Paulo Marcelo Porto Alves Blanco

Arquitetura de um sistema MES baseado em CORBA

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia

São Paulo
2002

Paulo Marcelo Porto Alves Blanco

Arquitetura de um sistema MES baseado em CORBA

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

Área de Concentração:

Engenharia Mecânica – Automação e Sistemas

Orientador:

Prof. Dr. Marcos R. P. Barretto

São Paulo

2002

AGRADECIMENTOS

Ao amigo e orientador Prof. Dr. Marcos R. P. Barretto, pelos valiosos conselhos, colaboração e apoio;

À minha esposa, pela paciência e compreensão;

A meus pais, pelo constante incentivo;

Ao pessoal do GU-CORBA da Sucesu, pela grande ajuda.

Resumo

Os sistemas integrados de manufatura sempre tiveram na heterogeneidade de software e hardware, entre os diversos fabricantes, um obstáculo à sua implementação satisfatória. Atualmente, novas técnicas de Sistemas Distribuídos e Abertos surgem proporcionando um novo potencial de integração, agilizando a implantação dos sistemas MES (Manufacturing Execution Systems) e simplificando sua integração com outros sistemas corporativos.

Fazendo uso da arquitetura CORBA de objetos distribuídos, como a infra-estrutura básica sobre a qual um framework MES pode ser desenvolvido, este trabalho descreve inicialmente a abrangência dos sistemas MES, para então adaptar um de seus modelos a esta infra-estrutura como forma de comprovar a utilidade da arquitetura de objetos distribuídos na integração de sistemas de manufatura.

Abstract

Integrated manufacturing systems have always found, in the hardware and software heterogeneity among different vendors, an obstacle to their satisfactory implementation. Nowadays, new Open and Distributed Systems technologies have appeared, providing a new integration potential, accelerating the implementation of MES (Manufacturing Execution Systems) and making it's integration to other corporate systems simpler.

Using the CORBA architecture for distributed objects as the basic infrastructure upon which a MES framework can be developed, this work at first describes the scope of the MES in order to, later, adapt one of it's models to the CORBA infrastructure, as a way of proving it's applicability to the integration of manufacturing systems.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

Lista de Abreviações e Siglas

BOA – Basic Object Adapter
CEP – Controle Estatístico de Processos
CIM – Computer Integrated Manufacturing
CORBA – Commom Object Request Broker Architecture
DCE – Distributed Computing Enviroment
DDC – Sistemas Digitais de Controle Direto
DNA – Distributed interNet Applications
ERP – Enterprise Resource Planning
IHM – Interface Homem-Máquina
ISO – Internetional Standards Organization
LIMS – Laboratory Information Management Systems
MES – Manufacturing Execution Systems
OLE – Object Linking and Embedding
OMA – Object Management Architecture
OMG – Object Management Group
OO – Orientado a Objetos
ORB – Object Request Broker
ORPC – Object Remote Procedure Call
MTS – Microsoft Transaction Services
POA – Portable Object Adapter
RPC – Remote Procedure Call
SEMATECH – SEmiconductor MAnufacturing TECHnology
S.O. – Sistema Operacional
XML – Extensible Markup Language

1 Introdução

Durante a última década, muitas indústrias ao redor do mundo implementaram seus sistemas de planejamento e gestão (*Enterprise Resource Planning* - ERP's). Os sistemas ERP fornecidos por empresas como PeopleSoft, Baan, JD Edwards e SAP melhoram a integração entre as aplicações das empresas, combinando processos comuns de negócio em um único sistema, o que inclui finanças, gerenciamento de materiais e planejamento de produção [MELLO, 2002a] [BROWN, 2000]. Os pacotes ERP são utilizados em conjunto com técnicas de reengenharia de negócios, para atualizar e aprimorar grande parte dos sistemas de suporte das empresas [BROWN, 2000].

Entretanto, enquanto os sistemas de gestão corporativa estão naturalmente voltados a objetivos de longo prazo, surge uma tendência crescente de percepção de que os MES podem ter um impacto considerável nos objetivos de curto prazo, bem como no desempenho e na capacidade da empresa de atingir seus objetivos [OMG, 1997a].

A necessidade de integração entre o chão-de-fábrica e a gestão do negócio é impulsionada por quatro aspectos relacionados: a necessidade de melhorar o tempo de resposta às necessidades do cliente, o imperativo da flexibilização da produção, a redução dos custos e a melhora da qualidade [MESA, 1997c]. Para melhorar o atendimento, as fábricas precisam gerenciar sua qualidade, seus custos, e seus cronogramas de entregas sob condições variáveis. Seus gerentes precisam ser capazes de coordenar novos pedidos, com novos requisitos, com sua disponibilidade de equipamentos e as características de seus materiais para alcançar a flexibilidade necessária.

Há também uma percepção crescente de que a produtividade ganha quando se usam a Internet e suas tecnologias para disponibilizar informações de maneira mais rápida e de um modo mais eficiente ao longo da empresa [ROY, 2002].

Exemplos de problemas que podem ser minimizados com a integração da fábrica incluem o gerenciamento e coordenação das atividades de produção, a alocação

eficiente de recursos, a adequação a normas e regulamentos de mercados específicos (como o farmacêutico) e a redução do tempo de ciclo de produtos [OMG, 1996a].

Um dos desafios da indústria moderna é a integração dos processos e das informações de manufatura com outros processos da empresa. Muitos ambientes de manufatura consistem de um conjunto de diversas aplicações que gerenciam diferentes atividades da empresa e que foram criadas de forma vertical, coexistindo lado a lado, mas sem integração horizontal. Isto significa que informação similar pode ser gerenciada de maneira redundante e incompatível entre as diversas aplicações, como controle de chão-de-fábrica e gerenciamento de pedidos. A fim de se poder gerenciar bem uma empresa, a informação deve ser compartilhada e fluir suavemente entre os processos do negócio [OMG, 1996a].

Dessa forma, fica evidente a necessidade de solução para a questão da integração do planejamento com a manufatura, derivada diretamente dos investimentos feitos pela indústria na aquisição de sistemas corporativos e do desejo da indústria de reduzir, cada vez mais, prazos e aumentar sua produtividade e flexibilidade. Para preencher esse espaço, a tecnologia adequada é a dos sistemas de execução de manufatura (*Manufacturing Execution Systems* – MES).

Os MES consistem em uma camada intermediária entre o chão-de-fábrica e o sistema de gestão corporativo e são os responsáveis por viabilizar a integração entre, por exemplo, o cadastramento de um novo pedido e a sua inserção no plano de produção de uma determinada máquina [MESA, 2001a]. Embora o conceito básico dos MES já seja conhecido há vários anos, a viabilização de sua aplicação com maior propriedade iniciou-se a partir de três motivadores básicos: a popularização dos ERP's, o surgimento de plataformas de hardware e software adequadas ao uso de PC's para controle e supervisão da produção e o advento do conceito de *e-manufacturing* [KAMAL, 1998].

Produtos MES focados na integração do controle supervisório com os ERP's, funcionando sobre plataformas PC e disponibilizando consultas, controle e gerenciamento através da Internet já são uma realidade e estão disponíveis para a aquisição como produtos de prateleira, voltados ao mercado da automação industrial.

Esses produtos são fornecidos sob a forma de componentes de software que implementam as diversas funções a serem exercidas por um sistema MES, como [MESA, 1997b]:

- Alocação de recursos e status;
- Agendamento de atividades e produção;
- Alocação de unidades de produção;
- Controle de documentação;
- Coleta / aquisição de dados;
- Gerenciamento de mão-de-obra;
- Controle e gestão de qualidade;
- Gerenciamento de processos;
- Gerenciamento de manutenção;
- Rastreamento de produção;
- Análise de desempenho.

Dessa forma, é importante esclarecer o seguinte: o que uma empresa adquire quando compra a licença de uso de um pacote MES é, na verdade, um conjunto de ferramentas e componentes que possibilitam que se crie seu próprio MES, adequado às particularidades de seu ambiente produtivo e de seu método de gestão corporativo.

A quase totalidade dos pacotes de ferramentas MES disponíveis hoje no mercado baseia-se na arquitetura definida pelos produtos desenvolvidos pela multinacional americana de software Microsoft [HOSKE, 1998]. Na verdade, o próprio desenvolvimento do mercado de software para controle supervisório tem estado até hoje amarrado à disponibilidade de soluções baseadas no sistema operacional Microsoft Windows. Atualmente, o sistema Windows NT (ou suas versões mais recentes, o Windows 2000 e o Windows XP Professional) é o sistema operacional dominante no controle supervisório, em função de suas características de multitarefa, operação em 32 bits e extensões para ambientes tempo real e determinísticos. Todas

essas características juntas possibilitam comunicação em tempo real entre componentes de hardware e software, essencial a aplicações industriais.

Une-se a esse fato a divulgação do modelo (ou *framework*) desenvolvido há alguns anos pela Microsoft para componentes em aplicações distribuídas, conhecido originalmente como arquitetura DNA (*Distributed interNet Applications*), que possibilitou um grau mais elevado de compatibilidade entre componentes de fornecedores diferentes [HOSKE, 1998] e constituiu a base para o desenvolvimento de uma nova tecnologia específica para a comunicação de dados no ambiente de manufatura, a tecnologia OPC (*OLE for Process Control*). A OPC será apresentada e analisada em maior profundidade nas seções subseqüentes do presente trabalho.

