá desempacotar o KQML do XML (método *unwrapKQML*) e guardar os campos KQML dentro de um objeto chamado *kqmlMessage*, que representa uma mensagem KQML. Uma vez as informações dentro do objeto *kqmlMessage*, o agente pode trabalhar com as informações contidas no KQML da forma que bem entender.

Quando um agente quer enviar uma mensagem KQML, este agente "carrega" as informações do KQML no objeto *kqmlMessage* e empacota o KQML em XML (método *wrapKQML*), transformando as mensagens KQML em XML para, somente depois, enviar as mensagens para o destinatário.

A Figura 23 ilustra uma mensagem KQML empacotada em XML.

```
<KQML>
<Performative name="ASK-IF" >
    <sender>AgentManagerMesa</sender>
    <receiver> AgentMachineMesa</receiver>
    <reply-with> AgentManagerMesa</reply-with>
    <in-reply-to>NONE</in-reply-to>
    <language>MMS</language>
    <content>Start()</content>
    <ontology>NONE</ontology>
</Performative>
</KQML>
```

Figura 23 – Mensagem KQML representada em XML

4.9.1 Método *wrapKQML*

O método *wrapKQML*, contido na classe *UdecisionMaker*, tem como objetivo empacotar o objeto *kqmlMessage*, que contém todos os campos de uma mensagem KQML, em um novo documento XML. Este método deve "traduzir" os atributos do objeto *kqmlMessage* em elementos do DOM[50] (MORRISON, 2000), que posteriormente serão descritos no formato de uma mensagem XML com todas as suas propriedades, tais como a versão do XML, o *Document Type Definition* (local que indica qual DTD é pertinente à esta mensagem XML) no elemento raiz do XML, etc.

Para "traduzir" os elementos do objeto *kqmlMessage* em elementos do DOM são usados o métodos *CreateXML* e *CreateDOM*, da classe *UxmlProcessor*. O método *CreateXML* recebe o objeto *kqmlMessage*, cria o objeto DOM (através de um componente específico para trabalhar com o XML) e chama o método *CreateDOM*. O método *CreateDOM* tem como função associar qual é o elemento raiz e qual é o DTD que está associado a mensagem XML. Ele também é responsável por carregar os campos do objeto *kqmlMessage* nos nós da árvore DOM. Após processar o método *CreateDOM*, o objeto DOM é "gravado" e transcrito para uma *string*. Por fim, ele retorna a *string* com o novo documento XML contento uma mensagem KQML.

4.9.2 Método *unWrapKQML*

O método *unwrapKQML*, contido na classe *UdecisionMaker*, tem como objetivo desempacotar o XML que contém as mensagens KQML trocadas entre os agentes.

Ao receber uma mensagem XML o método *unWrapKQML* processa a nova mensagem com o auxílio dos métodos *ProcessXML* e *ProcessDOM* contidos na classe *UxmlProcessor*. O método *ProcessXML* recebe uma *string* que contém uma mensagem XML, faz a validação do novo documento para saber se o XML recebido é válido (verificar se o documento XML está de acordo com o DTD do XML), e sendo válido o

[50] DOM – *Document Object Model* – árvore que contém os nós descritos no DTD

XML será invocado o método *ProcessDOM*. O método *ProcessDOM* tem como objetivo percorrer os nós da árvore DOM para obter os valores contidos nos *tags* do XML, que correspondem aos atributos do objeto *kqmlMessage*, e carregar estes atributos no objeto *kqmlMessage*. Feito isto, ele retorna o objeto *kqmlMessage*.

A Figura 24 ilustra o objeto DOM que correspondente a esta mensagem XML.

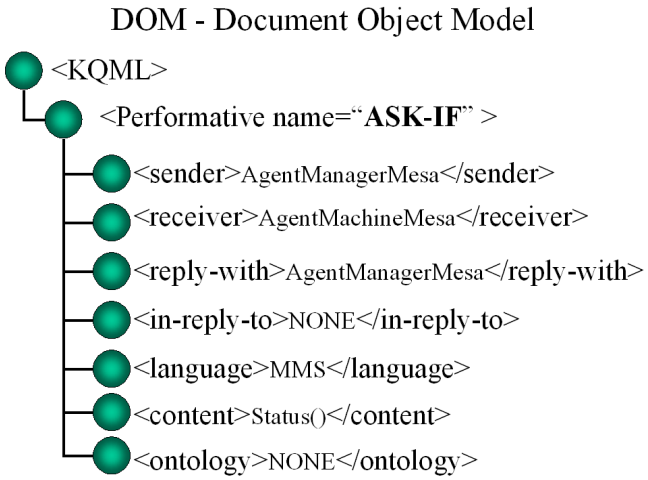


Figura 24 – Document Object Model

4.10 Demais Mensagens KQML entre os *CommonAgent* e os *AgentManager*

As demais mensagens KQML trocadas entre os *CommonAgent* e o *AgentManager* são diferentes das mensagens trocadas entre os *AgentManager* e os *AgentMachine* apenas no seu conteúdo. Isto porque o campo *content* das mensagens trocadas entre os *CommonAgent* e os *AgentMachine* contém as mensagens MMS propriamente ditas. Já o campo *content* das mensagens trocadas entre os *CommonAgent* e os *AgentManager* traz no seu conteúdo comandos que serão traduzidos pelos *AgentManager* desta linguagem para mensagens MMS, e vice-versa.

A Figura 25 ilustra como será a troca de mensagens entre os agentes.

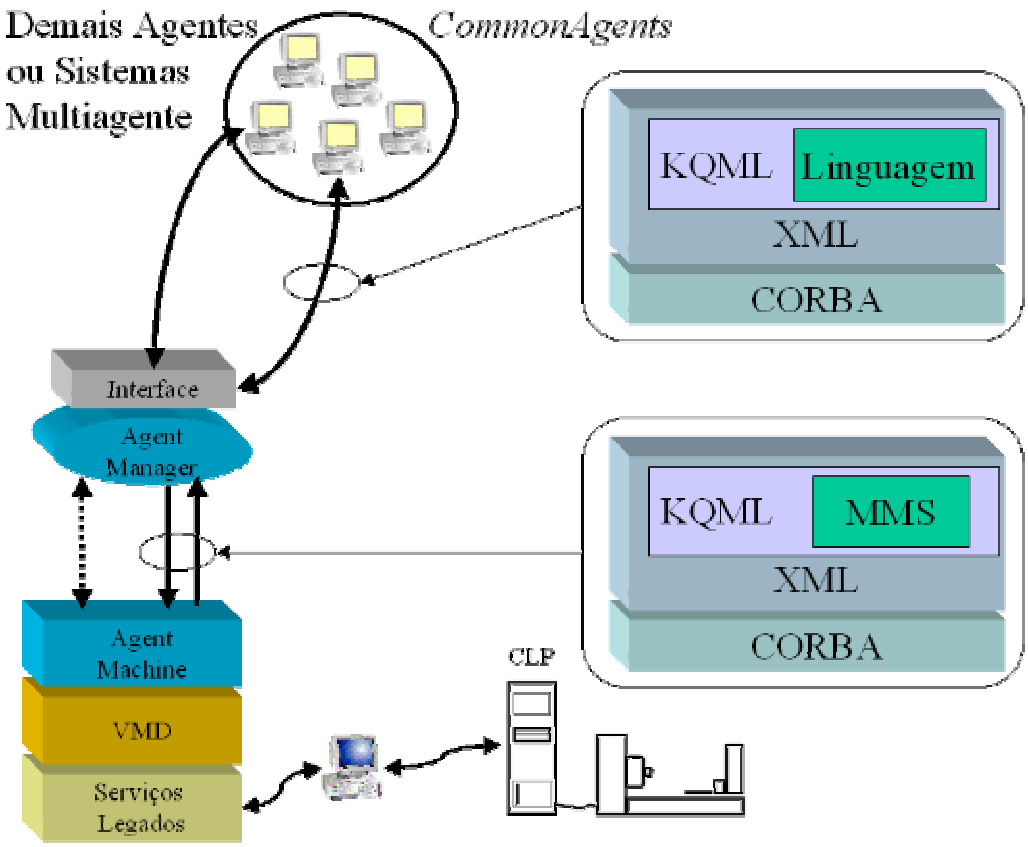


Figura 25 – Troca de Mensagens Entre os Agentes

O *AgentManager* é responsável por "traduzir" as mensagens recebidas de "fora" do contexto industrial que chegam até ele numa linguagem "comum" para uma forma compreensível dos dispositivos industriais, ou seja, para MMS. Esta "tradução" é possível pois as mensagens que chegam dos *CommonAgents* seguem uma ontologia que especifica como "montar" as mensagens de uma forma padrão. As mensagens "comuns" trocadas entre os *CommonAgent* e os *AgentManager* tem como o objetivo a troca de informações de "alto nível" entre os agentes para a coordenação da CFM, como por exemplo: identificar qual o dispositivo o *AgentManager* está representando, ordenar o início do processo de produção e obter os resultados quando o dispositivo termina o seu processamento.

Um aspecto importante na interação entre agentes está relacionado com o conhecimento comum que os agentes possuem sobre os objetivos relativos a um determinado domínio e referidos na linguagem de conteúdo. Para os *CommonAgent*

estabelecerem a comunicação com os *AgentManager*, os *CommonAgent* é importante um entendimento adequado na troca de mensagens, para isso eles devem saber qual é o vocabulário usado, quais os "comandos" são aceitos e permitidos, e os conceitos envolvidos. Este aspecto cria naturalmente espaço para adoção de ontologias.

A ontologia usada deve proporcionar os conceitos e as principais relações envolvidas no domínio de chão de fábrica sem ambigüidades para prover respostas das perguntas sobre os dispositivos de chão de fábrica do tipo: Qual dispositivo é este? Quais as suas funções e habilidades? Como proceder para requisitar dados e enviar comandos quando da utilização do dispositivo? Respondendo a estas perguntas os agentes que pertencem à classe *CommonAgents* terão uma forma clara de como proceder na comunicação com os dispositivos do chão de fábrica.

Esta proposta para ontologia não foi implementada e nem pesquisada em profundidade, pois este não é o foco principal do trabalho. Isso merece um estudo muito mais amplo e cauteloso para decidir quais são as principais classes que representam os dispositivos de chão de fábrica, e as informações contidas nessas classes.

4.11 As mensagens trocadas entre os *AgentMachine* e os Dispositivos do Chão de Fábrica

Os *AgentMachine* são os agentes responsáveis por "interfacear" a troca de informações entre os agentes/sistemas multiagente e os dispositivos no chão de fábrica. Desta forma, os *AgentMachine* são encarregados de estabelecer a comunicação com o VMD, através do ORB, assim que o sistema é lançado. Para realizar este serviço ele irá chamar o processo de inicialização da comunicação, que é responsável por estabelecer uma associação com o VMD através do serviço *Initiate()* e criar o ambiente para a execução dos processos MMS com o serviço *CreateProgramInvocation()*. Após isto, o ambiente MMS está "instanciado" e pronto para trocar informações/comandos. A Figura 26 ilustra um diagrama de seqüência destas interações (BOOCH *et al.* 2000) e (LEE, 1997).

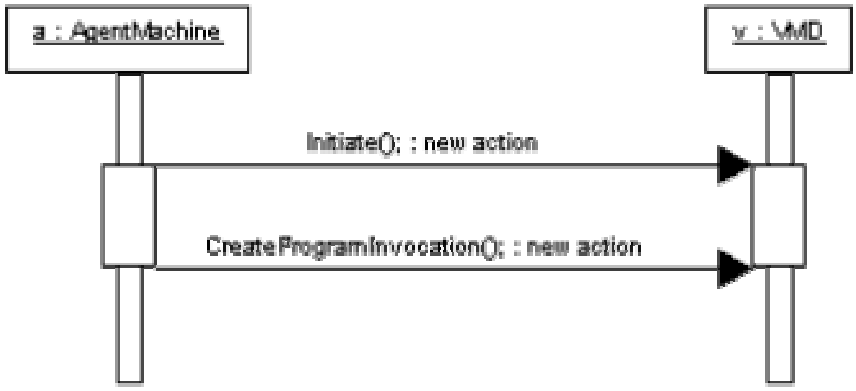


Figura 26 – Iniciar os serviços

Da mesma forma que o *AgentMachine* é responsável por estabelecer a comunicação com o VMD, ele é responsável por terminar esta comunicação quando necessário. Para finalizar a comunicação com o VMD o *AgentMachine* usa o processo *TerminateCommunication(),* que é responsável por finalizar o ambiente para execução dos programas com o serviço *DeleteProgramInvocation(),* e finalizar a associação através do serviço *Conclude()* (Figura 27).

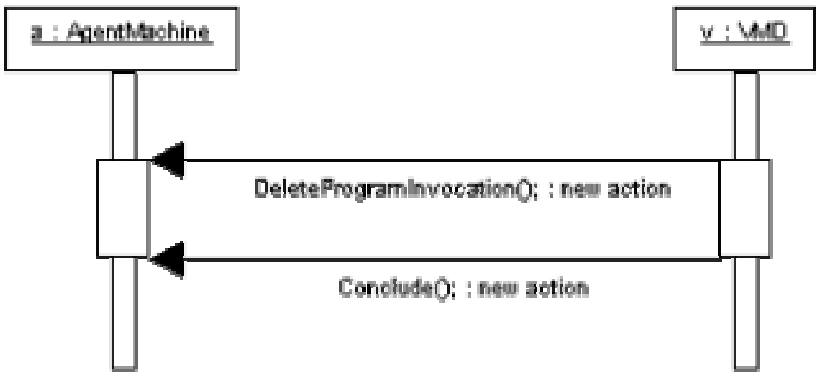


Figura 27 – Terminar Serviços

Uma vez estabelecida a comunicação com os VMD, as demais mensagens são particulares a cada dispositivo industrial.

As principais mensagens trocadas entre o *AgentMachine* e o VMD dos Centros de Usinagem estão modeladas nos diagramas de seqüência a seguir.

O *AgentMachine* usa o comando *Write* para colocar o CNC dos Centros de Usinagem em modo de operação remoto e para energizar o equipamento (Figura 28).

Figura 28 – Write

Para carregar um determinado domínio que será utilizado pelos centros de usinagem são usadas as mensagens/comandos *InitiateDownLoadSequence(), LoadSegment()* e *TerminateDownLoadSequence()* pelo *AgentMachine* (Figura 29)

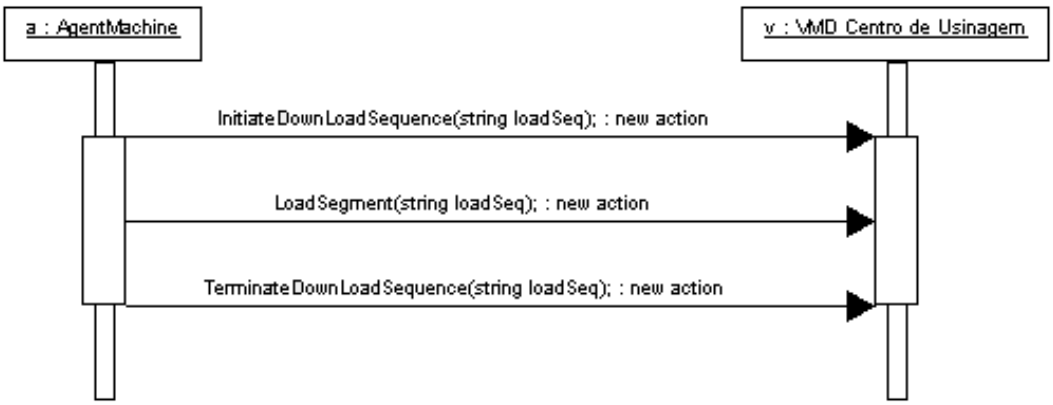


Figura 29 – Load Sequence

O comando *Start()* coloca o Centro de Usinagem em funcionamento (Figura 30) e o comando *Stop()* (Figura 31) interrompe o processo de usinagem por algum motivo.