Entretanto, a necessidade de se ater a uma arquitetura única para a implementação de sistemas MES é uma ilusão. Já existem disponíveis arquiteturas abertas, elaboradas por organizações independentes, que viabilizam exatamente o mesmo grau de interoperabilidade e flexibilidade com as vantagens inerentes da independência de fornecedor. Uma delas é a arquitetura CORBA (*Common Object Request Broker Architecture*), desenvolvida pela OMG (*Object Management Group*) e que propõe um modelo adequado ao desenvolvimento da interoperabilidade entre objetos de software independentemente da plataforma em que eles sejam executados, da linguagem em que eles sejam implementados e do fornecedor que os tenha desenvolvido.

Combinando-se o potencial dessa arquitetura para objetos distribuídos a ferramentas modernas como a linguagem Java (independente de plataforma) e o advento do uso dos PC's no chão-de-fábrica, dispõe-se de toda a infra-estrutura necessária para a implementação de soluções MES totalmente independentes de plataforma ou sistema operacional, que não precisam ficar restritas ao ambiente e filosofia computacional PC / Windows, mas sim serem expandidas para plataformas de maior ou menor porte conforme a necessidade, adequando-se de grandes sistemas Unix a até pequenos controladores *embedded* rodando máquinas virtuais Java leves [ATHERTON, 1998], [HIGHLANDER, 1998].

1.1 Objetivo

Desenvolvido com base nesse conceito, o objetivo do presente trabalho é apresentar um modelo básico para aplicações MES, que faça uso da arquitetura aberta CORBA para resolver os problemas de interoperabilidade entre os diversos componentes de um sistema de controle de chão-de-fábrica e um sistema de gestão empresarial. Entretanto pretende-se, neste trabalho, apresentar uma solução funcional que demonstre a viabilidade da aplicação destas tecnologias em um sistema de controle industrial, não se tencionando, em momento algum, fornecer um modelo definitivo para a implementação total de um sistema complexo como é um sistema MES. Será apresentado ainda um protótipo desenvolvido na linguagem multiplataforma Java, embora o modelo proposto seja independente da linguagem de programação escolhida para a implementação.

Conforme será discutido em maior profundidade posteriormente, o foco do modelo proposto restringe-se a duas das onze funções dos sistemas MES definidas em [MESA, 1997b]: a função de alocação de recursos e a função de coleta de dados.

A função de Alocação de Recursos e Status compreende o gerenciamento de recursos como máquinas, ferramentas, materiais e outras entidades que devem estar disponíveis para que a produção transcorra normalmente. Esta função proporciona um histórico detalhado dos recursos e assegura que os equipamentos estejam prontos para o trabalho e forneçam informações de status em tempo real.

Já a atividade de Coleta de Dados implementa a interface de dados entre os equipamentos de produção e os registros e relatórios associados a cada unidade produtiva. Os dados podem ser coletados do chão-de-fábrica manual ou automaticamente de um equipamento, proporcionando, no segundo caso, informações atualizadas de forma instantânea.

Essas duas funções podem ser, em algumas classificações, atribuídas aos chamados sistemas SCADA (*Supervisory Control and Data Acquisition*), uma categoria de sistemas mais restrita que os MES, dedicada ao interfaceamento com os dispositivos de controle e medição e a implementação de interfaces homem-máquina [BOYER,

2002]. Entretanto, uma vez que os SCADA constituem um subconjunto do conceito MES, a relevância das duas funções escolhidas para os objetivos do trabalho permanece.

Os capítulos subseqüentes apresentam a conceituação teórica necessária à compreensão dos sistemas MES e da tecnologia CORBA, usada como infra-estrutura básica para a implementação do modelo. Em seguida, será apresentado o modelo elaborado, uma relação dos resultados obtidos e uma discussão sobre esses resultados.

2 Revisão Bibliográfica

2.1 Evolução dos sistemas de automação industrial

Paralelamente à evolução dos sistemas computacionais, que vem sendo observada desde a década de 70 e que sempre foi absorvida pelos setores administrativos das empresas, pode-se identificar uma outra tendência de evolução ocorrendo no setor produtivo, ou no chão-de-fábrica dessas mesmas empresas. Na verdade, desde que os dispositivos de controle de equipamentos de produção começaram a evoluir das simples chaves liga-desliga, alguma forma de computação passou a ser necessária no ambiente industrial [CONSIDINE, 1993].

Uma visão geral da cronologia da evolução dos sistemas de automação industrial pode ser obtida na figura 1.

A partir do final da década de 50, com um projeto pioneiro elaborado pela TRW e pela Texco na refinaria de Port Arthur no Texas, abriu-se um novo mercado para a aplicação de sistemas digitais no controle de processos industriais. Entretanto, os computadores inicialmente utilizados em aplicações industriais eram demasiadamente caros, grandes, lentos e, principalmente, pouco confiáveis, o que restringia sua aplicação ao domínio dos sistemas de supervisão e cálculos não críticos de valores ideais dos parâmetros de produção.

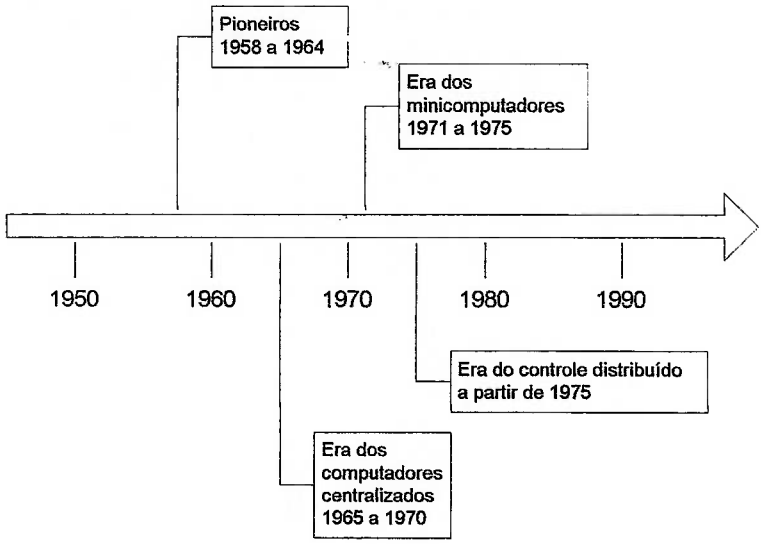

Figura 1 - Evolução do controle computadorizado

No início da década de 60 iniciou-se o uso de computadores em aplicações realmente de controle de processos, a princípio em instalações de indústrias químicas e outras situações de processos produtivos contínuos, onde se encontrou a possibilidade de substituição dos sistemas analógicos pelos sistemas digitais de controle direto (DDC).

A fim de facilitar a programação dos DDC e viabilizar sua aplicação em larga escala, desenvolveram-se linguagens especiais de programação orientadas a blocos e ao preenchimento de campos em branco, onde o operador simplesmente informava entradas, saídas, fatores de escala e outros dados em tabelas predefinidas. Todavia, a implementação de algoritmos de controle avançados ainda era muito difícil e a confiabilidade dos DDC ainda era pequena.

Com o advento da tecnologia de semicondutores no início da década de 70 e a possibilidade de se desenvolver computadores menores, mais rápidos e mais confiáveis, surgiram os chamados minicomputadores, equipamentos que possibilitavam o projeto de sistemas de controle eficientes baseados no conceito de DDC. Inicialmente os minicomputadores ainda eram pequenos demais para uso eficiente em controle de processos (Westinghouse P50, DEC PDP-8), mas logo as

deficiências foram corrigidas e modelos mais poderosos surgiram, como o DEC PDP-11, sendo que o número de instalações DDC cresceu de cerca de 5.000 em 1970 para 50.000 em 1975 [POPVIC, 1990]. Basicamente, a função dos DDC era fazer a leitura cíclica de diversos sensores, aplicar uma conversão analógica-digital e analisar os dados usando um algoritmo de controle, para então atuar sobre o processo a fim de otimizar o resultado do algoritmo de controle.

Enquanto de um lado se trabalhava no desenvolvimento dos minicomputadores, por outro lado a divisão hidramática da General Motors trabalhava em uma solução para um dispositivo programável de controle que reduzisse os custos de instrumentação e evitasse os problemas dos sistemas de controle baseados em relês. Nascia então o PLC (*programmable logic controller*).

O primeiro PLC surgiu em 1969 com uma CPU de 1K de memória e 178 pontos de E/S (Entrada/Saída). Entretanto, com o aparecimento dos minicomputadores em 1972 e a mudança radical que eles causaram na tecnologia de controle digital, os PLC's logo seriam capazes de incorporar uma maior capacidade de processamento e de gerenciar um maior número de instruções. Também a interface homem-máquina seria melhorada, através da utilização dos CRT's, que capacitavam os operadores a utilizar os diagramas tipo *ladder* para compreender o funcionamento de seus equipamentos de controle e inspecionar tanto o processo quanto o PLC.

Com esse aumento de capacidade, os PLC tornaram-se adequados não só para as aplicações seqüenciais para as quais haviam sido desenvolvidos, mas também para a realização de cálculos e tomadas de decisão.

Na década seguinte, fazendo uso das novas tecnologias de comunicação de dados, os controladores programáveis puderam ser integrados em sistemas mais complexos de automação, os chamados sistemas distribuídos, sendo que no início da década de 80 desenvolveram-se padrões internacionais a fim de possibilitar a compatibilidade de hardware e a portabilidade de software.