Figura 30 – Start (Centro de Usinagem)

Figura 31 – Stop (Centro de Usinagem)

Sempre que o *AgentMachine* necessitar saber o *Status* físico dos Centros de Usinagem ele invoca o comando *Status()*, que por sua vez retorna o estado atual do dispositivo (Figura 32).

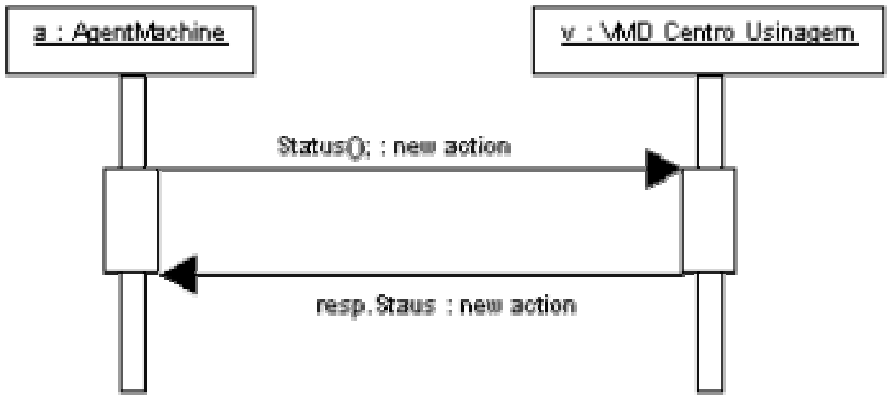


Figura 32 – Status (Centro de Usinagem)

As principais mensagens trocadas entre o *AgentMachine* e o VMD da Mesa Posicionadora estão modeladas a seguir.

O comando *CreateProgramInvocation* é usado para criar a invocação de um programa que será usado pela Mesa Poscionadora (Figura 33) e o comando *DeleteProgramInvocation* tem por objetivo finalizar esta invocação após a execução dos serviços (Figura 34).

Figura 33 – Criar Invocação de Programa

Figura 34 – Excluir Invocação de Programa

Sempre que o *AgentMachine* necessitar saber o *Status* físico da Mesa Posicionadora ele invoca o comando *Status()*, que por sua vez retorna o estado atual do dispositivo (Figura 35).

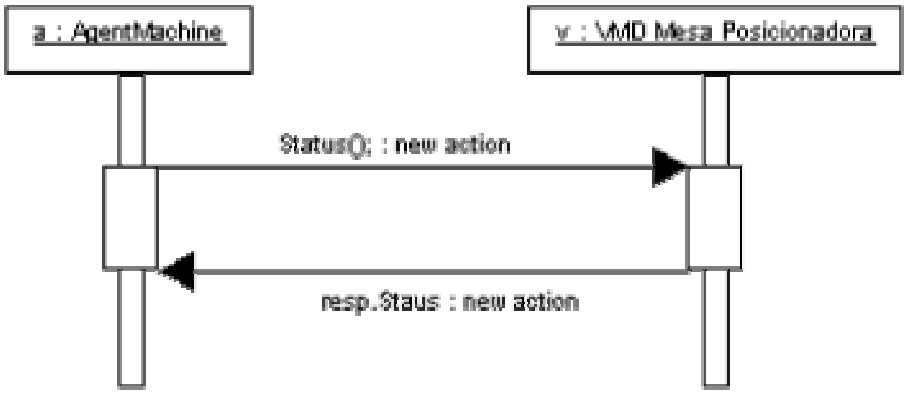


Figura 35 – Status (Mesa Posicionadora)

Sempre que é necessário mudar a posição da mesa, usa-se o comando *DataExchange()* (Figura 36).

Figura 36 – DataExchange

As principais mensagens trocadas entre o *AgentMachine* e o VMD do Robô estão modeladas a seguir.

Para carregar algum domínio utilizado pelo Robô são usadas as mensagens *InitiateDownLoadSequence(), LoadSegment()* e *TerminateDownLoadSequence()* pelo *AgentMachine* (Figura 37).

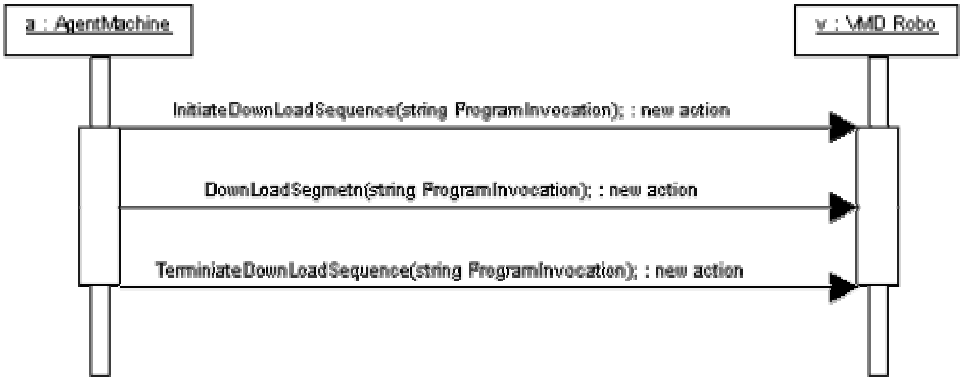


Figura 37 – Load Sequence

O comando *Start()* coloca o Robô em funcionamento (Figura 38).

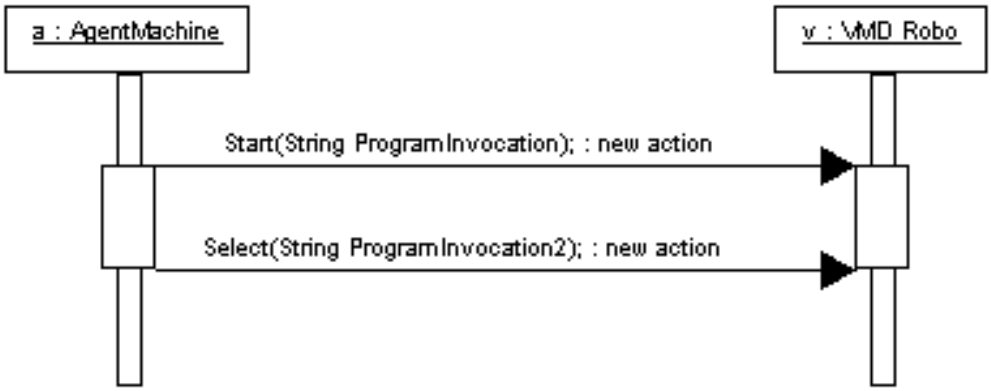


Figura 38 – Start (Robô)

Sempre o *Status* físico do Robô é obtido pelo comando *Status()*, que por sua vez retorna o estado atual do dispositivo (Figura 39).

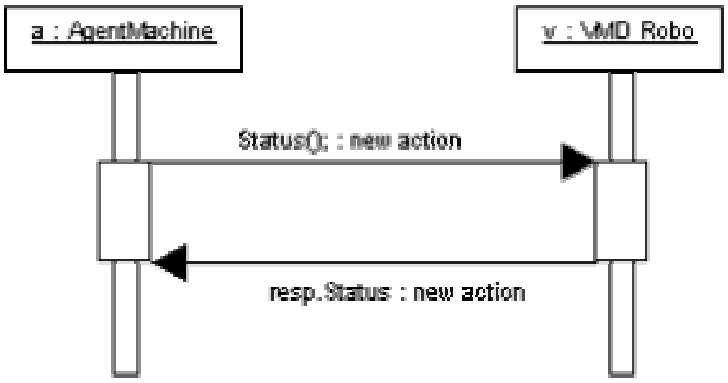


Figura 39 – Status (Robô)

As principais mensagens trocadas entre o *AgentManager* e os *AgentMachine* estão modeladas a seguir.

A performativa *ask-if()* é feita quando algum agente necessita saber alguma informação, por exemplo o *AgentManager* pergunta ao *AgentMachine* o *status* físico do dispositivo (seja ele qual for). O *AgentMachine*, por sua vez, responde ao *AgentManager* usando a performativa *reply()*.

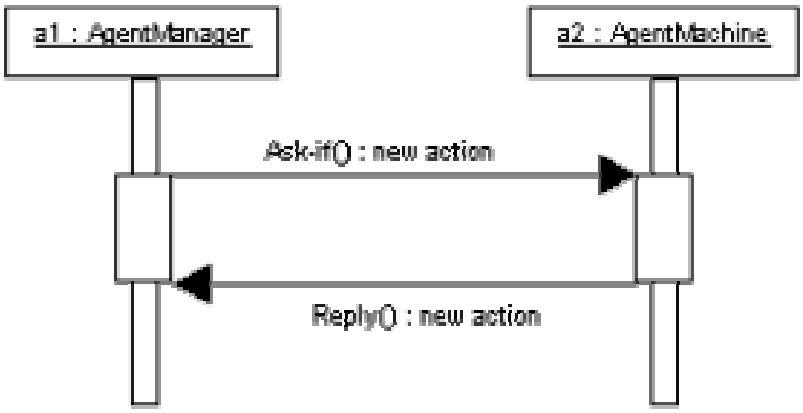


Figura 40 – Ask-if / Reply

Já a performativa *tell()* é usada para comunicar alguma agente alguma informação não solicitada ou enviar algum comando, por exemplo o *AgentMachine* detecta algum problema na execução de um serviço e informa ao *AgentMachine.* Quando um agente não pode executar algum procedimento informado pela performativa *tell()*, este agente responde ao emissor da mensagem com a performativa *sorry()* informando da impossibilidade de executar a ação (Figura 41).

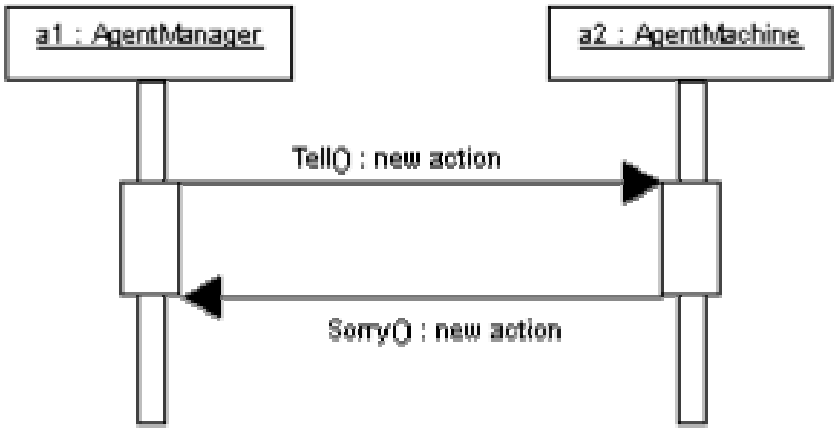


Figura 41 – Tell / Sorry

A Figura 42 mostra a troca de informações/interação entre os agentes e um dispositivo industrial na execução de uma tarefa (rotacionar a mesa posicionadora):

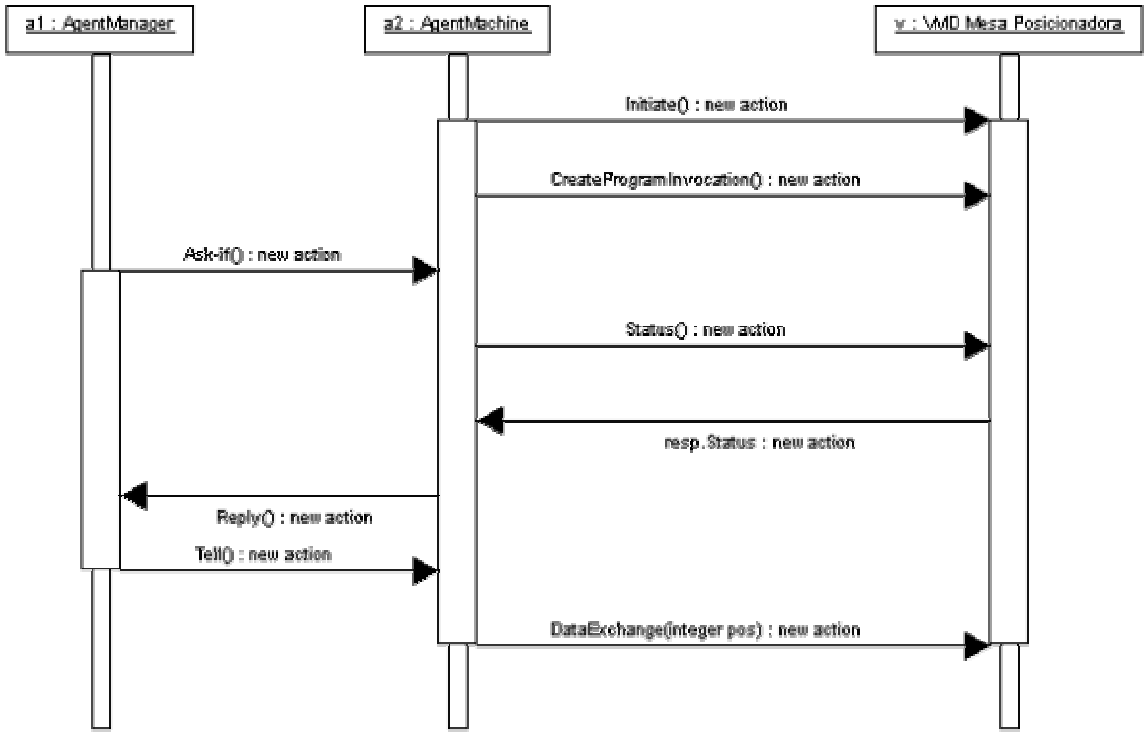


Figura 42 – Interação

4.12 Ambiente de Simulação

O protótipo foi implementado usando a plataforma de sistemas multiagente *Massyve Kit 3.0* (MASSYVE, 2000). O *Massyve Kit 3.0* é uma ferramenta que permite o desenvolvimento de aplicações sistemas multiagente, de maneira amigável e interativa. Ele é essencialmente destinado para a área de educação e atividades de treinamento, embora, na prática, ele possa ser usado para desenvolver sistemas multiagente de relativa complexidade para vários domínios de aplicações.