A evolução para os sistemas multicomputadorizados para automação industrial levaram a uma hierarquização das estruturas de controle, cuja característica

fundamental é a possibilidade de decomposição dos sistemas em uma série de níveis funcionais, entre os quais os mais comuns são o nível de controle direto, o nível supervisório da planta, o nível de *scheduling* e controle e o nível de gerenciamento da planta, onde cada qual tem as seguintes funções:

- Nível de controle direto – aquisição de dados; monitoramento de processos e de sinais de diagnóstico; funções de controle (aberto ou fechado) com base nas diretrizes do nível superior.
- Nível de supervisão – otimização de processos de controle; *loops* de controle adaptativo; otimização da coordenação entre equipamentos de produção, monitoramento de performance.
- Nível de *scheduling* e controle – logística de produção e decisões estratégicas com base em pedidos de clientes, níveis de inventário e restrições de energia e recursos, entre outras.
- Nível de gerenciamento da planta – análise de mercado, estatísticas de pedidos, cálculos de preços, despacho de pedidos, supervisão de produção e entregas e monitoramento de produtividade. Normalmente deve ser integrado com os departamentos como compras, vendas e pessoal.

Cabe aqui uma observação: a decomposição funcional em níveis hierárquicos da estrutura do sistema não precisa ser análoga à decomposição em níveis de hardware. Ou seja, pode-se ter um mesmo equipamento desempenhando papéis de dois níveis hierárquicos distintos.

Os sistemas descentralizados e hierarquicamente organizados para automação industrial, embora bastante evoluídos conceitualmente, são de implementação ainda bastante trabalhosa, em função basicamente da heterogeneidade de hardware e software presente entre os produtos dos diversos fornecedores do ramo da automação industrial, o que acaba dificultando a integração entre os diversos componentes do sistema.

Atualmente, a maneira prática de se minimizarem os problemas de interfaceamento de hardware e de compatibilidade de software é a aquisição de sistemas de um mesmo

fornecedor. Isso, entretanto proíbe que o cliente busque uma otimização da relação custo-benefício em cada subsistema, e acaba limitando a performance global do sistema. Esta é uma situação que pede por uma solução do tipo das que a arquitetura CORBA se dispõe a implementar.

2.2 Sistemas MES

2.2.1 Definição

Manufacturing Execution Systems, ou Sistemas de Execução de Manufatura, é um conceito abrangente, cujo significado dificilmente pode ser resumido em uma simples frase. Todavia, entre as diversas definições apresentadas na literatura disponível, uma das mais recorrentes é fornecida pela Manufacturing Execution Systems Association em [MESA, 2001a]: "Os Sistemas de Execução de Manufatura são sistemas que fornecem informações, possibilitando a otimização de atividades de produção do cadastro do pedido até o produto final. Utilizando dados atualizados e acurados, os MES guiam, iniciam, relatam e respondem às atividades da planta conforme elas vão ocorrendo. A rápida resposta às variações das condições operacionais resultante, aliada ao foco na redução das atividades que não agreguem valor ao produto, levam à operação eficiente dos processos da planta. Os MES melhoram o retorno dos recursos operacionais assim como possibilitam a melhora de desempenho no atendimento a prazos, gestão de estoques, margem bruta e fluxo de caixa. Os MES proporcionam informações críticas à empresa sobre atividades de produção ao longo de toda a planta e da cadeia de fornecimento, através de comunicações bidirecionais".

Como se torna claro a partir dessa definição, o termo "manufatura" da sigla MES não se restringe somente aos setores produtivos de uma empresa mas a toda a sua estrutura, uma vez que o conceito extrapola a integração de equipamentos de fabricação com sistemas de engenharia e de gerenciamento de produção. Os sistemas MES, para serem eficientes, devem fazer uso também das informações provenientes dos sistemas de gestão empresarial (ou sistemas ERP – *Enterprise Resource Planning*) e do fluxo de informações presentes ao longo de todos os setores de uma empresa.

Uma das funções em que os MES bem implementados são úteis é na criação de uma ponte entre o escritório e o chão-de-fábrica, proporcionando aos funcionários da planta dados reais e atualizados que complementam os sistemas gerenciais orientados ao escritório. Os MES, quando em conjunto com os sistemas de planejamento, fazem com que as linhas de trabalho e o plano idealizados no escritório sejam implementados no chão-de-fábrica. Fundamentalmente, o objetivo central de uma implementação de MES consiste na redução do tempo necessário para se detectar a necessidade de alteração em um processo industrial e na redução do tempo de resposta após uma tomada de decisão neste sentido.

Uma visão da localização dos sistemas MES com relação aos demais sistemas presentes nas empresas de manufatura atualmente pode ser obtida das figuras 2 e 3, utilizadas como referência pela MESA. Na definição da MESA, um MES opera como um canal de informações, que se liga e fornece dados a outros sistemas, sobrepondo-se a alguns outros tipos de solução que, por sua vez, podem sobrepor-se entre si. Por exemplo, a programação de produção pode aparecer tanto nos sistemas MES quanto aos SCM (*Supply Chain Management*); a gestão de mão-de-obra aparece nos MES e nos ERP; controle de documentação nos MES e nos P/PE (*Product and Process Engineering*) [VIRDHAGRISWARAN, 1995]. Os graus de sobreposição podem variar em função do tipo de indústria e implementação [MESA, 1997c].

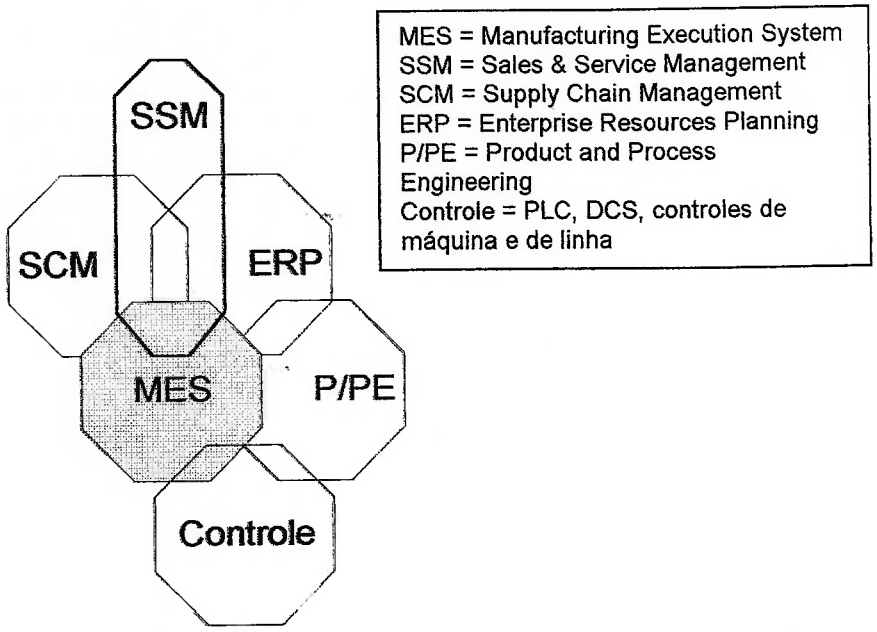

Figura 2 - Contexto dos MES nas corporações[1]

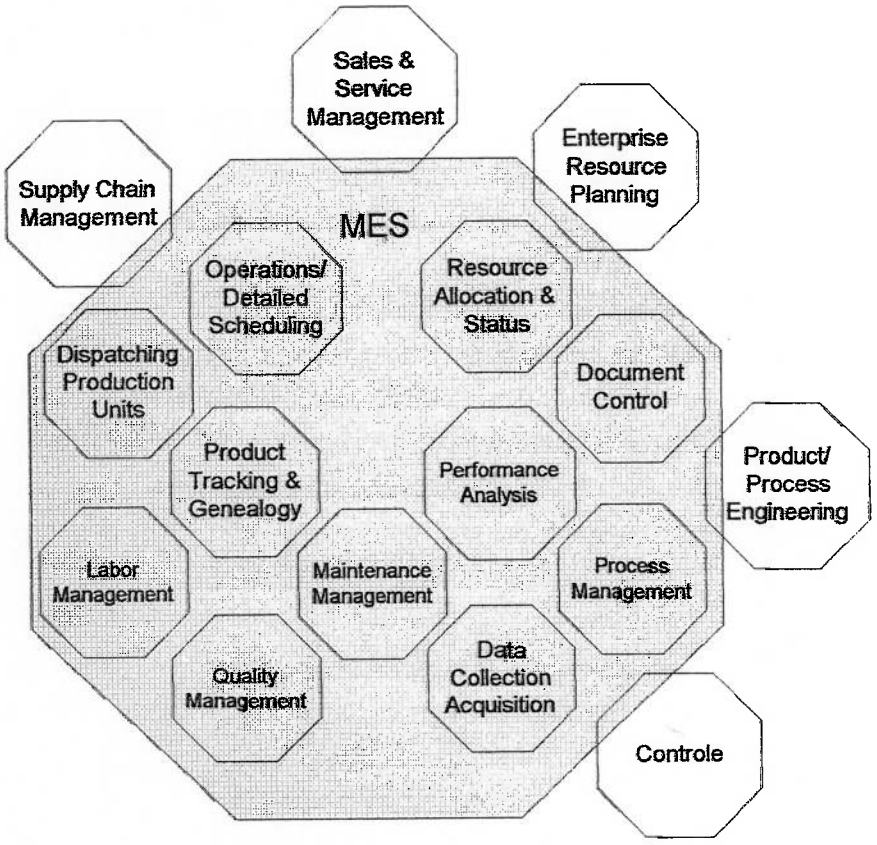

Figura 3 – Detalhamento do contexto dos MES nas corporações[2]

[1] Adaptado de [MESA, 1997f]

2.2.2 As funções do MES

As implementações de MES resultam na automação do fluxo de informação da empresa desde o cadastro do pedido até a remessa do produto, passando por todos os passos das áreas técnicas e administrativas. Na verdade, segundo [KAMAL, 1998], "disponibilizar informações no lugar certo, na hora certa, é a essência da manufatura integrada". Um sistema MES propriamente implementado deve proporcionar assistência computacional a diversas funções envolvidas na fabricação de produtos, dos chamados "sistemas gerenciais" aos "sistemas de produção" [MESA, 1997c], incluindo:

2.2.2.1 Alocação de Recursos e Status

O MES deve ser capaz de gerenciar os recursos de produção disponíveis na planta, como máquinas, ferramentas, mão-de-obra, habilidades específicas, materiais e equipamentos de apoio, além de outras entidades como documentos que precisem estar disponíveis para o trabalho das equipes de produção.