Em geral, o *Massyve Kit 3.0* permite aos usuários configurarem algumas opções de sistema para projetar arquiteturas de referências, derivar sistemas particulares auxiliados por um editor gráfico, e gerenciar completamente o ciclo de vida dos agentes. Uma vez os agentes lançados, eles podem enviar/receber mensagens de/para cada agente (via CORBA), usando um protocolo de comunicação livre de contexto.

Desde a versão 2.2 do Kit, os agentes podem ser executados em máquinas distribuídas. Os agentes são implementados como objetos. Os usuários podem criar aplicações multiagente particulares por meio de modificações na "classe de agentes padrão" (alterando, excluindo e adicionando novos atributos e/ou métodos), que apenas oferece os serviços básicos de comunicação e gestão.

Como já descrito neste capítulo, três classes de agentes foram criadas (derivadas da classe de agentes padrão do *Massyve Kit 3.0*). Para efetuar as alterações na classe de agentes padrão e implementar os softwares quem emulam os dispositivos industriais, foi utilizada a ferramenta de desenvolvimento C++ Builder 5.0 (BORLAND, 2000). O C++ Builder 5.0 é um poderoso ambiente de desenvolvimento C/C++, também em ANSI C++, para acelerar a construção de aplicações. O C++ Builder permite a criação de sistemas flexíveis com suporte ao XML e usar um *middleware* flexível com suporte à plataforma CORBA, impulsionando o desenvolvimento de aplicações para computação distribuída.

O ORB CORBA utilizado na comunicação entre os agentes da classe *AgentMachine* e os módulos que simulam os dispositivos industriais foi o ACE-TAO (TAO, 2001), ACE versão 5.1 e TAO versão 1.1.

O ambiente do *Massyve Kit 3.0* quando lançado também inicia o serviço de nomes do ACE-TAO. No momento que os agentes são lançados eles fazem o seu cadastro junto ao serviço de nomes, tornado este serviço transparente aos usuários. Já os softwares que emulam os dispositivos industriais de chão de fábrica devem ser cadastrados no serviço de nomes manualmente.

Nenhuma extensão do CORBA (como FT[51], RT[52] e QoS[53]) foi usada nesta implementação, maiores detalhes ver seção 5.1.

Os Sistemas Operacionais utilizados nas simulações com o protótipo foram o MS-Windows XP e MS-Windows NT 4.0.

---

[51] FT – *Fault Tolerance*
[52] RT – *Real Time*
[53] QoS – *Quality of Service*

A Figura 43 ilustra a interface da ferramenta *Massyve Kit 3.0* (editor gráfico) com os agentes utilizados na simulação. As linhas representam as conexões (canais de comunicação) possíveis e mostram qual agente pode se comunicar com qual. Os agentes no primeiro quadro pertencem à classe *CommonAgent*, os agentes do quadro do meio pertencem à classe *AgentManager* e os agentes do quadro mais inferior pertencem à classe *AgentMachine*. Os círculos que contém os títulos C1, C2 e C3 significam respectivamente Computador1, Computador2 e Computador3 e identificam em qual computador os agentes estão rodando. Maiores detalhes sobre a distribuição dos agentes são encontrados na seção 4.13.

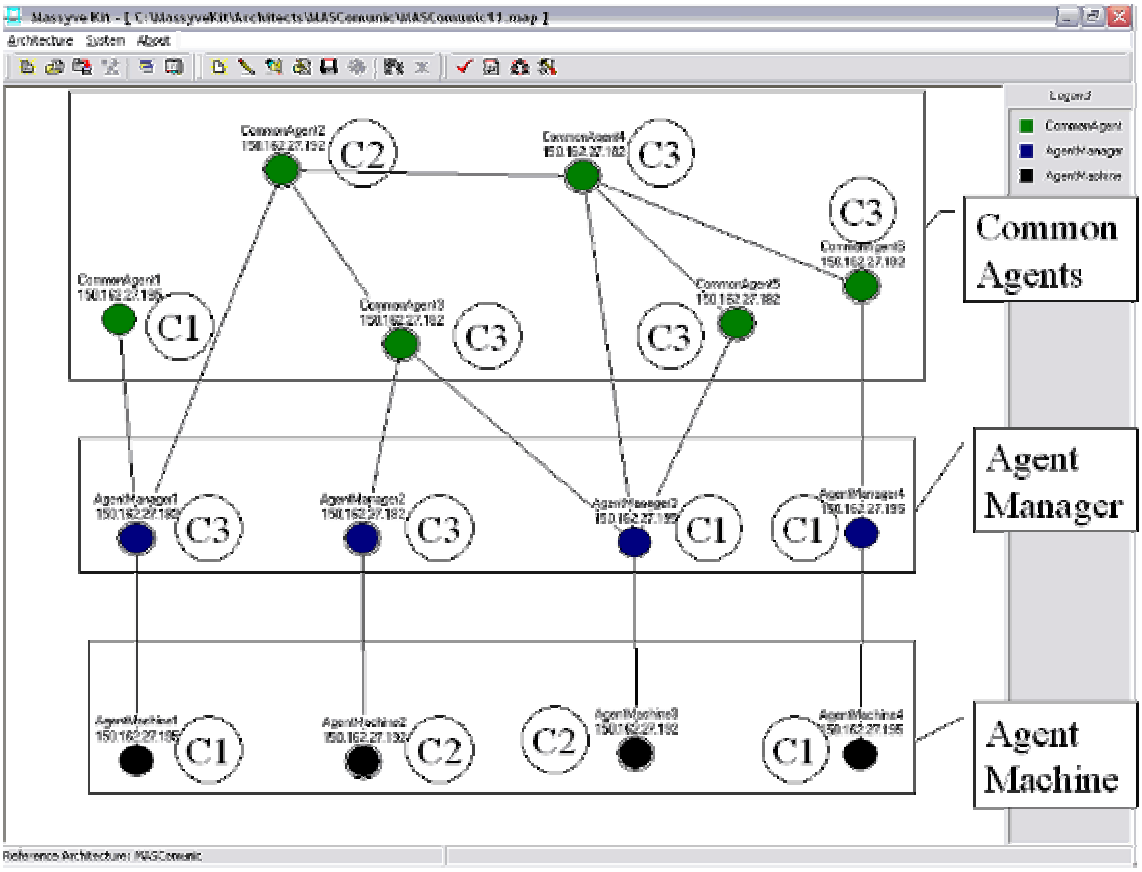


Figura 43 – Interface do Editor Gráfico Massyve

A Figura 44 está dividida em duas partes. A primeira parte ilustra o *Massyve Kit 3*.0 com quatro agentes "lançados", sendo dois *CommonAgent,* um *AgentManager* e um *AgentMachine.* A segunda parte ilustra a interface gráfica dos agentes e do software que emula o dispositivo industrial, representando a comunicação de um dos dispositivos de

chão de fábrica (mesa posicionadora) com o seu respectivo agente *AgentMachine*, e deste com o agente *AgentManager*, responsáveis pela troca de informações.

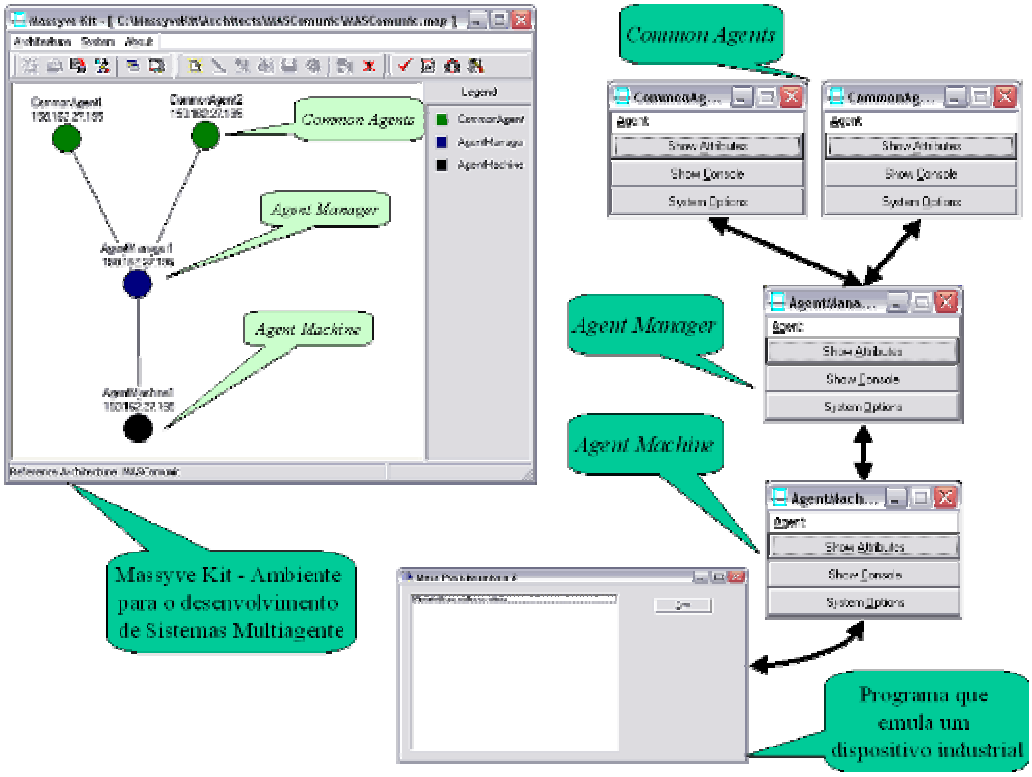


Figura 44 – Protótipo

4.13 Testes e Análise dos Resultados

Esta seção tem o intuito de descrever os testes e o funcionamento do protótipo dentro de um cenário da troca de mensagens entre os agentes e os dispositivos do chão de fábrica em um ambiente distribuído.

Foram lançados no total 14 agentes. Seis agentes pertencentes à classe *CommonAgent,* como meio de representar uma arquitetura multiagente qualquer. Quatro agentes pertencentes à classe *AgentManager*, e mais quatro agentes pertencentes à classe *AgentMachine*, que por sua vez representam os quatro dispositivos de chão de fábrica propostos na dissertação (Mesa Posicionadora, Centro de Usinagem Horizontal, Centro de Usinagem Vertical e Robô).

Os agentes foram distribuídos aleatoriamente em três PCs Intel Pentium II 500 MHz com 128 Mb de memória RAM, dois deles com a plataforma MS-WindowsXP e um com a plataforma MS-Windows NT 4.0 inseridos em uma rede Ethernet, TCP/IP, com um *switch* de 100 Mbits, da seguinte maneira:

- Computador 1: um agente da classe *CommonAgent*, dois agentes da classe *AgentManager*, dois agentes da classe *AgentMachine*, e um software para emular o dispositivos de chão de fábrica da Mesa Posicionadora, conforme mostra a Figura 45.

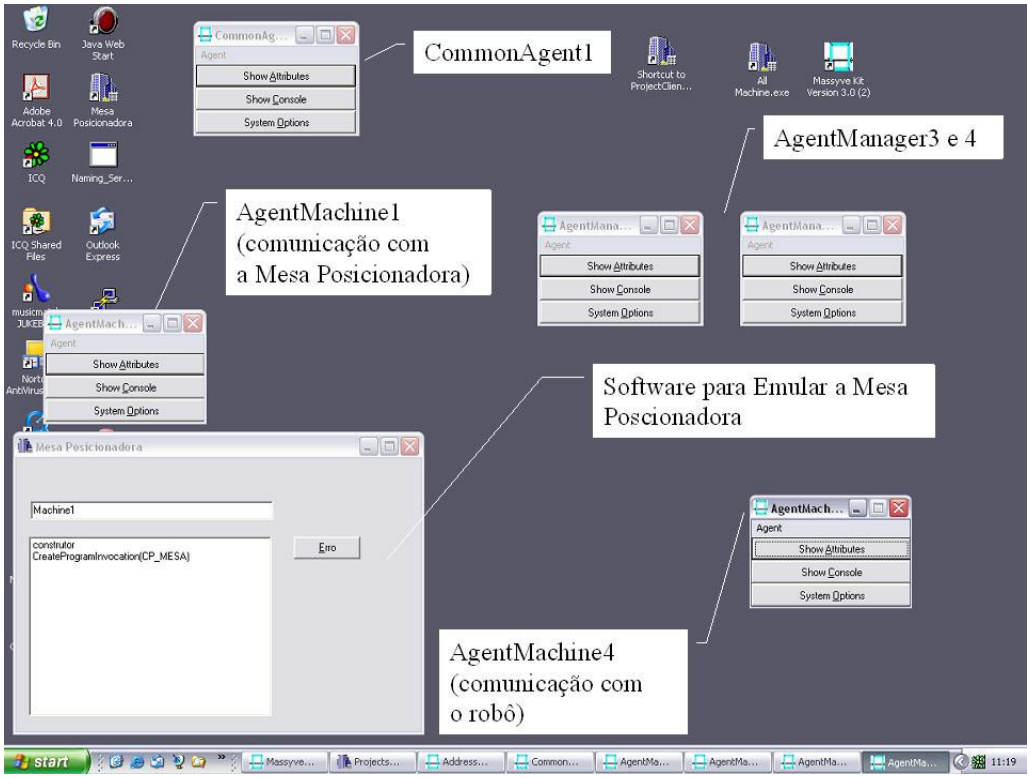


Figura 45 – Computador 1

- Computador 2: um agente da classe *CommonAgent*, dois agentes da classe *AgentMachine*, e três softwares para emular os dispositivos de chão de fábrica Centro de Usinagem Horizontal, Centro de Usinagem Vertical e Robô, conforme mostra a Figura 46.

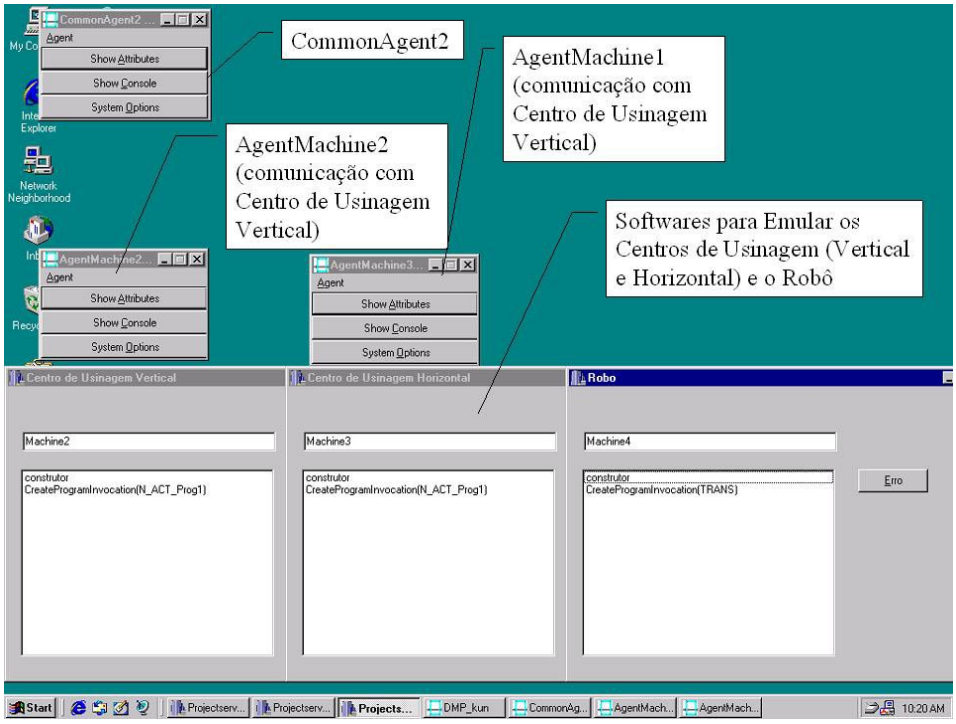


Figura 46 – Computador 2

- Computador 3: quatro agentes da classe *CommonAgent* e dois agentes da classe *AgentManager* , conforme mostra a Figura 47.

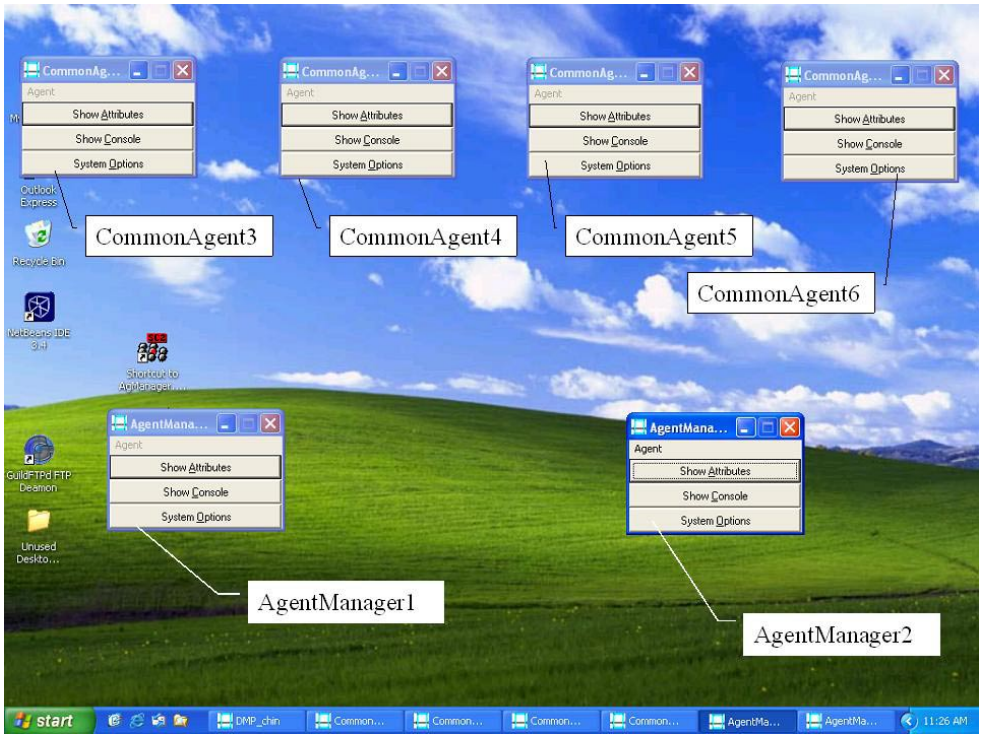


Figura 47 – Computador 3

Apesar dos testes terem sido efetuados usando o sistema operacional Windows, nada impede a implementação da arquitetura proposta em um sistema multiagente para outros sistemas operacionais (Linux, por exemplo) ou multiplataforma (Windows, Linux entre outras).

Parte-se do princípio que todos os agentes pertencentes a classe *CommonAgent* já estão cientes de quem são e o que representam os agentes pertencentes à classe *AgentManager*, assim, todos os *CommonAgents* já sabem como enviar e qual o formato das mensagens enviadas aos *AgentManager*.

Os testes foram realizados da seguinte forma:

Um *CommonAgent* envia uma mensagem KQML-XML para um *AgentManager*. Este traduz o conteúdo da mensagem (comando/solicitação de serviço) no formato de mensagens MMS, empacota o comando/solicitação de serviço em uma mensagem KMQL-XML e encaminha para o *AgentMachine* executar o serviço. O *AgentMachine* recebe e mensagem e atua com o dispositivo do chão de fábrica, que por sua vez, responde as requisições. Quando o *AgentMachine* recebe a resposta do chão de fábrica ele devolve o resultado para o *AgentManager*, que responderá ao *CommonAgent*.

As mensagens testadas foram:

- Solicitação do status físico do dispositivo.

Os agentes *CommonAgent* pedem o status físico do dispositivo de chão de fábrica e recebem a resposta *OPERABLE* (operável) ou *INOPERABLE* (inoperável). Para isso um *CommonAgent* envia uma mensagem KQML-XML para um *AgentManager* com a performativa *Ask-if* e o conteúdo *Status()*. O *AgentManager* encaminha esta mensagem para o *AgentMachine* que ele representa e aguarda a resposta. O *AgentMachine* executa a chamada da mensagem MMS *mm_Status()* no dispositivo de chão de fábrica. O retorno da mensagem é empacotada em uma mensagem KQML-XML, com uma performativa *Reply*, que indica uma resposta, e enviada ao *AgentManager*, que por sua vez enviará esta resposta ao *CommonAgent* que fez a requisição.

- Iniciar o funcionamento do dispositivo de chão de fábrica Centro de Usinagem Vertical.

Um *CommonAgent* ordena o início de um trabalho (*job*) ao Centro de Usinagem Vertical enviando uma mensagem KQML-XML para um *AgentManager* que representa este dispositivo com a performativa *Tell* e o conteúdo *Iniciar*. O *AgentManager* traduz o comando *Iniciar* que ele recebeu para o comando *Start()*, empacota estes novos dados em uma mensagem KQML-XML e encaminha para o *AgentMachine.* O *AgentMachine* verifica se o Centro de Usinagem Vertical está operável ou não através de uma chamada MMS (*mm_Status()*). Caso o status do dispositivo industrial seja *OPERABLE* ele executa o serviço, espera o retorno da mensagem MMS *mm_Start(N_ACT_Prog1)* e responde ao *AgentManager* com a performativa *Reply* que o serviço foi executado com sucesso. Por outro lado, se o retorno da mensagem MMS *mm_Start(N_ACT_Prog1)* falhar (for *false*) ou se o status do dispositivo é *INOPERABLE* , o *AgentMachine* encaminha ao *AgentManager* uma mensagem KQML-XML com a performativa *Sorry* indicando que o serviço não pode ser executado com sucesso. O processo para iniciar o funcionamento dos demais dispositivos é o mesmo, apenas o valor passado como parâmetro na mensagem *mm_Start()*, e claro, os destinatários da mensagem mudam.

- Rotacionar a mesa posicionadora.

Segue praticamente o mesmo processo descrito anteriormente, um *CommonAgent* ordena à Mesa Posicionadora rotacionar para a posição X. O *CommonAgent* envia uma mensagem KQML-XML para o *AgentManager* que representa este dispositivo com a performativa *Tell* e o conteúdo *Rotacionar*valor*. O *AgentManager* traduz o comando *Rotacionar*valor* que ele recebeu para o comando *DataExchange(valor)*, empacota estes novos dados em uma mensagem KQML-XML e encaminha para o *AgentMachine.* Este verifica se a Mesa Posicionadora está operável ou não através de uma chamada MMS (*mm_Status()*). Caso o status do dispositivo industrial seja *OPERABLE* ele executa o serviço, espera o retorno da mensagem MMS *mm_DataExchange(valor)* e responde ao *AgentManager* com a performativa *Reply* que o serviço foi executado com sucesso. Por outro lado, se o retorno da mensagem MMS *mm_DataExchange(valor)* falhar (for *false*) ou se o status do dispositivo é *INOPERABLE* , o *AgentMachine* encaminha ao

AgentManager uma mensagem KQML-XML com a performativa *Sorry* indicando que o serviço não pode ser executado com sucesso.

- Envio de mensagens de erro

Quando um erro ocorre no dispositivo de chão de fábrica o *AgentMachine* responsável pela comunicação com este dispositivo manda uma mensagem KQML-XML para o *AgentManager* que o representa para os demais agentes indicando da falha no dispositivo. Esta mensagem contém a performativa *Tell* e o campo *content* a mensagem *INOPERABLE*.

Durante os testes foram efetuadas várias baterias de troca de mensagens entre os agentes e os dispositivos do chão de fábrica. Os *CommonAgent* faziam requisições de serviços e aguardavam as respostas confirmando o sucesso, o fracasso ou o erro nas operações. Também foram simulados erros nos dispositivos do chão de fábrica; os *AgentMachine*, responsáveis pela monitoração dos dispositivos, enviaram as mensagens de "aviso/alerta" aos *AgentManager* informando a eles o problema.

As mensagens trocadas entre os agentes e os dispositivos industriais foram "instantâneas", porém não obtivemos os tempos de envio/resposta das mensagens devido ao ambiente de testes não ser uma planta industrial real e também não possuir dispositivos industriais reais, mas sim softwares que emulavam o funcionamento destes. Podemos estimar que o processamento das mensagens não atende os requisitos temporais que as aplicações do chão de fábrica que em geral exigem. Supomos que os fatores que contribuem para o não cumprimento destes requisitos temporais se deve ao "peso" no processamento das mensagens KQML-XML (que descrevem toda a estrutura da mensagem) e o *overhead* provocado pela adoção do CORBA como um *middleware* na troca das mensagens. Também vale ressaltar que o ambiente utilizado para o desenvolvimento dos agentes (*Massyve Kit 3.0*) não tem a preocupação com o tempo de envio e recebimento das mensagens.

O sistema mostrou um alto nível de interoperabilidade, possibilitando que um agente ou um sistema multiagente (representados pelos *CommonAgent*) que não "conhece" e não "entende" a sintaxe de comandos MMS possa atuar com o dispositivo do

chão de fábrica "conversando" com apenas um agente que representa cada dispositivo do chão de fábrica, o *AgentManager*, pois todo o resto da comunicação está "escondido" por ele, e, permitiu também, que os dispositivos de chão de fábrica respondessem as requisições/comandos recebidos.

Requisitos de Software e Hardware do Protótipo

Alguns requisitos para executar o ambiente multiagente são necessários. Separamos estes requisitos em nível de software e hardware para facilitar a compreensão do leitor.

Requisitos de Software

- Sistema Operacional Microsoft Windows 95.
- Sistema Operacional Microsoft Windows 98.
- Sistema Operacional Microsoft Windows NT 4.0: para rodar a versão 3.0 do *Massyve Kit*, normalmente é necessário trabalhar com o *login* de Administrador, pois pode ocorre problemas ao acesso em arquivos e diretórios.

É necessário ter as DLL[54]s (*ace_bp.dll, tao_bp.dll* e *orbsvcs_bp.dll*) do ORB ACE-TAO em todas as máquinas onde será rodado o sistema multiagente.

Sempre que for necessário efetuar alterações/incluir novas características os agentes implementados no protótipo se faz necessário o Compilador Borland C++ Builder (5.0 ou superior).

Requisitos de Hardware

O sistema requer uma configuração mínima para suportar as aplicações e, principalmente, lançar os agentes. O espaço em memória RAM e em disco rígido necessário irá depender do número de agentes que o sistema terá. Obviamente, quanto mais agentes no sistema, mais memória RAM (principalmente) e espaço em disco rígido serão necessários.

[54] DLL – *Dynamic Linking Library*

O tamanho dos agentes em disco rígidos é relativamente pequeno, 700 KB na classe padrão e chegando até 950 KB nos agentes implementados. Contudo, uma vez lançados, os agentes usam um grande espaço na memória, podendo alcançar até 12 MB, por cada agente. Por esta razão, é normal que o sistema faça *swapping* em disco para suportar a execução da aplicação toda. Isso pode prejudicar a performance do sistema.

Configuração Mínima

- Intel Pentium 400 MHz ou outro processador compatível;
- 128 MB RAM;
- 150 MB HD;
- Monitor VGA Color;
- Mouse compatível com Microsoft Windows.

Configuração Recomendada

- Intel Pentium 800 MHz ou outro processador compatível;
- 256 MB RAM;
- 150 MB HD;
- Monitor SVGA Color;
- Mouse compatível com Microsoft Windows.

É importante ressaltar que estes requisitos foram necessários para a implementação do protótipo, mas não são necessários para a implementação da proposta em outros sistemas multiagente que podem ser utilizados em outros sistemas operacionais, como por exemplo plataformas Linux. Para isso é necessário um estudo dirigido a cada sistema multiagente e o(s) seu(s) respectivo(s) sistema(s) operacional(is).

5. Conclusões

As empresas são cada vez mais requisitadas a mostrar uma maior agilidade para manter a competitividade enquanto enfrentam novas tendências de manufatura como a customização em massa. A agilidade necessária não deve ser limitada apenas aos aspectos de alto nível, mas deve ter as características de suporte a todos os níveis da empresa, em especial os desafios de tornar ágil o chão de fábrica (BARATA, 2000).

Esta dissertação trouxe uma proposta de plataforma para Sistemas Multiagente para usar vários protocolos padrão com o objetivo de integrar as informações desde o nível mais baixo do chão de fábrica, através do protocolo MMS, até o nível mais alto, compartilhamento de conhecimento, KQML, para atuar na tomada de decisões de maneira padronizada (KQML + XML) e robusta (CORBA) integrados e embutidos com Sistemas Multiagente. A integração dos dados permite às organizações competir de forma diferenciada pois proporciona meios para sua reconfiguração e readaptação de maneira rápida, aproveitando as oportunidades que o mercado oferece ou respondendo às expectativas que o mercado exige. Recentes tendências na indústria estão enfatizando a relevância da agilidade, compreendida como a habilidade de reconhecer, rapidamente reagir a cooperar com as mudanças imprevisíveis do ambiente (CAMARINHA-MATOS *et al.* 2000). O dinamismo proporcionado por ambiente flexível e ágil as mudanças ajuda a aumentar a flexibilidade nos processos de decisão.

A proposta apresentada traz uma visão harmonizada mostrando como é possível encapsular a comunicação nos diferentes níveis industriais sem que os atores envolvidos tenham a necessidade de conhecer e compreender os detalhes de todos os protocolos e dispositivos industriais envolvidos nesta comunicação, suprindo, assim, os problemas existentes na troca de informações nos diferentes níveis industriais. A compatibilidade técnica é proporcionada e as tecnologias usadas para a obtenção dos dados no chão de fábrica tornam-se transparentes para aqueles que estão requisitando as informações, tornando a comunicação harmônica. Também uma maior flexibilidade é possibilitada quanto ao uso dos equipamentos do chão de fábrica em diversas composições de

possíveis células no nível de software, levando a inteligência o mais perto possível dos dispositivos industriais.

Qualquer um que queira desenvolver um sistema multiagente onde exista a necessidade de obter e enviar informações/comandos para o chão de fábrica pode fazer isso de maneira transparente, eficiente e robusta através da plataforma descrita.