Além disso, o MES deve ser capaz de proporcionar informações sobre o histórico de cada um dos elementos supracitados, garantir que esses recursos estejam devidamente preparados para a atividade de produção e acompanhar seu status em tempo real, de modo que os objetivos de produção sejam atingidos.

2.2.2.2 Agendamento de operações detalhado

O sistema MES deve desenvolver o seqüenciamento de produção baseado nas prioridades, atributos, características e receitas associadas a cada produto, em cada pedido. É parte dos objetivos do MES a redução do tempo de preparação das máquinas (*setup*), ao mínimo indispensável e para tanto o sistema deve ser capaz de considerar, no agendamento de atividades, aspectos como cores, formatos, formulação, processos análogos e componentes equivalentes, além de ter condições de avaliar situações de processamento de materiais em paralelo e gerar *mixes* de produção o mais eficientes possível.

---

[2] Adaptado de [MESA, 1997f]

2.2.2.3 Alocação de unidades de produção

Todo o fluxo de produtos na empresa deve ser gerenciado pelo MES, seja sob a forma de pedidos, lotes, trabalhos ou outro método desejado. A seqüência em que esses pedidos são processados deve ser controlada pelo MES, levando-se em consideração a cadeia de prioridades e todas as eventuais alterações no plano, que podem ocorrer mesmo após o início do processo produtivo.

O MES deve dispor da capacidade de alterar o plano de trabalho no chão-de-fábrica e de considerar também o uso de atividades de retrabalho e sucateamento de itens durante o processo produtivo, além de balancear adequadamente as atividades em curso a cada instante.

Ou seja, além de realizar o planejamento adequado das seqüências de atividades em cada pedido (como apresentado no item 2.2.2.2), o MES deve ser capaz de considerar o universo de todos os pedidos em carteira, de modo a tornar o atendimento dos prazos de produção o mais eficiente possível, dentro das limitações dos recursos disponíveis na planta.

2.2.2.4 Controle de documentação

O processo produtivo normalmente dispõe de um conjunto de documentação, necessário à sua realização adequada. O sistema MES deve ser capaz de disponibilizar aos operadores informações como desenhos de fabricação e montagem, receitas, diagramas e esquemas elétricos, instruções operacionais, instruções de segurança, listas de peças, ordens de produção e outros documentos que possam ser necessários ao trabalho no chão-de-fábrica.

Além disso, o MES deve viabilizar o registro eficiente de histórico da produção e a comunicação entre turnos, de modo a minimizar os erros durante o processo. Deve ainda incluir informações sobre segurança e uso adequado dos equipamentos, documentações ISO e gerenciamento de Ações Corretivas.

2.2.2.5 Coleta de dados

O MES deve gerenciar as diversas informações de produção associadas a cada pedido e a cada produto, tais como parâmetros de produção, índices de produtividade,

relatórios de problemas e alarmes gerados ao longo do processo produtivo. É desejável que o MES seja integrado aos equipamentos de produção, de modo que esses dados estejam disponíveis para consulta em tempo real, podendo ser utilizados para a supervisão eficiente das atividades de chão-de-fábrica.

2.2.2.6 Gestão de mão-de-obra

As atividades de gestão de mão-de-obra de responsabilidade do MES referem-se ao controle da alocação dos recursos, manutenção de registros atualizados das capacidades específicas dos colaboradores envolvidos no processo produtivo, controle de suas certificações e análise de sua produtividade.

O MES deve dispor de interface com sistemas de controle de freqüência dos colaboradores, de modo a poder realizar a realocação da mão-de-obra em casos de ausência de elementos importantes ao processo, minimizando as perdas de produtividade.

2.2.2.7 Gestão da qualidade

As funções do MES relacionadas com a gestão da qualidade no processo produtivo referem-se à análise em tempo real das características significativas dos itens produtivos, utilizando ferramentas como o Controle Estatístico de Processos (CEP), para detectar tendências anormais nas atividades de fabricação e tomar ações imediatas para corrigi-las, como apresentado em [KUO, 2000].

A coleta de dados para essas tomadas de decisão pode ser feita tanto através do interfaceamento de módulos específicos diretamente a dispositivos e instrumentos de controle e medição, como também através de terminais de chão-de-fábrica em que os operadores lançam periodicamente os dados coletados manualmente. Já as ações corretivas podem resumir-se a informar os operadores de anormalidades ou até mesmo sugerir alternativas de solução e atuar diretamente no processo produtivo.

As ferramentas disponíveis ao MES para a gestão da qualidade podem ainda incluir o gerenciamento dos instrumentos de medição e calibração e interfaces com os sistemas de gerenciamento de laboratórios (LIMS – *Laboratory Information Management Systems*).

2.2.2.8 Gerenciamento de processos

O MES deve monitorar as atividades produtivas constantemente e proporcionar aos operadores informações suficientes para a atuação corretiva nos processos em caso de necessidade, podendo inclusive atuar diretamente quando tecnicamente viável.

É aconselhável que o MES seja capaz de identificar e processar os alarmes provenientes do chão-de-fábrica (recebidos através de sua interface de coleta de dados apresentada em 2.2.2.5), de modo a garantir que os responsáveis pela produção estejam cientes das ocorrências e tomem as medidas necessárias para garantir o prosseguimento das atividades com perdas mínimas.

O MES deve ainda acompanhar os pedidos e os itens produzidos mesmo entre os diversos estágios dos processos produtivos, ou seja, em etapas de estoque intermediário ou transporte.

2.2.2.9 Gestão de atividades de manutenção

O sistema deve realizar as atividades referentes ao planejamento e agendamento de manutenções preventivas, bem como acionar trabalhos de manutenção corretiva em situações de alarme. Deve, ainda, ser mantido um histórico das atividades de manutenção por equipamento e o status corrente de cada equipe de manutenção.

2.2.2.10 Rastreabilidade e genealogia de produtos

Durante a fabricação dos itens componentes de cada pedido, devem ser mantidas atualizadas informações sobre os equipamentos que estão processando cada item, os operadores e supervisores responsáveis por cada etapa, eventuais problemas e alarmes, as matérias-primas utilizadas e as ferramentas usadas.

Após a conclusão das etapas de produção, esse histórico deve ser utilizado para viabilizar a rastreabilidade dos itens vendidos, usando como base dados como o número de série dos produtos finais e seus componentes ou números de ordens de produção.

2.2.2.11 Análise de desempenho

O sistema deve oferecer relatórios de análise em tempo real dos diversos índices de avaliação do desempenho do processo produtivo, como produtividade por máquina e operador, informações sobre parâmetros de qualidade, conformidade da produção com o cronograma original, utilização de recursos e tempo do ciclo de produção de cada item.

Esses dados devem ser apresentados tanto na forma de relatórios como de forma *on-line*, a fim de possibilitar a avaliação contínua do desempenho e a atuação imediata no processo pelos responsáveis.

2.2.3 Os benefícios do MES

Uma das particularidades dos MES reside no fato de ser uma tecnologia já utilizada em uma grande variedade de tipos de empresas, com resultados comprovadamente positivos. Obviamente, o grau de satisfação dos usuários dos MES assim como os benefícios advindos de sua implantação dependem diretamente da qualidade do sistema utilizado, mas uma pesquisa realizada pela MESA forneceu os seguintes resultados, que servem como referência para um eventual estudo de viabilidade ou processo de decisão [MESA, 1997a]:

- Redução no tempo de ciclo de fabricação: 60% dos fabricantes afirmaram que obtiveram uma redução no tempo de ciclo de fabricação da ordem de 40% ou superior;
- Redução no tempo de entrada de dados: 60% dos fabricantes entrevistados responderam ter obtido uma redução no tempo de entrada de dados da ordem de 75% ou superior;
- Redução na burocracia entre turnos: 63% das empresas afirmaram ter obtido uma redução no volume de documentação trocada entre os turnos da ordem de 50% ou superior;
- Redução dos prazos de entrega: 50% afirmaram ter obtido uma redução nos prazos de fabricação de 30% ou melhor.

Como já dito anteriormente, o papel principal dos MES é o de ser a ponte entre os sistemas de planejamento e o chão-de-fábrica, atuando diretamente na redução dos custos e tempo de produção, racionalizando ao máximo a operação da planta. Quando um sistema MES é utilizado de modo a atuar sobre os processos industriais em tempo real, a interface de dados passa a transferir informações sempre atualizadas para os sistemas de planejamento e gestão, auxiliando na elaboração de custos, folhas de pagamento, controle de pedidos e lotes, inventário e diversos outros fatores importantes para o sucesso da empresa [MESA, 1997a].