Algumas limitações durante a implementação do protótipo e o estudo das tecnologias apresentadas foram levantadas, e a sua aplicação como uma solução definitiva ainda não está clara ou pode não ser ideal. Um dos problemas constatados é o "relacionamento" entre os agentes pertencentes à classe *AgentMachine* e o VMD. A interface entre eles é feita através de uma IDL, onde os *AgentMachine* fazem o papel de cliente e o VMD o papel de servidor. Até este ponto todas as mensagens geradas pelos *AgentMachine* têm como respostas o retorno das funções (*void, AnsiString, int*) onde é possível saber o estado do VMD. Contudo, o VMD, ao gerar uma exceção ou mensagens não solicitadas (como o caso de uma falha), não pode mandar estas mensagens aos *AgentMachine* pois ele faz apenas o papel de Servidor, apesar das características do MMS permitir o papel de Cliente e de Servidor ao mesmo tempo. É possível fazer com que o VMD tenha papel de Cliente/Servidor, porém, esta solução não foi adotada pois torna o VMD "amarrado" aos *AgentMachine*. Além disso, teria todo o "peso" da parte cliente (que não é tão *thin* assim) CORBA incorporado na aplicação VMD, que pode comprometer a atuação junto com o dispositivo industrial (por questões de tempo real). Se, em um futuro próximo, os agentes se tornarem "móveis", isso poderá ser um empecilho.

Como uma solução deste problema, os *AgentMachine* monitoram continuamente o cliente (VMD) e todas as mensagens não solicitadas no lado do cliente são depositadas em uma área onde o agente faz uma verificação contínua (como um *black board*[55]) .

Como já citado em (RABELO, 1997), a norma MMS não se apoia numa outra norma, *de facto*, que é o TCP/IP, criando grandes problemas de implantação do MMS (na

[55] *Black Board* – Quadro Negro – Maiores detalhes em BITTENCOURT (1998)

sua especificação original) dada a grande penetração do TCP/IP nas empresas. Por outro lado, o TCP/IP não tem um bom desempenho nas questões de tempo real que as aplicações de chão de fábrica exigem.

A implementação do protótipo buscou implantar todas as mensagens MMS de acordo com a norma, contudo devido à falta de um ambiente verdadeiramente consistente algumas mensagens podem não estar de acordo com a norma.

A adoção da arquitetura CORBA pelas aplicações que trocam informações com o chão de fábrica (*AgentMachine*) deve ser estudada com maior ênfase. Devido ao fato do COBRA "rodar em cima" do protocolo TCP, ele pode não ser indicado para aplicações que tem como uma das principais características o requisito temporal.

O modelo proposto é uma alternativa que pode ser implementada em sistemas multiagente, sem maiores impactos nas arquiteturas de comunicação já existentes ou a serem criadas; porém existem limitações quanto a quesitos de tempo. Vale ressaltar que a proposta proporciona o aumento da integração e troca de informações entre aplicações de ato nível e os dispositivos do chão de fábrica.

5.1 Perspectivas para Trabalhos Futuros

Como sugestões para a continuidade e o aperfeiçoamento deste trabalho é sugerida a pesquisa mais detalhada em algumas áreas onde não foi possível, principalmente, e devido à sua grande abrangência. Algumas destas sugestões, por si só, são interessantes e importantes para o aumentar a interoperabilidade e a "confiança" na comunicação entre os agentes e os dispositivos do chão de fábrica:

- A utilização de um ORB CORBA estendendo as funções de RT para integração em tempo real dos atores *AgentMachine* e dispositivos industriais ou os seus sistemas legados (objetos VMD), capaz de satisfazer os requisitos de ordem temporal das aplicações é uma necessidade de extrema importância do ponto de vista de aplicações industriais.

- Estender as funções de FT e QoS de um ORB CORBA permitindo maior segurança na entrega das mensagens.
- Uma interface dos *AgentManager* com a Web, como uma forma de integração com usuários finais que querem obter informações do chão de fábrica. Isso quer dizer, dar "poderes" aos *AgentManager* manter uma base de dados em um servidor Web, tornando estas informações acessíveis de qualquer lugar, desde que exista acesso à Internet.
- Trabalhar no aumento da segurança na transação das informações entre os agentes, seja qual for a classe a que eles pertencem.
- Arquiteturas híbridas, baseadas em bancos de dados federados/distribuídos para agentes, onde os dados não são enviados de um agente para outro por um protocolo de alto nível (como nas maneiras usuais), mas sim acessando bases de dados federadas dos agentes. Com esta abordagem os protocolos de alto nível estão "livres" para atividades de cooperação e controle, separando as funções de controle dos dados (RABELO, 2000c).
- O "desenvolvimento" ou o emprego de uma ontologia para descrever os dispositivos de chão de fábrica e as suas características trazendo clareza – comunicar efetivamente as diferentes pretensões para os agentes, coerência – deve ser consistente internamente, e extensibilidade – capacidade de definir novos termos se baseando no vocabulário existente, de tal forma que não se precise fazer revisões de definições existentes. O emprego de uma ontologia padronizará um vocabulário para a troca de mensagens entre os agentes.

Anexo A – Principais Performativas KQML

Neste anexo serão apresentadas as principais performativas KQML reservadas. Isto é, são reservadas no que se refere quando uma implementação usa qualquer nome de uma das performativas de maneira inconsistente com o que será apresentado abaixo, então a implementação não será coerente com o KQML.

Definições de novas performativas devem seguir o estilo das definições desta seção. A definição deve conter o seguinte:

- O nome da performativa;
- Todos os parâmetros chave que a performativa pode conter;
- Categorias sintáticas e semânticas para todos os valores e parâmetros com palavras-chave não reservadas;
- Qualquer limitação adicional sintática e semântica para valores de parâmetros com palavras-chave reservadas;
- Os valores padrão para todos os parâmetros ausentes;
- As semânticas, em termos de uma declaração que o emissor está fazendo para si mesmo, do nome da performativa aplicado aos parâmetros.

Performativas de Informações Básicas

```
tell
      :sender <word>
      :receiver <word>
      :content <expression>
      :language <word>
      :ontology <word>
      :in-reply-to <expression>
      :force <word>
```

Performativas deste tipo indicam que o comunicado da sentença contida no *:content* do emissor para o receptor.

```
deny
      :sender <word>
      :receiver <word>
      :content <performative>
      :language KQML
      :ontology <word>
```

```
      :in-reply-to <expression>
```

Performativas deste tipo indicam que o significado da <*performative*> embutido não é verdade para o emissor.

```
untell
      :sender <word>
      :receiver <word>
      :content <expression>
      :language <word>
      :ontology <word>
      :in-reply-to <expression>
      :force <word>
```

Uma performativa deste tipo é equivalente a uma *deny* de um *tell*

Performativas de Base de Dados

Estas performativas, INSERT, DELETE, etc. provêm uma habilidade aos agentes requisitar a outros agentes inserir ou excluir sentenças em seus bases de conhecimento.

```
insert
      :sender <word>
      :receiver <word>
      :content <expression>
      :language <word>
      :ontology <word>
      :reply-with <expression>
      :in-reply-to <expression>
      :force <word>
```

O emissor requisita ao receptor para adicionar a sentença *:content* na sua base de conhecimento. A performativa pode falhar ou ter sucesso. Possíveis erros e avisos são:

- Conteúdo duplicado, sentença já existe na base de conhecimento.
- Conteúdo contradiz sentenças que já estão na base de conhecimento.
- Emissor não autorizado a inserir o conteúdo.

```
delete
      :content <performative>
      :language KQML
      :ontology <word>
      :reply-with <expression>
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

O emissor requisita ao receptor para excluir a sentença *:content* da sua base de conhecimento. A performativa pode falhar ou ter sucesso. Possíveis erros e avisos são:

- Conteúdo não existe na base de conhecimento.
- Conteúdo necessariamente verdadeiro na base de conhecimento.
- Emissor não autorizado a excluir o conteúdo.

```
delete-one
      :content <performative>
      :order { first | last | undefined }
      :language KQML
      :ontology <word>
      :reply-with <expression>
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

O emissor requisita ao receptor excluir uma sentença da sua base de conhecimento.

O parâmetro opcional *:order* especifica se a sentença a ser excluída deve ser encontrada primeiro ou em último caso na base de conhecimento. O valor padrão para o parâmetro *:order* é *undefined*.

A performativa pode falhar ou ter sucesso. Possíveis erros e avisos são:

- Conteúdo não existe na base de conhecimento.
- Conteúdo necessariamente verdadeiro na base de conhecimento.
- Emissor não autorizado a excluir o conteúdo.

```
delete-all
      :content <performative>
      :language KQML
      :ontology <word>
      :reply-with <expression>
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

Esta performativa é como a performativa *delete-one*, exceto que a resposta deve ser uma coleção de aspectos instanciados correspondentes a todas as sentenças excluídas.

A performativa pode falhar ou ter sucesso. Possíveis erros e avisos são:

- Conteúdo não existe na base de conhecimento.
- Conteúdo necessariamente verdadeiro na base de conhecimento.
- Emissor não autorizado a excluir o conteúdo.

Respostas Básicas

```
error
      :sender <word>
      :receiver <word>
      :in-reply-to <expression>
      :comment <string>
      :code <integer>
```

Uma performativa deste tipo indica que o emissor não pode entender ou considera ser ilegal a mensagem referenciada pelo parâmetro *:in-reply-to*. O parâmetro *:code* dá um código numérico para classificar o tipo de erro. O parâmetro *:comment* pode ser usado para retornar uma *string* adicional descrevendo como o emissor considera que a mensagem não tem uma formatação correta.

```
sorry :
      in-reply-to <expression>
      :sender <word>
      :receiver <word>
      :comment <string>
```

Uma performativa deste tipo indica que o emissor entende, mas não é capaz de prover uma (ou mais) resposta(s) para gerenciar os parâmetros referenciados pelo *:in-reply-to*. Uma performativa deste tipo pode ser usada em resposta para uma pergunta de um *evaluate* ou *ask-one*, quando nenhuma outra resposta é apropriada. O parâmetro opcional *:comment* pode ser usado para passar uma *string* que descreve situações específicas que conduziram a não execução do serviço ou respostas adicionais.

Performativas de Pesquisa Básica

```
evaluate
      :content <expression>
      :language <word>
      :ontology <word>
      :reply-with <expression>
      :sender <word>
```

```
      :receiver <word>
```

As performativas deste tipo indicam que o emissor gostaria que o destinatário simplificasse a expressão no parâmetro *:content* e respondesse com o resultado.

```
reply
      :content <expression>
      :language <word>
      :ontology <word>
      :in-reply-to <expression>
      :force <word>
      :sender <word>
      :receiver <word>
```

As performativas deste tipo indicam que o emissor acredita que o *:content* é uma resposta apropriada para a questão na mensagem *:in-reply-to*.

```
ask-if
      :content <expression>
      :language <word>
      :ontology <word>
      :reply-with <expression>
      :sender <word>
      :receiver <word>
```

Uma performativa deste tipo é o mesmo que *evaluate*, exceto que o *:content* deve ser uma *sentence schema* na *:language*.

```
ask-about
      :content <expression>
      :language <word>
      :ontology <word>
      :reply-with <expression>
      :sender <word>
      :receiver <word>
```

Uma performativa deste tipo é o mesmo que *ask-if*, exceto que as respostas devem ser uma coleção de todas as sentenças na base de conhecimento do destinatário.

```
ask-one
      :content <expression>
      :aspect <expression>
      :language <word>
      :ontology <word>
      :reply-with <expression>
      :sender <word>
      :receiver <word>
```

Uma performative deste tipo é como um *ask-if*, exceto que o parâmetro *:aspect* descreve a forma desejada da resposta.

```
ask-all
      :content <expression>
      :aspect <expression>
      :language <word>
      :ontology <word>
      :reply-with <expression>
      :sender <word>
      :receiver <word>
```

Uma performativa deste tipo é como um *ask-one*, exceto que a resposta deve ser uma coleção de aspectos instanciados correspondentes a junção de todas as sentenças *:content* no destinatário na base de conhecimento.

```
sorry
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

Uma performativa deste tipo indica que o emissor entende, mas não é capaz de prover uma (ou mais) resposta(s) para gerenciar o parâmetros referenciado pelo *:in-reply-to*. A performativa deste tipo pode ser usada em resposta para uma pergunta de um *evaluate* ou *ask-one*, quando nenhuma outra resposta é apropriada.

Performativas de Consulta com Multi-resposta

```
stream-about
      :content <expression>
      :language <word>
      :ontology <word>
      :reply-with <expression>
      :sender <word>
      :receiver <word>
```

Este tipo é como *ask-about*, exceto que no lugar de responder com uma coleção de respostas, o "respondedor" deve enviar uma serie de performativas que quando tomadas juntas identifiquem os membros daquela coleção.

```
stream-all
      :content <expression>
      :aspect <expression>
      :language <word>
      :ontology <word>
      :reply-with <expression>
      :sender <word>
```

```
      :receiver <word>
```

Este tipo é como *ask-all*, exceto que no lugar de responder com uma coleção de aspectos instanciados, o "respondedor" deve enviar uma serie de performativas que quando tomadas juntas identifiquem os membros daquela coleção.

```
eos
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

A performativa "*End Of Stream*" indica que a seqüência de respostas para uma nova mensagem de múltiplas respostas *:in-reply-to* terminou com sucesso. Nenhuma outra resposta será enviada.

Performativas de Efeito Básico

```
achieve
      :content <expression>
      :language <word>
      :ontology <word>
      :force <word>
      :sender <word>
      :receiver <word>
```

Performativas deste tipo são requisições que o destinatário tenta fazer para as sentenças no *:content* verdadeiro do sistema (tecnicamente, o emissor quer que o receptor entenda a sentença como verdadeira no sistema).

```
unachieve
      :content <expression>
      :language <word>
      :ontology <word>
      :sender <word>
      :receiver <word>
```

Uma performativa deste tipo é o mesmo que um *deny* ou um *achieve*.

Gerador de Performativas

As seguintes performativas compreendem um mecanismo *generator* para entregar de resposta para uma performativa KQML.

```
standby
      :content <performative>
      :language KQML
      :ontology <word>
```

```
      :reply-with <expression>
      :sender <word>
      :receiver <word>
```

Este tipo indica que o emissor quer que o destinatário leve o que pretende ser a(s) resposta(s) da performative no *:content*, e anuncia a sua disponibilidade para aceitar pedidos para as respostas.

```
ready
      :reply-with <expression>
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

Este tipo indica que o emissor irá responder requisições para as respostas para as performativas incluídas em algumas performativas com o parâmetro *:in-reply-to*.

```
ready
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

Este tipo indica que o emissor gostaria de receber a próxima resposta daquelas prometidas pela performativa identificada pelo parâmetro *:in-reply-to*.

```
rest
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

Este tipo indica que o emissor gostaria de receber as respostas restantes, daquelas prometidas pela performativa *ready* identificada no parâmetro *:in-reply-to*.

```
discard
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

Este tipo indica que o emissor não irá emitir mais respostas para a performativa *ready* identificada no parâmetro *:in-reply-to*.

Performativas de Capacidade de Definição

```
advertise
      :content <expression>
      :language <word>
      :ontology <word>
      :force <word>
      :sender <word>
```

```
      :receiver <word>
```

Este tipo indica que o emissor está particularmente adaptado a processar as performativas da classe KQML descritas pelo parâmetro *:content*.

Performativas de Notificação

```
subscribe
      :content <expression>
      :language <word>
      :ontology <word>
      :force <word>
      :reply-with < expression >
      :sender <word>
      :receiver <word>
```

Este tipo indica que o emissor deseja que o receptor avise sobre futuras mudanças.

Performativas de Rede

```
register
      :name <word>
      :sender <word>
      :receiver <word>
```

Indica que o emissor pode entregar performativas para um agente nomeado pelo parâmetro *:name*.