Os benefícios dos MES vêm sendo progressivamente percebidos pela indústria de manufatura, sendo que algumas grandes corporações como a General Motors têm inclusive patrocinado pesquisas na área para adequar a tecnologia MES à sua realidade e conseguir, dessa forma, extrair o máximo de resultados com sua implantação [LONG, 1999].

2.2.4 Relação MES x CORBA

A definição da MESA para os Sistemas de Controle Supervisório e de Aquisição de Dados, ou SCADA, é a de que são os "responsáveis pela medição, monitoramento e manipulação de produção, pessoas, produtos e processos" dentro do ambiente de produção, ou chão-de-fábrica [MESA, 2000a]. Nessa definição, os Controles incluem todos os sistemas automatizados de controle de processos, como os PLC's (*Programmable Logic Controllers*) e os DCS (*Distributed Control Systems*); equipamentos de automação específicos, como leitores de códigos de barras e equipamentos RFID (*Radio Frequency Identification*); e controladores por software como os comandos NC (*Numeric Control*) e os soft PLC's. Os controles também incluem produtos discretos como sensores, *drives*, motores, inversores de freqüência e transdutores que podem fornecer informação em tempo real para os PLC's e DCS.

Desse modo, ao observar-se a figura 2 e analisarem-se a definição dos MES e as funcionalidades propostas para esta categoria de sistemas, percebe-se a necessidade de uma integração consistente com os Sistemas de Controle, a fim de que várias das 11 funções básicas dos MES como a coleta de dados, alocação de recursos, gerenciamento de processos e gerenciamento da qualidade sejam executadas.

Os Sistemas de Controle têm impacto direto sobre a criação dos produtos, da liberação de matéria-prima até as inspeções finais e, uma vez integrados aos MES, proporcionam ao usuário do sistema total controle e *feedback* das atividades de chão-de-fábrica.

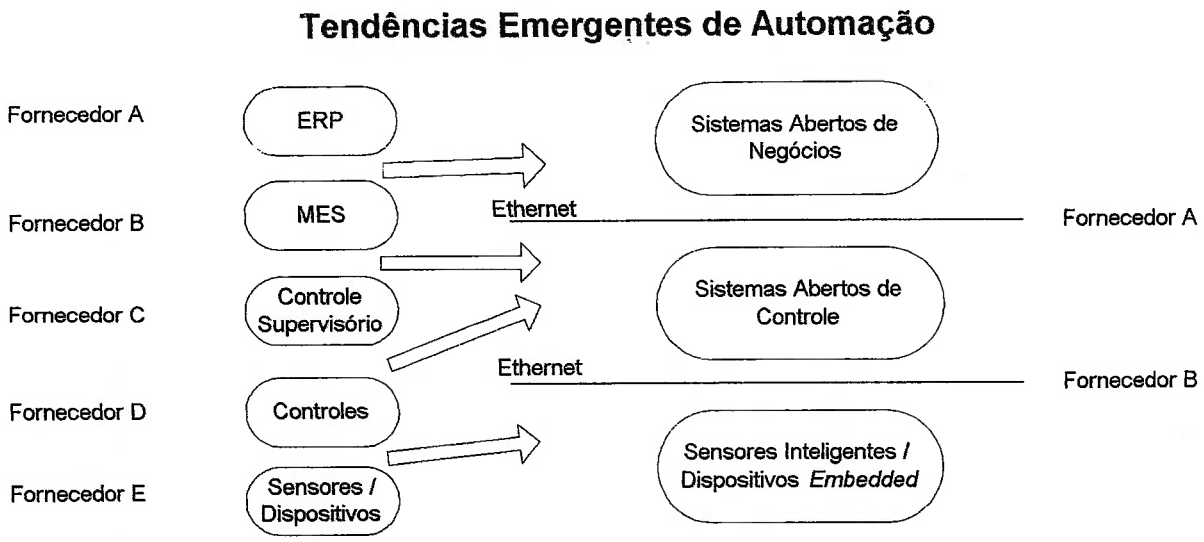

Figura 4 – Tendências de automação[3]

A Figura 4 apresenta um modelo de acesso aos dados de um Sistema de Controle, na visão da MESA. Nessa visão, cinco áreas funcionais em um sistema de automação (ERP, MES, Controle Supervisório, Controles/Sistemas de Automação e Sensores/Dispositivos) convergem para um modelo menos estruturado, baseado em sistemas abertos e padrões de software, como XML, COM/DCOM, Java e CORBA. Esses padrões viabilizam e organizam a aquisição e análise de dados, eliminando muitas das fronteiras existentes entre os MES, os sistemas de gestão e os sistemas de chão-de-fábrica, possibilitando o uso de um modelo de dados único para toda a empresa [MESA, 2000a].

A interação entre os MES e a camada de Controle é dinâmica, sendo que as informações de controle e status podem ser acessadas assim que forem criadas, de modo a satisfazer a requisitos de atividades como o CEP (Controle Estatístico de

[3] Adaptado de [MESA, 2000a]

Processos), por exemplo. Esses dados também podem ser obtidos em lote (*batch*) ou armazenados em um banco de dados para análises posteriores e atividades de manutenção e rastreamento de processos. Os sistemas de controle trabalham em tempo real, com um controle de tempo da ordem de menos de 1 segundo, a que será estabelecido como fator 1 X para efeito de entendimento do diagrama apresentado na figura 5[4].

Já a interface entre os MES e os sistemas de gestão do negócio (ERP's) dá-se com uma necessidade de dinamismo mais reduzida, uma vez que os ERP's trabalham com um horizonte de tempo de dias, semanas ou meses, em um fator de tempo 100 X. Os ERP's controlam a utilização dos produtos, os pedidos dos clientes e os requisitos de matéria-prima, enviando solicitações ao sistema de Execução (MES) para que sejam fabricados itens em quantidades adequadas à demanda.

Os MES, como responsáveis pelas atividades de manufatura e pelas operações relacionadas com a criação dos produtos propriamente dita, controlam receitas, informações e planos de produção e outras informações necessárias à camada de Controle. Como os MES trabalham com uma janela de tempo mais reduzida, de um dia, um turno, uma hora, um minuto ou um segundo, mas não chegam ao tipo de requisitos de tempo de resposta dos sistemas de Controle, seu fator de tempo pode ser visto como 10 X.

A Figura 5 apresenta a visão da MESA para a integração de informações entre as camadas de Controle, Execução e Gestão da empresa.

Neste contexto, os MES atuam como uma interface bidirecional no processo de manufatura, integrando e facilitando os fluxos de informação e comandos entre os sistemas de Controle e os sistemas de gestão do negócio.

---

[4] Esse tipo de requisito de desempenho dos sistemas da camada de Controle motivaram o desenvolvimento de extensões proprietárias de tempo real e tolerância a falhas para a arquitetura CORBA, como o produto HardPACK descrito em [LOCKE, 1999] e o projeto ARTDOM descrito em [WOHLEVER, 1999]. Além disso, o posterior desenvolvimento das especificações CORBA para tempo-real também foram em grande parte motivadas por esse requisito dos sistemas de controle de manufatura [SCHMIDT, 2000].

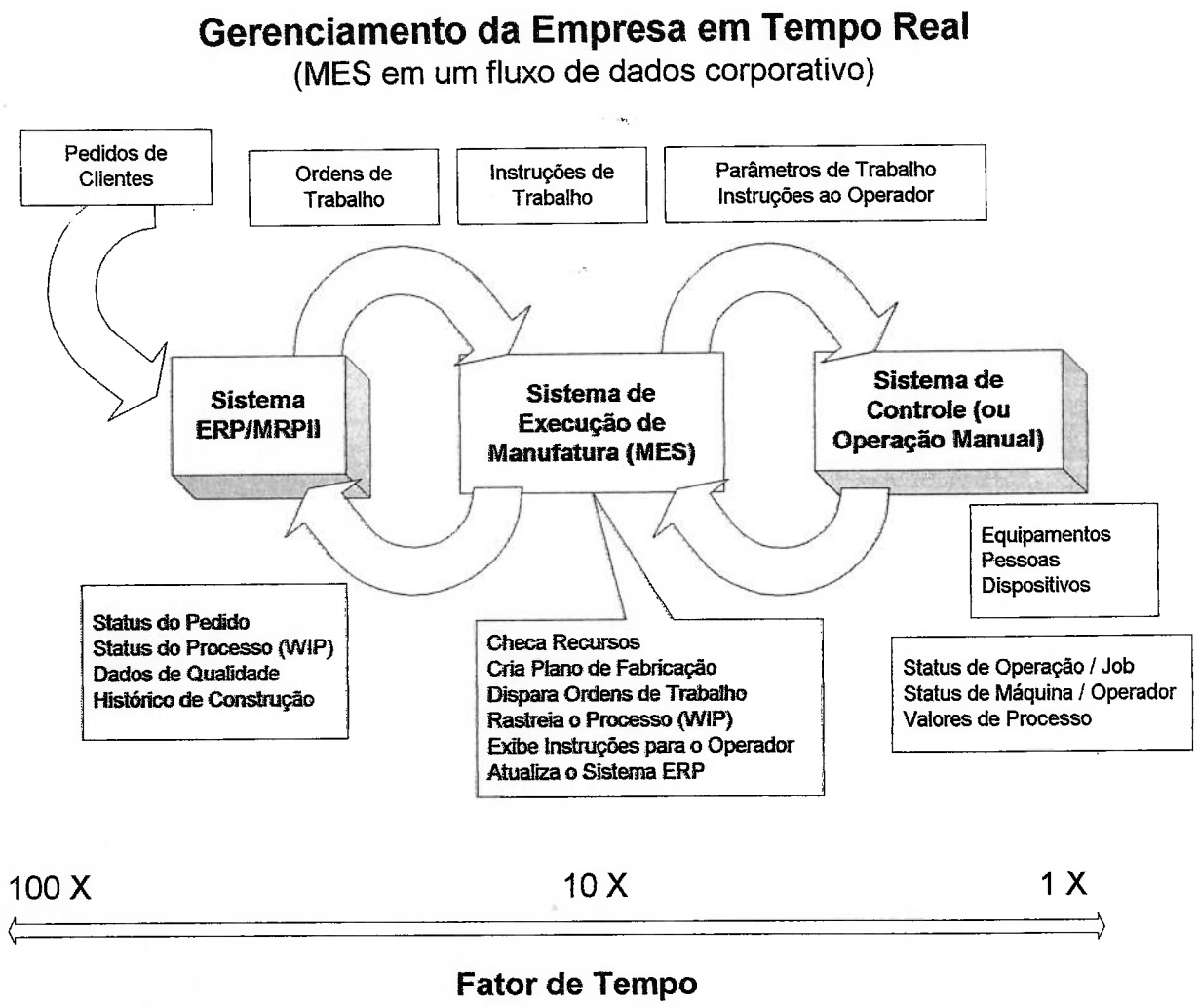

Figura 5 – Gerenciamento em tempo real das atividades de produção

Existe em andamento um consórcio (NIIP-SMART) que objetiva a criação de um modelo para a utilização das tecnologias COM/DCOM e CORBA na geração de objetos MES, da qual a MESA faz parte. Como poderá ser visto no item 2.3, que trata da análise de alguns dos produtos SCADA e MES disponíveis hoje no mercado, a maioria dos produtos de Controle são projetados para possibilitar comunicação com outras aplicações, possuindo interfaces de software que fazem uso de COM/DCOM ou CORBA.

Uma visão simplificada deste modelo pode ser vista na figura 6.

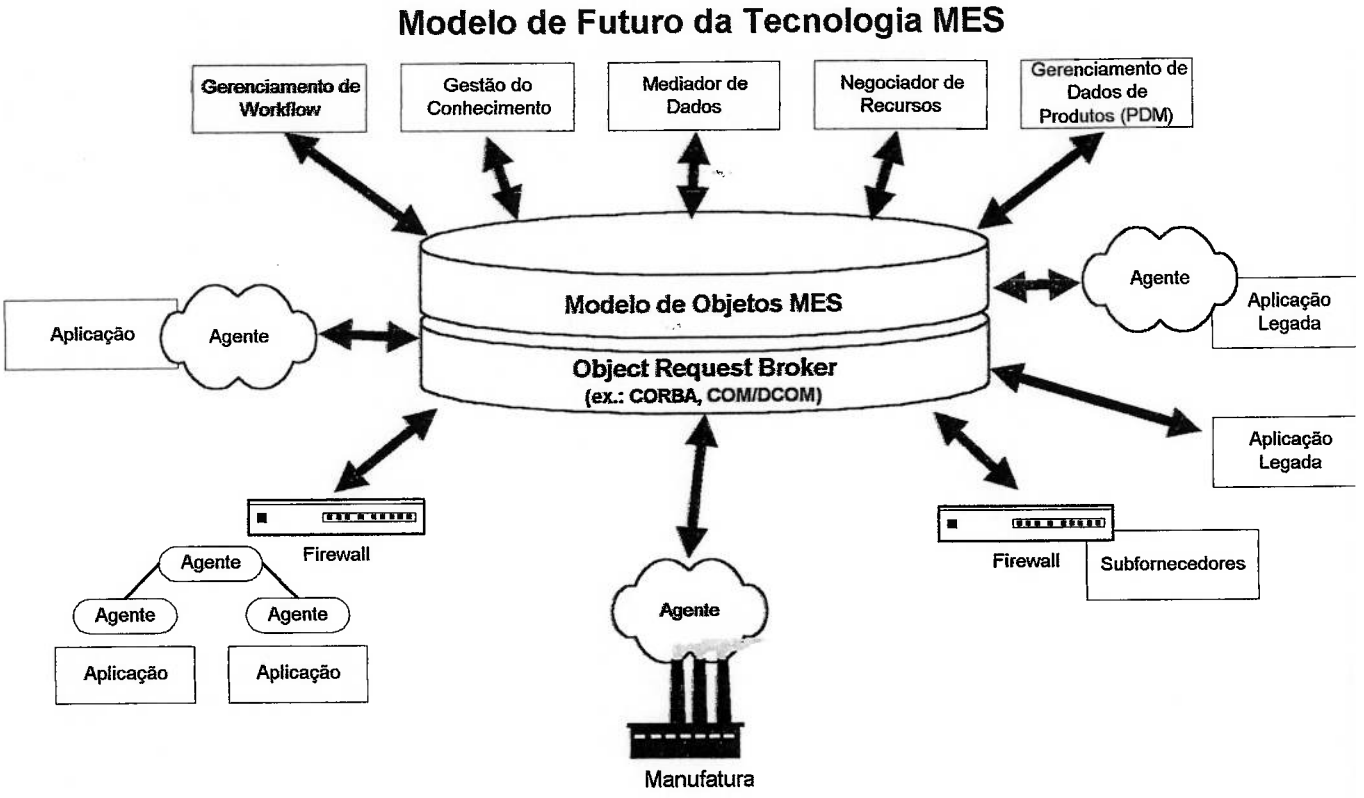

Figura 6 – Modelo NIIP-SMART[5]

Com a crescente inclusão das tecnologias COM/DCOM e CORBA nos sistemas de Controle e com os novos produtos baseados em software (*Soft Controllers*), o modelo de objetos criado pelo consórcio passa a poder ser estendido até a camada tradicionalmente de hardware dos ambientes de produção.

Os pacotes *Soft Controllers* possuem implementações para a emulação de PLC's, dispositivos de E/S e IHM's. Normalmente, esse tipo de software opera sobre plataformas Windows 2000 ou NT como um serviço e uma das principais vantagens que pode propiciar é a capacidade de integrar os dados do processo diretamente nas aplicações MES através de componentes CORBA ou COM/DCOM. Entretanto, a incorporação cada vez maior de inteligência a esses pacotes através, por exemplo, da tecnologia de Agentes, pode aumentar a gama e a flexibilidade das funções das aplicações de controle, além de melhorar seu desempenho, como exposto em [KOSAKAYA, 2002].

2.3 A Proposta CIM SEMATECH

2.3.1 *Frameworks*[6] para Sistemas Industriais Distribuídos

Um *framework*, no que se refere ao âmbito dos sistemas computacionais, consiste em uma infra-estrutura de software que possui duas características principais [WASKIEWICZ, 1997]:

- Proporciona um ambiente comum para a integração de aplicações e o compartilhamento de informações em um dado domínio de problema;
- Define as "regras de jogo" (protocolo) que permitem a objetos distribuídos comunicarem-se e colaborarem para realizar uma série de tarefas;

No caso de um *framework* orientado a objetos, o conceito refere-se a uma coletânea de abstrações reutilizáveis de objetos, serviços e protocolos, ou seja, o *framework* é muito mais do que um simples repositório ou biblioteca de objetos.

Cada domínio de aplicação deve desenvolver seu próprio *framework*, sendo que os mesmos são específicos para cada um deles, embora apresentem características de possibilidade de reutilização dentro de um dado domínio.

No presente documento, estando-se a tratar de sistemas integrados de manufatura, considera-se um *framework* como sendo um conjunto de abstrações de objetos relacionados ao ambiente de manufatura e os serviços mínimos necessários para se construírem aplicações com características de interoperabilidade e capacidade de substituição (*replaceable applications*) no domínio do MES.

2.3.2 Conceitos e princípios do framework CIM da SEMATECH

A organização SEMATECH (SEmiconductor MAnufacturing TECHnology) consiste em um consórcio sem fins lucrativos para o desenvolvimento de tecnologia formado em 1987 e baseado em Austin no estado americano do Texas [WASKIEWICZ,

---

[5] Adaptado de NIIP-SMART Reference Architecture

[6] Apesar de existir um termo equivalente a *framework* em português (arcabouço), optou-se por fazer uso do termo em inglês devido ao fato desse ser mais usualmente utilizado no mercado nacional de tecnologia da informação.

1997]. Entre os trabalhos desenvolvidos pela SEMATECH, existe o desenvolvimento de um framework para aplicações de CIM (*Computer Integrated Manufacturing*).

Por CIM entenda-se a integração total da empresa do segmento de manufatura, através do uso de sistemas integrados e da comunicação de dados, aliados a novas filosofias gerenciais que melhoram a eficiência da organização e dos colaboradores [WARREN, 1990]. O CIM compreende a integração total do ambiente de manufatura, englobando inclusive os sistemas MES.

O propósito do framework desenvolvido pela SEMATECH consiste em estabelecer uma arquitetura padrão para a aplicação dos conceitos de CIM no ambiente de manufatura de semicondutores, levando a um ambiente CIM aberto e com possibilidade de integração de equipamentos de diversos diferentes fornecedores [SEMATECH, 1998].