```
unregister
      :name <word>
      :sender <word>
      :receiver <word>
```

Este tipo é o mesmo que um *deny* de um *register*.

```
forward
      :to <word>
      :from <word>
      :content <performative>
      :language <word>
      :ontology <word>
      :sender <word>
      :receiver <word>
```

Este tipo indica que o emissor quer que o agente *:to* processe a informação no parâmetro *:content* como se isso viesse do agente *:from* diretamente.

```
broadcast
      :from <word>
      :content <performative>
      :language <word>
```

```
      :ontology <word>
      :sender <word>
      :receiver <word>
```

Este tipo indica que o emissor gostaria que o receptor dirigisse a performativa *broadcast* para cada um dos seus contatos, mesmo aqueles receptores que já receberam a performativa *broadcast*.

```
pipe
      :to <word>
      :from <word>
      :reply-with <expression>
      :sender <word>
      :receiver <word>
```

Este tipo indica que um futuro tráfego neste canal pode ser roteado para o agente *:to*.

```
break
      :in-reply-to <expression>
      :sender <word>
      :receiver <word>
```

Uma performativa deste tipo interrompe uma performativa *pipe*.

```
transport-address
      :name <word>
      :content <expression>
      :sender <word>
      :receiver <word>
```

A performativa *transport-address* é uma forma de definir uma associação entre um nome simbólico para um agente KQML e um endereço de transporte.

Anexo B – Descrição dos Serviços MMS

Serviços de Gerenciamento de Contexto

Esta classe de serviços é constituída dos seguintes serviços:

- **Initiate**: este serviço torna possível a um usuário MMS iniciar um diálogo com outro usuário MMS, estabelecendo os recursos necessários para manter a comunicação no contexto MMS, através da negociação dos serviços que serão suportados dentro deste contexto. A execução, com sucesso, do serviço "Initiate" é indispensável para que qualquer outro serviço possa ser atendido entre uma par de usuários MMS. Os argumentos deste serviço incluem o tamanho máximo da mensagem que o sistema deve suportar, o número máximo de pedidos de serviços confirmados pendentes e o aninhamento da estrutura de dados.
- **Conclude**: este serviço permite a um usuário MMS encerrar a comunicação com outro usuário MMS de forma negociada. É utilizado quando o usuário MMS não tem mais pedidos de serviço a emitir, pois concluiu todos os pedidos de serviços planejados.
- **Abort**: este serviço é utilizado por um usuário MMS para abandonar o contexto MMS imediatamente, de forma abrupta e sem negociação. O provedor de serviço pode também emitir o serviço *Abort*, neste caso os dois usuários são comunicados, quando possível, através de uma primitiva de Indicação deste serviço.
- **Cancel**: este serviço permite a um usuário MMS cancelar pedidos pendentes, isto é, de serviços que foram emitidos mas não foram completados através da recepção de uma primitiva de resposta para este serviço. É somente possível para serviços confirmados.
- **Reject**: é um serviço inicializado pelo provedor do serviço para notificar a ocorrência de erros de protocolos a um usuário MMS.

Serviços de Suporte do VMD

- **Status**: permite ao usuário MMS determinar as condições gerais de entidade respondedora, através da obtenção do estado do VMD.
- **UnsolicitedStatus**: este serviço, que é do tipo não confirmado, é usado pelo usuário MMS para relatar espontaneamente o seu estado.
- **GetNameList**: permite ao usuário MMS obter a lista (ou parte dela) dos nomes dos objetos definidos no VMD.
- **Identify**: este serviço é usado pelo usuário para obter informações de identificação do usuário MMS respondedor.
- **Rename**: este serviço permite a um usuário MMS modificar o nome de um objeto definido no VMD.

Serviços de Gerenciamento de Domínio

Os domínios são manipulados pelos usuários clientes MMS através de um conjunto de serviços definido no servidor MMS e que realizam operações tais como: a transferência de domínio do cliente ao servidor (*Download*), obtenção de domínio do servidor pelo cliente (*Upload*), carga pelo servidor de um domínio a partir de um arquivo, o salvamento pelo servidor de um domínio num arquivo, a destruição pelo servidor de um domínio, a determinação dos atributos do domínio.

Os domínios são manipulados pelo Cliente MMS, através dos seguintes serviços definidos no servidor MMS e que operam sobre este objeto:

- **InitiateDownLoadSequence**: permite ao usuário cliente começar o processo de transferência de uma imagem de programa executável para um servidor MMS (*Download*).
- **DownLoadSegment**: permite ao usuário servidor obter elementos da imagem carga *Download*.
- **TerminateDownLoadSequence**: permite ao usuário MMS terminar o serviço *Download*.

- **InitiateUpLoadSequence**: permite a um cliente MMS começar o processo de transferência de uma imagem de programa executável do servidor MMS (*UpLoading*).
- **UpLoadSegment**: permite a um cliente MMS obter um segmento do *UpLoad* do servidor MMS.
- **RequestDomainDownLoad**: permite ao servidor MMS pedir um servidor de arquivo subordinado, para começar uma função de *Download*.
- **RequestDomainUpLoad**: permite a um servidor MMS pedir um servidor de arquivo subordinado, para começar uma função *UpLoad*.
- **LoadDomainContent**: permite a um cliente MMS pedir que o servidor MMS tome a ação de carregar um domínio. Esta carga pode originar do próprio arquivo do servidor, ou pode levar o servidor a pedir serviços de um servidor de arquivo subordinado.
- **StoreDomainContent**: permite a um cliente MMS pedir que o servidor MMS tome a ação de transferir um conteúdo de Domínio a um arquivo (realizar um *UpLoad*).
- **DeleteDomain**: permite a um cliente MMS pedir que o servidor MMS exclua o domínio especificado e faça os seus recursos disponíveis.
- **GetDomainAttribute**: permite a um cliente MMS pedir que o servidor MMS providencie uma lista dos atributos do domínio especificado.
- **ObteinFile**: permite a um cliente MMS pedir que o servidor MMS tome as ações apropriadas para adquirir um arquivo nomeado para seu local. Este arquivo nomeado é para ser adquirido ou do cliente MMS ou de um servidor de arquivo subordinado.

Serviços de Gerenciamento de Invocação de Programa

O curso de invocações de programa são direcionados pelo cliente MMS através dos seguintes serviços:

- **CreateProgramInvocation**: este serviço é usado pelo cliente para criar um novo objeto Invocação de Programa em VMD.
- **DeleteProgramInvocation**: este serviço é usado pelo cliente para excluir um objeto Invocação de Programa em VMD.
- **Start**: este serviço é usado pelo cliente para levar uma Invocação de Programa previamente definida no estado *RUNNING*.
- **Stop**: este serviço é usado pelo cliente para passar uma Invocação de Programa do estado *RUNNING* ao estado *STOPPED*.
- **Resume**: este serviço é usado pelo cliente para passar uma Invocação de Programa que está no estado *STOPPED* para o estado *RUNNING*.
- **Reset**: este serviço é usado pelo cliente para passar uma Invocação de Programa de estado *STOPPED* para o estado *IDLE*.
- **Kill**: este serviço é usado pelo cliente para terminar uma Invocação de Programa, colocando-o no estado *UNRUNNABLE*.
- **GetProgramInvocationAttribute**: este serviço é usado pelo cliente para determinar os atributos de uma Invocação de Programa, seu estado, sua lista de domínios pendentes, e seu tempo de vida.

Serviços de Acesso Variável

Serviços usados pelo MMS Cliente:

- **Read**: para ler o conteúdo de uma ou mais variáveis MMS.
- **Write**: para atualizar o conteúdo de uma ou mais variáveis MMS.
- **InformationReport**: pedido pelo usuário cliente ou usuário servidor (VMD) para informar outro usuário MMS (servidor ou cliente) o valor de uma ou mais variáveis especificadas, como lido pelo usuário MMS requisitor.
- **GetVariableAcessAttributes**: retorna os atributos de um objeto de acesso à variáveis do VMD.
- **DefineNamedVariable**: criar um objeto variável nomeado no VMD.

- **DefineScatteredAcess**: cria um objeto de acesso *Scattered*, cujos componentes são objetos variáveis nomeadas, não nomeadas, ou mesmo acesso *Scattered*.
- **GetScatteredAcessAttributes**: retornar os atributos de Objeto de acesso *Scattered*.
- **DefineNameVariableList**: cria um objeto Lista de Variável nomeada.
- **GetNamedVariableListAttributes**: para pedir os atributos de um objeto lista de variável nomeada definido no VMD.
- **DeleteVariableAcess**: para excluir um ou mais objetos de acesso a variável nomeada ou *Scattered.*
- **DefineNameType**: para armazenar uma especificação do tipo nomeado para uso em definições posteriores de variáveis ou tipos.
- **DetNameTypedAttributes**: para pedir os atributos de um Objeto tipo nomeado.
- **DeleteNamedType**: para excluir um ou mais objetos do tipo nomeado

Serviço de Gerenciamento de Semáforo

Estes serviços permitem a definição e o controle de semáforos, no MMS há duas classes genéricas de semáforos:

- Semáforo "ficha"(*token*): permitindo simples ou múltiplos proprietários;
- Semáforo "*pool*": permitindo uma alocação dinâmica ou explícita de fichas nomeadas.

Um semáforo é modelado como um servidor, uma lista de proprietários e uma fila de requisitores. A diferença entre os dois tipos de Semáforos é a identidade das fichas, que existe no semáforo *pool* através de um nome individualizado para cada ficha.

Os seguintes serviços são usados pelos usuário MMS para operarem sobre esses objetos:

- **TakeControl**: permite obter o controle de um semáforo.

- **RelinquishControl**: libera o controle de um semáforo obtido.
- **DefineSemaphore**: cria o semáforo *Token* no usuário MMS respondedor.
- **DeleteSemaphore**: permite excluir um semáforo.
- **ReportSemaphoreStatus**: permite obter o estado sumarizado de um semáforo.
- **ReportPoolSemaphoreStatus**: permite obter o nome e o estado de *Tokens* nomeados controlados pelo semáforo *Pool*.
- **ReportSemaphoreEntryStatus**: permite obter o nome e o estado detalhado da lista de proprietários e requisitores relacionados a um semáforo.

Serviços de Comunicação com o Operador

São serviços que permitem a comunicação com o operador da estação através da saída e entrada de dados. Para isso é utilizado o objeto Estação Operador que descreve como trabalham os seguintes serviços:

- **Input**: permite a um usuário MMS obter dados de estação operador, com opção de ecoá-los no terminal de saída.
- **Output**: permite a um usuário MMS escrever uma mensagem para a estação operador.

Serviços de Gerenciamento de Eventos

- **DefineEventCondition**: para criar um objeto condição de eventos.
- **DeleteEventCondition**:para excluir um ou mais objetos de condição de Eventos definidos no MMS
- **GetEventConditionAttributes**: permite obter os atributos descritivos de uma condição de evento.
- **ReportEventConditionStatus**: para obter o estado de uma condição de evento.

- **AlterEventConditionMonitoring**: para alterar qualquer combinação dos atributos de uma condição de evento monitorada, como *Enable, Priority* e *Alarm Summary Reports*.
- **TriggerEvent**: para disparar um evento associado com uma condição de evento.
- **DefineEventAction**: para criar um objeto ação de evento num VMD.
- **DeleteEventAction**: para excluir um ou mais objetos Ação de Evento definidos em um VMD.
- **GetEventActionAttributes**: para obter os atributos de uma ação de eventos definidos no VMD.
- **ReportEventActionStatus**: para obter o número de registros de eventos que só especificam uma ação de evento definido no VMD.
- **DefineEventEnrollment**: para que o VMD adicione o cliente requisitor ou um cliente subordinado, na lista de usuários para os quais os pedidos de serviços *EventNotification* resultante de uma condição de evento específico estão para ser enviados.
- **DeleteEventEnrollment**: para pedir um ou mais Registros de Eventos.
- **GetEventEnrollmentAttributes**: para uma lista de notificação de registros de eventos que satisfazem um conjunto especificado de critérios.
- **ReportEventEnrollmentStatus**: para obter o estado de uma simples notificação de registro de evento.
- **AlterEventEnrollment**: para recolocar os atributos *EventConditionTransitions* e/ou *AlarmAcnowLedEventRule* de um objeto Registro de Evento referenciado na condição de evento monitorada.
- **EventNotification**: é um serviço não confirmado utilizado para notificar a um cliente registrado da ocorrência de uma transição de estado associado com a condição de evento.
- **AcknowledgeEventNotification**: para notificar um VMD que o usuário reconheceu uma notificação de evento recebido do VMD.

- **GetAlarmSumary**: para obter informações do VMD sobre o estado corrente de condições de eventos com o atributo *AlarmSummaryRequest* verdadeiro.
- **GetAlarmEnrollmentSumary**: para obter um resumo do VMD sobre o estado corrente de um registro de evento tendo uma *AlarmAcnowLedEventRule* diferente de *None*.

Serviços de Gerenciamento de Ocorrências (Jornal)

Serviços usados pelo usuário MMS cliente:

- **ReadJournal**: para recuperar as informações de uma ocorrência especificada.