Devido ao aumento constante da complexidade dos métodos de fabricação e dos equipamentos envolvidos na produção de semicondutores, as capacidades de controle de produção e processos proporcionadas pelos sistemas CIM são essenciais para o gerenciamento eficiente das fábricas de semicondutores. Entretanto, grande parte dos sistemas CIM atuais, monolíticos, baseados em ilhas isoladas de automação e de difícil modificação e manutenção, acabam agregando um alto custo para a indústria que se propõe a usá-los e acabam levando a um baixo grau de satisfação do cliente.

Neste contexto, o framework CIM da SEMATECH propõe-se a resolver muitos desses problemas e estabelecer o caminho para a próxima geração de sistemas de manufatura de semicondutores através dos seguintes passos [SEMATECH, 1998]:

- Proporcionar um sistema de projeto integrado e reutilizável que especifique claramente o escopo funcional, as fronteiras e o modelo padrão dos componentes necessários a um sistema de manufatura de semicondutores;
- Identificar as funcionalidades CIM comuns a todos os fabricantes de semicondutores;
- Basear a implementação em tecnologia aberta, avançada e validada de software e em padrões reconhecidos.

O framework CIM SEMATECH define um padrão de componentes que proporcionam funcionalidade comum através das aplicações CIM e possibilita a integração dessas aplicações. Os componentes do framework, todavia, não proporcionam sozinhos toda a funcionalidade de um sistema CIM integrado, sendo projetados para ser especializados e estendidos com funções e comportamentos adicionais a fim de possibilitar a correção dessa deficiência e a capacidade de adaptação necessária para uma solução CIM específica. O framework especifica apenas as porções do sistema CIM necessárias à integração padrão de aplicações, deixando o resto do sistema em aberto a fim de que fornecedores e usuários possam ter a flexibilidade de atender a necessidades específicas de cada caso. Com relação às sete camadas OSI, o framework da SEMATECH foca na camada da aplicação [WASKIEWICZ, 1997].

O coração do framework consiste em um conjunto de abstrações do processo de manufatura de semicondutores (processo de especificação de *waffers*, usinagem e assim por diante) e de serviços (como localização do *waffer* ou ajuste de parâmetros), que são tipicamente incorporados nas aplicações (por exemplo, de gerenciamento de materiais e controle de máquinas), e fornecidos em plataformas computacionais distribuídas formadas que usam tecnologia de software padrão para implementar comunicações, interfaces homem-máquina ou bancos de dados,.

A versão 2.0 do framework SEMATECH é focada especificamente no gerenciamento de informações de manufatura e no controle de fases operacionais e de planejamento da fabricação de waffers semicondutores. Entretanto, grande parte desta especificação pode ser aplicada em outros ambientes de manufatura, como o encapsulamento de componentes e a montagem de circuitos impressos [SEMATECH, 1998].

2.3.3 O escopo funcional

A especificação do framework CIM SEMATECH define um conjunto de componentes funcionais projetados para trabalhar juntos a fim de formar um sistema integrado de manufatura, sendo agrupados por área de aplicação conforme segue:

Tabela 1 - Agrupamento dos componentes funcionais CIM SEMATECH[7]

Serviços de Fábrica • Gerenciamento de documentos • Gerenciamento de versões • Broker de Eventos • Gerenciamento de histórico	**Controle de Máquinas** • Gerenciamento de máquinas • Gerenciamento de receitas • Rastreamento de recursos	**Controle avançado de processos** • Gerenciamento de Plug-In's • Execução de Plug-In's • Gerenciamento de controle • Execução de controle • Banco de dados de controle • Plano de coleta de dados
Gerenciamento de Fábrica • Fábrica • Lançamento de produtos • Operações de Fábrica • Requisição de produtos	**Gerenciamento de Materiais** • Gerenciamento de produtos • Gerenciamento de itens duráveis • Gerenciamento de itens de consumo • Inventário • Especificação de produto • Relação de materiais	**Gerenciamento de especificação de processos** • Especificação de processos • Capabilidade de processos
Mão-de-obra de Fábrica • Gerenciamento de pessoal • Gerenciamento de habilidades	**Movimentação de material** • Movimentação de material	**Gerenciamento de Cronograma** • Despacho de mercadorias

A especificação suporta múltiplos níveis de empacotamento das funcionalidades, de forma que uma grande variedade de aplicações de diferentes fornecedores com capacidades similares podem ser acomodadas desses grupos. Isso significa que dada uma ligação específica de um framework a um conjunto de tecnologias de computadores e sistemas de software, uma aplicação pode ser instanciada e executada nesse ambiente e irá registrar-se como um de uma série de objetos bem conhecidos para proporcionar um conjunto de serviços predefinidos no framework.

[7] Adaptado de [SEMATECH, 1998]

2.3.4 Os benefícios do CIM SEMATECH

Dentro do domínio específico dos sistemas de manufatura de semicondutores, a proposta CIM da SEMATECH apresenta os seguintes benefícios às empresas as quais se propõem a adotá-la [HAWKER, 1997]:

- Maior velocidade e menor custo no desenvolvimento ou seleção de componentes específicos para atender a necessidades localizadas de solução, através da integração com outros componentes ou através da extensão de componentes preexistentes para o atendimento de novas necessidades;
- Orientação a propriedades fundamentais das novas tecnologias de sistemas de informação, como interoperabilidade, capacidade de substituição, extensibilidade e reutilização;
- Infra-estrutura voltada para desenvolvimento da independência de fornecedores ou sistemas proprietários e monolíticos.

2.3.5 Detalhamento do framework CIM SEMATECH

O framework CIM da SEMATECH foca no nível dos sistemas de execução de manufatura (MES), entre o controle direto de equipamentos e o planejamento corporativo, conforme mostrado na figura 7.

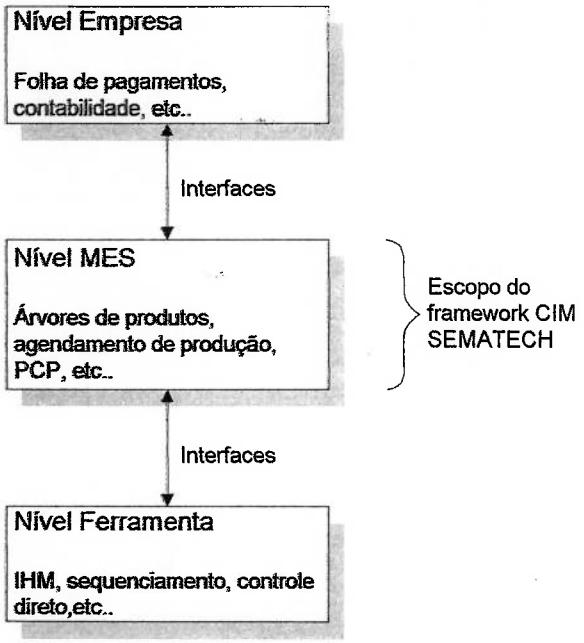

Figura 7 - Contexto no sistema corporativo[8]

Sendo assim, o framework CIM define as interfaces entre as camadas funcionais imediatamente acima e abaixo dele. Também aprofunda-se mais do que os sistemas tradicionais MES e acaba sobrepondo-se em parte ao nível identificado por "ferramenta" na figura 7, incorporando funções como o rastreamento dos waffer, modelagem do manuseio de materiais e modelagem da disponibilidade de máquinas.

As unidades construtivas do framework da SEMATECH são os chamados "componentes", unidades indivisíveis de funcionalidade do sistema e que possuem fronteiras e interfaces padronizadas. Esses componentes são então agrupados nos chamados "grupos funcionais", que acabam por compor a fábrica. De uma maneira geral, os componentes e os grupos são aqueles definidos anteriormente na tabela 1.

Como é de se esperar, os componentes são formados por classes, que definem, por sua vez, os métodos e as variáveis presentes em cada objeto do framework. A granularidade do framework CIM SEMATECH, portanto, pode ser resumida conforme a figura 8 a seguir:

---

[8] Adaptado de [SEMATECH, 1995]

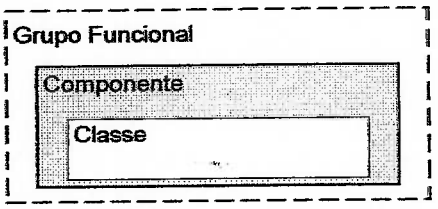

Figura 8 - Granularidade do CIM SEMATECH

As classes, por sua vez, podem ser de dois tipos: públicas (que podem ser vistas de fora do componente a que pertence) ou privativas (que não podem sê-lo). Através dos métodos disponibilizados pelas classes, cada grupo funcional pode fornecer ou solicitar serviços dos componentes dos outros grupos funcionais, da mesma forma que os componentes de um mesmo grupo também podem compartilhar seus serviços.

Para maior clareza, o relacionamento entre os elementos do framework CIM SEMATECH e entre seus grupos funcionais pode ser visto nas figuras 9 e 10.