- **WriteJournal**: para atualizar as informações no registro de ocorrências.
- **InitializeJournal**: para inicializar todo ou parte de um registro de ocorrências ao estado de vazio.
- **ReportJournalStatus**: para determinar o número de entradas em um registro de ocorrências.

Serviços de Gerenciamento de Arquivos

Estes serviços permitem a manipulação de pequenos arquivos seqüenciais em aplicações industriais.

- **FileOpen**: permite abrir um arquivo para ser lido.
- **FileRead**: para transferir parte ou todo conteúdo de um arquivo aberto do servidor para o cliente MMS.
- **FileClose**: permite liberar os recursos associados com a transferência de arquivos.
- **FileRename**: permite trocar o nome de um arquivo virtual no servidor MMS.

- **FileDelete**: permite excluir um arquivo do servidor MMS.
- **FileDirectory**: permite obter os nomes e os atributos de um arquivo ou grupo de arquivos no servidor MMS.

Glossário

O domínio de tecnologias industriais e informáticas é fortemente marcado pela utilização de siglas em inglês já relativamente estabelecidas no meio acadêmico e no mercado. Com o intuito de facilitar a compreensão do leitor, foi criado um glossário dos termos utilizados na dissertação com as traduções de alguns dos termos em inglês e com as siglas mais usadas no texto.

A

AI – *Artificial Intelligence* – Inteligência Artificial
AMS – *Agent Management Specification*
ASCII – *American Standard Code for Information Exchange* – Código Americano Padrão para a Troca de Informações

B

Black Board – Quadro Negro
BNF – *Bakus-Naur Form*

C

CASE – *Computer Assisted Software Engineering*
CFM – Célula Flexível de Manufatura
CIM – *Computer Integrated Manufacturing* – Produção Integrada por Computador
CLP – Controlador Lógico Programável
Companion Standards – padrões contendo os detalhes e aspectos específicos de aplicação para cada equipamento não coberto pelo padrão MMS genérico.
CNC – Comando Numérico Computadorizado
CORBA – *Common Object Request Broker Architecture*

D

DAI – *Distributed Artificial Intelligence* – Inteligência Artificial Distribuída
DARPA – *Defense Advanced Research Projects Agency* – Agencia de Projetos e Pesquisa de Defesa Avançada do Departamento de Defesa dos Estados Unidos
DLL – *Dynamic Linking Lybrary*
DOM – *Document Object Model* – Modelo de Objeto de Documento
DTD – *Document Type Definition* – Definição do Tipo de Documento

F

FIP – *Factory Instrumentation Protocol*, proposta francesa
FIPA – *Foundation for Intelligent Physical Agents* – Fundação para Agentes Físicos Inteligentes
FIPA-ACL – *FIPA Agent Communication Language* – Linguagem de Comunicação FIPA

FT – *Faul Tolerance* – Tolerância a Falhas

G
GIOP – *Generic Inter-ORB Protocol* – Protocolo Inter-ORB Genérico
GML – *Generealized Markup Language* – Linguagem de Marcação Generalizada

H
HDLC – *High Level Data Link Control*
HTML – *Hyper Text Markup Language*

I
IDL – *Interface Definition Language* – Linguagem de Definição de Interface
IEC – *International Electrotechnical Comission*
IEEE – *Institute of Electrical and Electronics Engineers*
IIOP – *Internet Inter-ORB Protocol* – Protocolo Inter-ORB Internet
ISA – *Instrumentation Society of America*

K
KIF – *Knowledge Interchange Format* – Formato para Intercâmbio de Conhecimento
KQML – *Knowledge Query Manipulations Language*

L
LAN – *Local Area Network* – Rede de Área Local
LLC – *Logical Link Control*

M
MAC – *Medium Access Control*
MAP – *Manufacturing Automation Protocol* – Protocolo de Automação da Produção
MMS – *Manufacturing Message Specification* – Especificação de Mensagens Industriais
MMSI – *Manufacturing Message Specification Interface* – Interface da Especificação de Mensagens Industriais

O
OMG – *Object Management Group* – Grupo de Gerenciamento de Objetos
ORB – *Object Request Broker*
OSI – *Open Systems Interconnection* ou *Open Systems Interconnect*

P
PROFIBUS – *Process Filedbus*, proposta alemã

Q
QoS – *Quality of Service* – Qualidade de Serviço

R
RT – *Real Time* – Tempo Real
RCP – *Remote Procedure Call* – Chamada de Procedimento Remoto

S

SC^2 – *Supply Chain Smart Coordination* – Coordenação Inteligente da Cadeia de Produção

SGML – *Standard Generealized Markup Language* – Padrão de Linguagem de Marcação Generalizada

SMTP – *Simple Mail Transfer Protocol*

SQL – *Structured Query Language*

T

TCP/IP – *Transmission Control Protocol / Internet Protocol*

U

UML – *Unified Modeling Language* – Linguagem de Modelagem Unificada

V

VMD – *Virtual Manufacturing Device* – Dispositivo Virtual de Manufatura

X

XML – *Extensible Markup Language* – Linguagem de Marcação Extensível

W

W3C – *World Wide Web Consortium*

Referências Bibliográficas

ANDERL, A.; ARLT, M.; 2000. IPDM Systems. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 523-530.

AHO, A. V.; SETHI, R; ULLMAN, J. D.; 1986. Compilers: Principles, Techniques and Tools. Addison-Wesley.

ARIZA, T.; FERNÁNDEZ F. J.; RUBIO, F. R.; 2001a. Communicating to a Manufacturing Device using MMS/CORBA. In: DIGITAL ENTERPRISE CHALLENGES: LIFE-CYCLE APPROACH TO MANAGEMENT AND PRODUCTION, IFIP TC5 / WG5.2 {\&} WG5.3 ELEVENTH INTERNATIONAL PROLAMAT CONFERENCE ON DIGITAL ENTERPRISE – NEW CHALLENGES (Nov. 2001: Budapest, Hungary). *Proceedings*. Hungary 2001. p. 452-462.

ARIZA, T.; FERNÁNDEZ F. J.; RUBIO, F. R.; 2001b. Implementing a Virtual Manufacturing Device for MMS/CORBA. In: 8th IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION (EFTA'01). (Oct. 2001: Antibes, France). *Proceedings*. France, 2001.

ARIZA, T.; RUBIO, F. R.; 1998a. Communicating MMS Events in a Distributed Manufacturing System Using CORBA. In: 15th IFAC WORKSHOP ON DISTRIBUTES COMPUTER CONTROL SYSTEMS (DCCS'98). (Sep. 1998: Como, Italy). *Proceedings*. Italy 1998.

ARIZA, T.; RUBIO, F. R.; 1998b. MMS-Manager: Device Management in Heterogeneous Environment Based on CORBA. In: 3rd PORTUGUESE CONFERENCE ON AUTOMATICA CONTROL (CONTROLO'98). (Sep. 1998: Coimbra, Portugal). *Proceedings* Portugal, 1998. P. 501-506. http://www.esi2.us.es/~rubio/publica.html

AZEVEDO, A. L.; SOUZA, J. P.; OLIVEIRA, R. T.; 1999. A multi-Agent Framework For Order Negotiations in Distributed Manufacturing Enterprises – An Architecture For Distributed Diagnosis Systems; In: IFAC WORKSHOP. (2-4, Dec. 1999: Vienna, Austria) Vienna, Austria 1999. ISBN 0-08-043657-9 p. 133-138.

AZEVEDO, A. L.; SOUSA, J. P.; SOARES, A. L.; 1996. Requeriments for an Agent Based Information System Supporting Variably Coupled Networked Enterprises. In: 2nd IEEE/ECLA/IFIP INTERNATIONAL CONFERENCE ON ARCHITECTURES

AND DESIGN METHODS FOR BALANCED AUTOMATION SYSTEMS. (Jun. 1996: Portugal). Balanced Automation Systems II – Implementation Challenges for Anthropocentric Manufacturing. *Proceedings*. Portugal : Chapman & Hall, p. 137-144, ISBN 0-412-78890-X, 1996.

BARATA, J.; CAMARINHA-MATOS, L. M.; 2000. Shop Floor Reengineering to Suport Agility in Virtual Enterprise Environments. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 381-394.

BARBER, K. S.; WHITE, E; GOEL, A; at al. 1998. Dynamic, Self-Organizing, Multi-Agent Based Shop Floor Control. In: SECOND INTERNATIONAL WORKSHOP ON INTELIGENT CONTROL – THE FOURTH JOINT CONFERENCE ON INFORMATION SCIENCES – RESEARCH TRIANGLE PARK. (Oct. 1998: North Carolina, USA). *Proceedings*, USA, 1998. http://citeseer.nj.nec.com/252864.html

BENECH, D.; DESPREATS, T.; RAYNAUD, U.; 1997. A KQML-CORBA based Architecture for Intelligent Agents Communication in Cooperative Service and Network Management. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF MULTIMIDIA NETWORKS AND SERVICES (MMNS´97). (Jul. 1997: Montreal, Canada). *Proceedings*, Canada, 1997. http://citeseer.nj.nec.com/benech97kqmlcorba.html

BITTENCOURT, G.; 1998. Inteligência Artificial Distribuída. In: I WORKSHOP DE COMPUTAÇÃO DO INSTITUTO TECNOLÓGICO DE AERONÁUTICA. (6.: Out. 1998) Anais. São José dos Campos, SP.

BITTENCOURT, G.; 2001. Inteligência Artificial – Ferramentas e Teorias. Segunda Edição, Editora da UFSC – Florianópolis, 2001

BNF. http://cs.boisestate.edu/~tcole/cs354/vg/bnf.pdf em 20 de novembro de 2002.

BNF2. http://tracfoundation.org/spec/notation.htm em 28 de novembro de 2002.

BOEING; 1998. D38550-6 Communications Requirements to Boeing Network http://www.boeing.com/companyoffices/doingbiz/edaas/pdf/D38550-6_Comm.pdf

BORLAND; 2000. http://www.borland.com/

BOKMA, A; 2000. CogNet: Integrated Information and Knowledge Management and its Use in Virtual Organizations. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec.

2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 362-370.

BOOCH, G.; RAUMBAUGH, J.; JACOBSON, I.; 2000. The Unified Modelinf Language User Guide, Reading: Addison-Wesley, ISBN 0-2014-57168-4.

BRITO, L; NEVES, J.; MOURA, F.; 2000. A Mobile-Agent Based Architecture for Virtual Enterprises. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 167-174.

CAMARINHA-MATOS, L. M.; AFSARMANESH, H; RABELO, R. J.; 2000. Suporting Agility in Virtual Entreprises. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 89-104.

CARVALHO, A; TOVAR, E.; OLIVEIRA, J.; et al.; 1996. Integrations of Manufacturing Applications Overcoming Heterogeneity of Preserve Investiment. In: 2nd IEEE/ECLA/IFIP INTERNATIONAL CONFERENCE ON ARCHITECTURES AND DESIGN METHODS FOR BALANCED AUTOMATION SYSTEMS. (Jun. 1996: Portugal). Balanced Automation Systems II – Implementation Challenges for Anthropocentric Manufacturing. *Proceedings*. Portugal : Chapman & Hall, p. 191-200, ISBN 0-412-78890-X, 1996.

CHAFFEE, A.; MARTIN, B.; 1999. Introduction to CORBA; Jguru Training by the Magelang Institute, 1999, http://jGuru.com

CHANDRA, C.; SMIRNOV, A. V.; SHEREMETOV, L. B.; 2000. Agent-Based Infrastructure of Supply Chain Network Management. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 221-232.

CULLEN, F.; 2001. Bridging Legacy Data with XML. In: MYERSON, J. M. Enterprise Systems Integration. Second Edition. New York: Auerbach Publications. p. 137-142.

DEUGO, D; OPPACHER, F; ASHFIELD B.; et al.; 1999. Communication as a Means to Differentiate Objects, Components and Agents. In: TECHNOLOGY OF ORIENTED LANGUAGES AND SYSTEMS. (01-05.: Aug. 1999: Santa Barbara, California), *Proceedings*. California, 1999. – www.computer.org

DIGITAL EQUIPMENT CORPORATION; 1998. Software Product Description –

DEComni MMS http://www.compaq.com/info/SP4789/SP4789PF.PDF

DIPIPPO, L. C.; FAY-WOLFE, V.; NAIR, L; et al.; 2001; A Real-Time Multi-Agent System Architecture for E-Commerce Applications. In: FIFTH INTERNATIONAL SYMPOSIUM ON AUTONOMOUS DECENTRALIZED SYSTEMS, ISADS 2001. (26-28.: Mar. 2001: Dallas, Texas, USA), *Proceedings*. Texas, USA, 2001. P. 357-364.

FERBER, J.; 1999. Multi-Agent Systems – An Introduction to Distributed Artificial Intelligence, Reading: Adison Wesley, Londres.

FILEDBUS, 2003a. Foundation Fieldbus, Technical Overview, FD-043 Revision 3.0, Austin, Texas.

FIELDBUS, 2003b. Foundation Fieldbus, White Paper – acessado em fevereiro de 2003 – http://www.fieldbus.org/

FILHO, R. S. S.; 2000. Uma Arquitetura Baseada em CORBA para Workflow de Larga Escala. Campinas. Dissertação Apresentada Para a Obtenção do Grau de Mestre no Instituto de Computação. Universidade Estadual de Campinas.

FININ, T.; FRIZSON, R.; MCKAY, D.; 1992. A Language and Protocol to Support Intelligent Agent Interoperability. In: CE AND CALS WASHINGTON 92 CONFERENCE. (Jun. 1992). *Proceedings*.

FININ, T.; FRIZSON, R.; 1994. KQML – A Language and Protocol for Knowledge and Information Exchange, Technical Report CS-94-02, Computer Science Department, University of Maryland.

FIPA, 1997. FIPA 97 Specification, part2: Agent communication language, 1997. Geneva, Switzerland. http://www.fipa.org

FIPA, 1999. FIPA Specification 1999: Agent communication language, 1999. Geneva, Switzerland. http://www.fipa.org

FIPA, 2002 – http://www.fipa.org Acessado em 23 de outubro de 2002.

FRAGA, J.; MAZIERO, C; LUNG, L. C.; LOQUES FILHO, O. G; 1997. Implementing Replicated Services in Open Systems Using a Reflective Approach. In: 3rd INTERNATIONAL SYMPOSIUM ON AUTONOMOUS DECENTRALIZED SYSTEMS (ISADS´97). (09-11: Apr. 1997: Berlin, Germany). *Proceedings*. Berlin, Germany. p. 273.

FRICK, G.; MÜLLER-GLASER, K. D.; 2000. A Virtual Project House Infrastructure for Distributed Development Processes. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL

ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 193-202.

GRUBER, T. R.; 1992 Toward Principles of the Design of Ontologies Used for Knowledge Sharing. In: INTERNATIONAL WORKSHOP ON FORMAL ONTOLOGY. (1992: Padova, Italy). *Proceedings*. Italy, 1992.

GUYONNET, G; GRESSIER-SOUDAN, E; WEIS, F; 2001. COOL-MMS: a CORBA Approach for ISO-MMS; Cnam Laboratoire Cedric, Paris, France http://tulipe.cnam.fr/personne/gressier/ECOOP.html, acessado em 05 de março de 2001.

HEREDIA, J.A.; FAN, I.S.; ROMER, F. et al.; 1996. An Approach to Knowledge Representation and Performance Measurement for a Quality Engineering System. In: 2nd IEEE/ECLA/IFIP INTERNATIONAL CONFERENCE ON ARCHITECTURES AND DESIGN METHODS FOR BALANCED AUTOMATION SYSTEMS. (Jun. 1996: Portugal). Balanced Automation Systems II – Implementation Challenges for Anthropocentric Manufacturing. *Proceedings*. Portugal : Chapman & Hall, p. 154-162, ISBN 0-412-78890-X, 1996.

IEEE, 2003. IEEE 802 LAN/MAN Standards Committee – acessado em fevereiro de 2003 – http://www.ieee802.org

ISA, 2003 http://www.isa.org – acessado em fevereiro de 2003.

KEAHEY, K.; 2001; A Brief Tutorial on CORBA; http://www.cs.indiana.edu/hyplan/kksiazek/tuto.html acessado em 23 de outubro de 2001.

KERN, V. M.; 1997. Manutenibilidade da Semântica de Modelos de Dados de Produtos Compartilhados em Rede Interoperavel. Florianópolis. Tese Apresentada Para a Obtenção do Grau de Doutor em Engenharia– Centro Tecnológico, Universidade Federal de Santa Catarina.

KQML – Knowledge Query and Manipulation Language; http://www.csee.umbc.edu/kqml/ acessado em 10 de março de 2001.

KUSIAK, A.; HE, D.; 1996. Reengineering Manufacturing Process for Agility. In: 2nd IEEE/ECLA/IFIP INTERNATIONAL CONFERENCE ON ARCHITECTURES AND DESIGN METHODS FOR BALANCED AUTOMATION SYSTEMS. (Jun. 1996: Portugal). Balanced Automation Systems II – Implementation Challenges for Anthropocentric Manufacturing. *Proceedings*. Portugal : Chapman & Hall, p. 3-16, ISBN 0-412-78890-X, 1996.

KUSIAK, A.; 1989. Artificial Intelligence: Implications for CIM, Prentice-Hall International Editions, 1989.

LABROU, Y.; FININ, T.; PENG, Y.; 1999. The Interoperability Problem: Bringing Together Mobile Agents an Agent Communication Languages. In: 32nd HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICCS'99) (5-8.:Jan. 1999: Maui, Hawaii). *Proceedings*. Maui, Hawaii, 1999. www.computer.org

LABROU, Y.; FININ, T.; PENG, Y.; 1999b. Agent Communication Languages: The Current Landscape. IEEE Journal, v. 14, n .2 (Apr.) p. 45-52 www.computer.org

LEE, R. C.; TEPFENHART, W. M.; 1997. UML and C++ – A Pratical Guide to Object-Oriented Development, Reading: Prentice Hall, ISBN 0-13-619719-1.

LEITÃO, P. J.; MACHADO, J. M.; LOPES, J. R.; 1996. A Manufacturing Cell Integration Solution. In: 2nd IEEE/ECLA/IFIP INTERNATIONAL CONFERENCE ON ARCHITECTURES AND DESIGN METHODS FOR BALANCED AUTOMATION SYSTEMS. (Jun. 1996: Portugal). Balanced Automation Systems II – Implementation Challenges for Anthropocentric Manufacturing. *Proceedings*. Portugal : Chapman & Hall, ISBN 0-412-78890-X, 1996. p. 209-216.

LEITÃO, P. J.; RESTIVO, F; PUTNIK, G.; 2001. A Multi-Agent Based Cell Controller. In: 8th IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION. (2001: France). 2001.

LESSER, V. R.; 1999. Cooperative Multiagent Systems: A Personal View of the State of the Art. IEEE Trans. Knowledge and Data Engeneering, Jan.-Feb. Special Issue.

LUIGI; 2002. Ontology Working Group - http://www.csc.liv.ac.uk/~valli/WG-Ontology/index.html. Acessado em 05 de dezembro de 2002

LUNG, L. C.; FRAGA, J.; FARIES, J; *et al.*; 2000. Experiências com Comunicação de Grupo nas Especificações Fault Tolerant CORBA. In: XVIII Simpósio Brasileiro de Redes de Computadores – SBRC'2000 – SBC (23-26.:Mai. 2000) Belo Horizonte – MG.

MASSYVE; 2000. http://www.gsigma-grucon.ufsc.br/massyve/, acessado em 18 de novembro de 2000.

MISCHE, M. A.; 2001. Definition Systems Integration. In: MYERSON, J. M. Enterprise Systems Integration. Second Edition. New York: Auerbach Publications. p. 3-10.

MONTESCO, C. A. E.; 2001. UCL – Uma Linguagem de Comunicação para Agentes de Software. São Carlos. Dissertação Apresentada Para a Obtenção do Grau de Mestre em Ciências – Área de Ciências de Computação e Matemática Computacional Universidade de São Paulo – USP, Outubro 2001

MORRISON, M.; 2000. XML Unleashed, Reading: Sams Publishing, ISBN 0-672-31514-9 .

NODINE, M.; CHANDRASEKARA, D.; 1999. Agent Communication Languages for Information-Centric Agent Communities. In: 32nd HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICCS'99) (5-8.:Jan. 1999: Maui, Hawaii). *Proceedings*. Maui, Hawaii, 1999. www.computer.org.

NOY, N. F.; McGUINNESS, D. L.; 2002 Ontology Development 101: A Guide to Creating Your First Ontology- Stanford University, Stanford, CA, 94305 http://ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html. Acessado em 06 de dezembro de 2002.

O'BRIEN, J.A.; 2002. Sistemas de Informação e as Decisões Gerenciais na Era da Internet; tradução Cid Knipel Moreira. São Paulo: Editora Saraiva. ISBN 85-02-03276-3.

OBELHEIRO, R. R.; FRAGA, J.; WESTPHALL, C. M.; 2001. Controle de Acesso Baseado em Papéis para o Modelo CORBA de Segurança. In: 19° SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES. (21-25 Mai. 2001: Florianópolis). Anais. Florianópolis, Santa Catarina.

OSÓRIO, A. L.; GIBON, P.; BARATA, M . M.; 2000. Electronic Commerce With XML/EDI in Virtual Enterprises. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 311-324.

PINHO, A. N.; 1988. Um Estudo da Especificação de Mensagem da Manufatura (MMS) do Protocolo de Comunicação para Ambientes Industriais MAP. Florianópolis. Dissertação (Mestre em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

PIMENTEL, M. G. C.; TEIXEIRA, C. A. C.; SANTANCHÉ, A.; 2000. XML:Explorando suas Aplicações na Web. In: JAI 2000-SBC, XIX Jornada de Atualização em Informática.

PIRES, A.; 1997. CORBA: Um Padrão Para Objetos Distribuídos. Developers Magazine, Axel Books. São Paulo, n. 12, Ano I. (Ago).

QUINTAS, A.; LEITÃO, P.; 1997. A Cell Controller Arcgitecture Solution: Description and Analysis of Performance and Costs. In: ISIP'97, OE/IFIP/IEEE CONFERENCE ON INTEGRATED AND SUSTAINABLE INDUSTRIAL PRODUCTION. Chapman & Hall (Re-engineering for Sustainable Industrial Production). (May. 1997: Lisbon, Portugal) *Proceedings*. Lisbon, Portugal, 1997. ISBN 0-412-79950-2. p. 182-195.

RABELO, R. J.; CAMARINHA-MATOS, L. M.; 1994. Negotiation in Multi-Agent

Based Dynamic Scheduling. Robotics and Computer-Integrated Manufacturing – An International Journal, Pergamon. U. K., vol. 11 n. 4 (Dec).

RABELO, R. J.; 1997. Um Enquadramento para o Desenvolvimento de Sistemas de Escalonamento Ágil da Produção – Uma Abordagem Multiagente. Lisboa. Dissertação Apresentada Para a Obtenção do Grau de Doutor em Engenharia Eletrotécnica, Especialidade de Robótica e Manufactura Integrada. Universidade de Nova Lisboa, Faculdade de Ciências e Tecnologia.

RABELO, R. J.; CAMARINHA-MATOS L. M.; AFSARMANESH H.; 1998. Multiagent perspectives to agile scheduling. In BASYS'98, INTELLIGENT SYSTEMS FOR MANUFACTURING. (Ago. 98: Czech Republic) Prague, Czech Republic. (Kluwer Academic)

RABELO, R. J.; KLEN, A. P.; FERREIRA, A. C.; 2000a. For a Smart Coordination of Distributed Business Processes. In: IFIP TC5/WG5.3 FOURTH IFIP/IEEE INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY FOR BALANCED AUTOMATION SYSTEMS IN MANUFACTURE AND TRANSPORTATION (ADVANCES IN NETWORKED ENTREPRISES VIRTUAL ORGANIZATIONS, BALANCED AUTOMATION AND SYSTEMS INTEGRATION) (29.: Sep. 2000: Berlin, Germany). *Proceedings*. Berlin, Germany. ISBN 0-7923-7958-6 p. 81-90.

RABELO, R. J.; CAMARINHA-MATOS, L. M.; VELLEJOS, R. V.; 2000b. Agent-Based Brokerage For Virtual Enterprise Creation in the Moulds Industry. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 281-290.

RABELO, R. J.; CAMARINHA-MATOS, L. M.; VELLEJOS, R. V.; 2000c. Federated Multi-Agent Scheduling in Virtual Enterprises. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil..

RABELO, R. J.; 2001. Interoperating Standards in Multiagent Agile Manufacturing Scheduling Systems. International Journal of Computer Applications in Technology (IJCAT), ISSN 0952 8091.

RABELO, R. J.; KLEN, A. A. P.; KLEN, E. R.; 2002. A Multi-Agent System for Smart Coordination of Dynamic Supply Chains. In: IFIP TC5/WG5.5 THIRD WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE´02). (1-3.: May. 2002: Sesimbra, Portugal). *Proceedings*. Sesimbra,

Portugal. p. 379-386.

RABUSKE, R. A.; 1995. Inteligência Artificial. Editora da UFSC – Florianópolis, 1995

SHANG, H.; ZHAO, Z.; THORN, R.; *et al*.; 2001. A Web-Based MMS-Enabled Virtual Environment System. In: IMC-19 - 19th International Manufacturing Conference Manufacturing The Future Through Innovation And Research (28-30.: Aug. 2002: Belfast, Ireland). *Proceedings*. Belfast, Ireland.

SIEGEL, J.; 2000. CORBA3 Fundamentals and Programming. Second Edition. Reading: John Wiley & Sons, Inc. ISBN 0-471-29518-3

SILVEIRA, J. L.; 1991. Estudo da Utilização do Padrão MMS em Aplicações Fabris. Florianópolis. Dissertação (Mestre em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

SMITH, S. S.; HOBERECHT, W. C.; JOSHI, S. B.; 1995. *IEEE Transactions on Robotics and Automation*, Vol. 11, No. 4, August 1995, p. 558-570.

SINGH, M. P.; 1998 Agent Communication Languages: Rethinking the Principles. IEEE Journal, v. 31, n .12 (Dec.) p. 40-47. www.computer.org

SISCO; 1995. System Integration Specialists Company, Inc. 1995. An Explanation of the Architecture of the MMS Standard. 1995. USA. http://www.sisconet.com

SISCO; 1996. System Integration Specialists Company, Inc. – Device GateWay; 6605 19 Mile Road, Sterling Heights, MI 48314-1408, USA. http://www.sisconet.com

SISCO; 1998a. System Integration Specialists Company, Inc. – MMS-EASE Reference Manual; 6605 19 Mile Road, Sterling Heights, MI 48314-1408, USA. http://www.sisconet.com

SISCO; 1998b. System Integration Specialists Company, Inc. – AX-sS4 MMS; 6605 19 Mile Road, Sterling Heights, MI 48314-1408, USA. http://www.sisconet.com

SISCO; 1998c. System Integration Specialists Company, Inc. – MMS-EASE Lite for Embedded Systems; 6605 19 Mile Road, Sterling Heights, MI 48314-1408, USA. http://www.sisconet.com

SOUSA, J. P.; AZEVEDO, A. L.; SOARES, A. L.; 2000. A Conceptual Framework For Aggregate Management of Virtual Entreprises. In: IFIP TC5/WG5.3 SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brazil). *Proceedings*. Florianópolis, Brazil. p. 395-402.

TAJKUMAR, T. M.; LEWIS, R. J. Jr; 2001. Evaluating Object Middleware: DCOM and CORBA. Enterprise Systems Integration. Second Edition. New York: Auerbach Publications. p. 143-158.

TAO; 2001. http://www.tenermerx.com/programming/corba/tao_bcb/index.html; acessado em julho de 2001.

THURAISINGHAM, B.; SPAR, D. L.; 2001. Using CORBA to Integrate Database Systems. In: MYERSON, J. M. Enterprise Systems Integration. Second Edition. New York: Auerbach Publications. p. 261-270.

UNIVERSITY OF MARYLAND; 1994 KQML - A language and protocol for knowledge and information exchange 1994. Technical Report CS-94-02, Computer Science Department University of Maryland and Valley Forge Engineering Center, Unisys Corporation.

USHER, J. M.; WANG, Y.; 2000. Negotiation Between Intalligent Agents for Manufacturing Control. In: EDA 2000 Conference, (Orlando, Florida, USA). *Proceedings*. Orlando, Floriada. 2000. http://citeseer.nj.nec.com/usher00negotiation.html

VERÍSSIMO, P; MELRO, S.; CASIMIRO, A.; et al. 1996. Distributed Industrial Information Systems: Design and Experience. In: 2nd IEEE/ECLA/IFIP INTERNATIONAL CONFERENCE ON ARCHITECTURES AND DESIGN METHODS FOR BALANCED AUTOMATION SYSTEMS. (Jun. 1996: Portugal). Balanced Automation Systems II – Implementation Challenges for Anthropocentric Manufacturing. *Proceedings*. Portugal : Chapman & Hall, p. 137-144, ISBN 0-412-78890-X, 1996.

VIEIRA, W. J. M.; 2000. Agentes Móveis Adaptáveis para Operação Remota. Lisboa. Dissertação apresentada para a obtenção do Grau de Doutor em Engenharia Electrotécnica, Especialidade de Robótica e Manufatura Integrada. Universidade de Nova Lisboa, Faculdade de Ciências e Tecnologia.

W3C; 2000. XML Schema Part 0: Primer. W3C Working Draft, 7 April 2000. [Online] http://www.w3.org/TR/xmlschema-0/.

WEINSTEIN, P. C.; BIRMINGHAM, W. P.; DURFEE, E. H.; 1999. Agent-Based Digital Libraries: Descentralization and Coordination. IEEE Communications Magazine. (Jan.) 1999 – 0163-6804/99.

WEN, W.; MIZOGUCHI, F.; 1999. Analysis and Verification of Multi-Agent Interaction. In: SIXTH ASIA PACIFICA ENGINEERING CONFERENCE. (07-10.: Dec. 1999: Takamatsu, Japan). *Proceedings*. Takamatsu, Japan. p. 252.

WILLRICH, R.; 1991. Uma Proposta de um Modelo de Implementação de Serviços de Apoio a Aplicações Industriais Segundo o Padrão MMS. Florianópolis. Dissertação

(Mestre em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

WOBBE, W.; 1996. Antropocentric Productions Systems are Socio Technological Innovations. In: 2nd IEEE/ECLA/IFIP INTERNATIONAL CONFERENCE ON ARCHITECTURES AND DESIGN METHODS FOR BALANCED AUTOMATION SYSTEMS. (Jun. 1996: Portugal). Balanced Automation Systems II – Implementation Challenges for Anthropocentric Manufacturing. *Proceedings*. Portugal : Chapman & Hall, ISBN 0-412-78890-X, 1996.

WOOLDRIDGE, M.; 1998a. A Knowledge-theoretic Approach to Distributed Problem Solving. In: 13th EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, (23-28.: Aug. 1998). *Proceedings*. Brighton, UK, John Wiley and Sons, Chichester, p. 308-312.

WOOLDRIDGE, M.; JENNINGS, N. R.; 1998b. Pitfalls of Agent-Oriented Development. In: 5th INTERNATIONAL WORKSHOP, ATAL '98. (4-7.: Jul. 1998). *Proceedings* Paris, France, p. 385-391.

ZAPTHINK; 2001. The Pros and Cons of XML, Technical Report. http://www.zapthink.com/reports/proscons-view.html, acessado em 13 de janeiro de 2003.