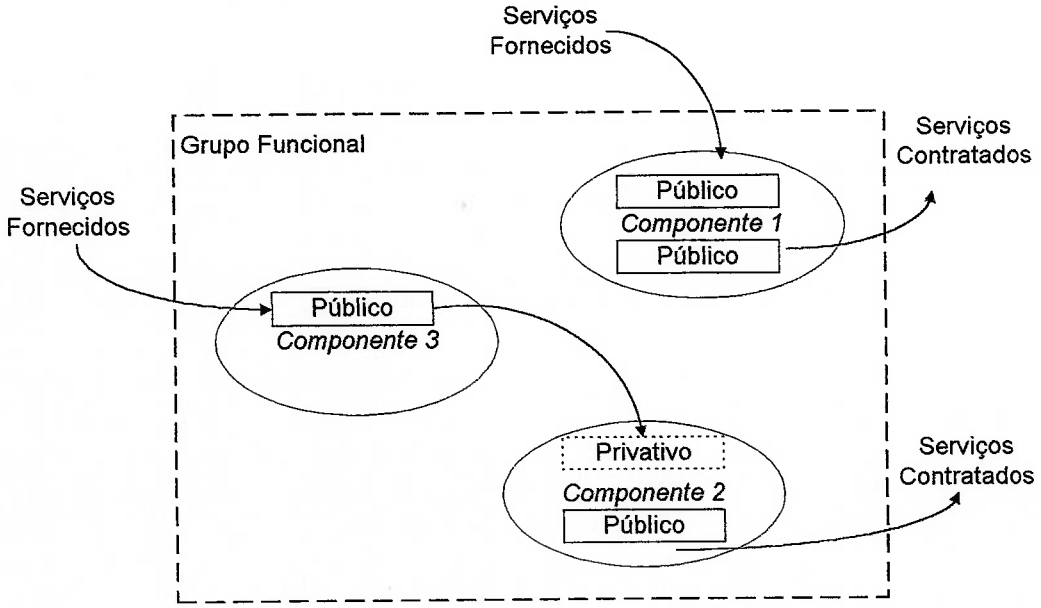

Figura 9 - Relacionamento entre os elementos

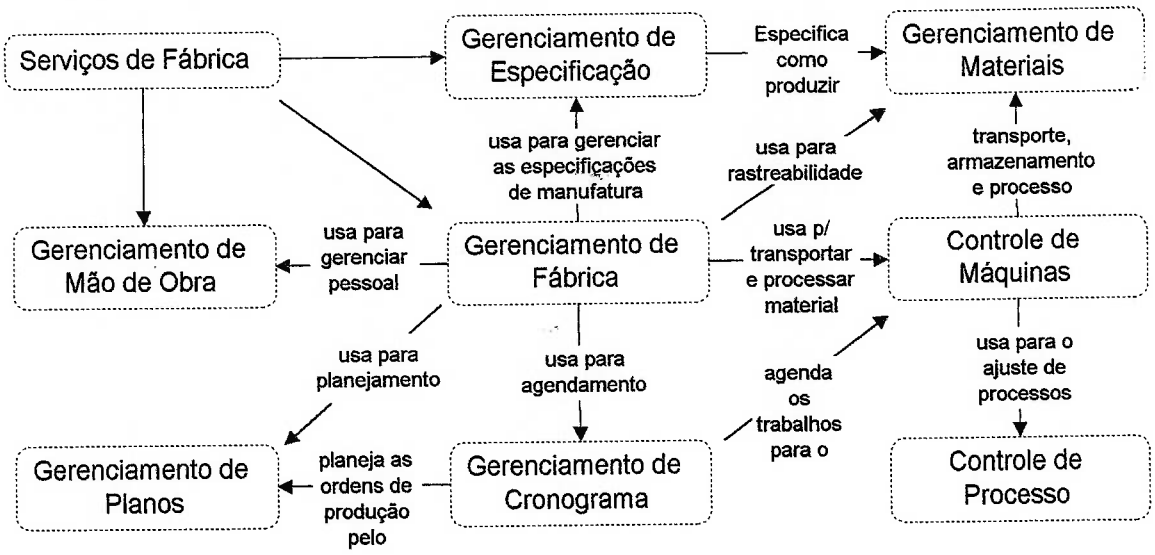

Figura 10 - Relacionamento entre os grupos funcionais

Como exemplo ilustrativo, tomemos o grupo funcional de Gerenciamento de Especificação. A função deste grupo é fornecer uma arquitetura para a geração e gerenciamento das diversas especificações dos produtos fabricados na empresa. Os componentes que estão incorporados neste grupo são os seguintes:

- Gerenciamento de Documentos - gerencia documentos e revisões
  - A classe *DocumentSpecification* é uma especificação on-line para os produtos e processos de manufatura
- Gerenciamento de Versão - suporta a troca nas especificações
- Gerenciamento de Definições de Processo - define os processos e as seqüências de passos de fabricação no nível do processo
- Gerenciamento de Especificações de Produtos - define as especificações e a maneira de produzir no nível do produto

Uma das grandes propriedades do CIM SEMATECH consiste na facilidade que o framework oferece para o uso de aplicações "plugáveis" (*plug-in's*), conforme pode ser visto na figura 11, que representa, de uma maneira geral, as funções e a abrangência do framework. Todavia, esse tipo de interoperabilidade não pode ser obtido de maneira imediata, sendo que para duas ou mais aplicações serem "plugáveis" elas devem compartilhar o mesmo *binding*, ou conexão.

A figura 11 apresenta o framework na sua concepção original, com o *binding* incluindo o Sistema Operacional, Comunicações, Banco de dados, Interface do Usuário e Linguagem.

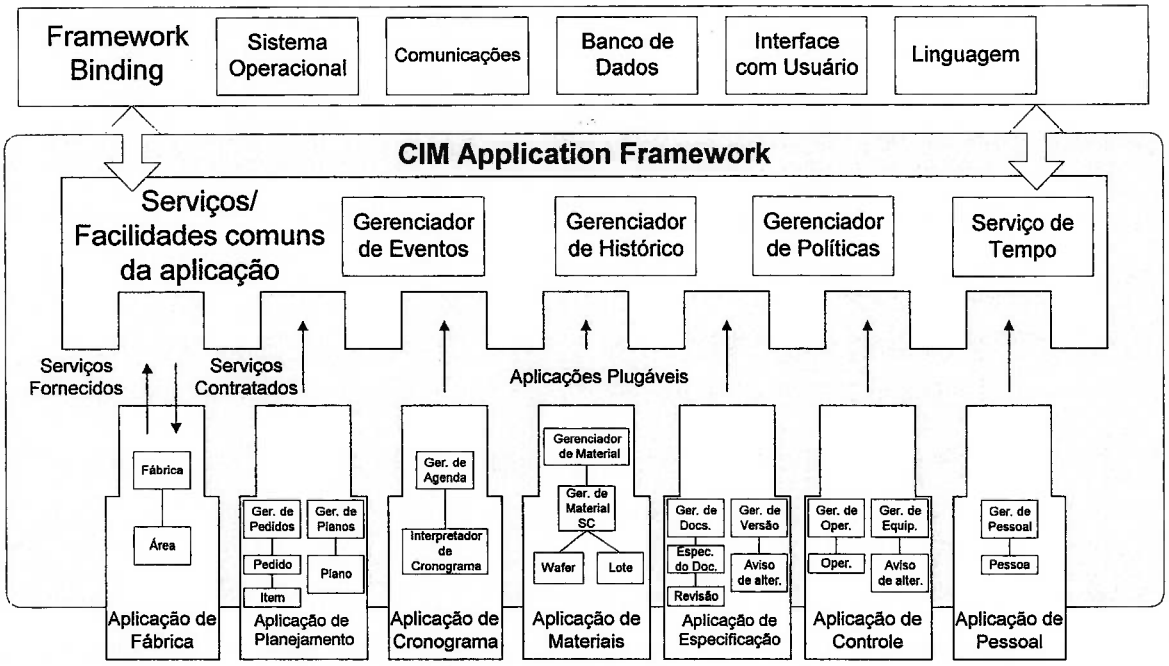

Figura 11 - O SEMATECH CIM framework[9]

Pode-se perceber, a partir da figura 11, uma possibilidade clara de uso da arquitetura CORBA para resolver-se alguns dos problemas envolvidos, principalmente o de implementação da camada *Framework Binding*, sendo que esse potencial de aplicação será explorado mais a frente, neste documento.

No caso das interfaces com o usuário, essa se torna um cliente tipo *plug-in* do framework, fazendo uso da implementação existente do framework para exibir dados e transferi-los para dentro do sistema. Apesar de aplicações contendo interfaces com usuários interfacearem com o framework CIM, a tecnologia IHM propriamente dita não faz parte do escopo de infra-estrutura do mesmo.

---

[9] Adaptado de [SEMATECH, 1995]

2.3.6 A relação da SEMATECH com a OMG

Já desde a versão 1.2 de seu framework CIM, a SEMATECH identificou na OMG uma possibilidade bastante concreta de parceria [SEMATECH, 1995]. Uma vez que a arquitetura CORBA define o ORB para possibilitar e regular a interoperabilidade entre objetos e aplicações através de plataformas de hardware, sistemas operacionais e linguagens heterogêneas, a SEMATECH decidiu orientar seu framework na direção da utilização da arquitetura padrão CORBA em vez de preocupar-se em desenvolver toda a parte de infra-estrutura de que precisaria para concluir seu trabalho.

Na versão 2.0, o relacionamento já é bastante explícito, sendo que as especificações do framework CIM passam a relacionar-se com as especificações adotadas pela OMA de diversas maneiras, fazendo com que o framework da SEMATECH precise ser visto no contexto desse grupo maior de especificações para que se possa ter uma compreensão correta das interfaces especificadas.

Sempre que a SEMATECH vê a possibilidade de utilizar especificações da OMA, isso é feito. Quando tal não é possível, devido a particularidades do domínio dos semicondutores, os incrementos à especificação da OMG são definidos e apresentados sendo que, assim que a OMG incorpora as novas melhorias sugeridas, a SEMATECH altera também suas especificações para referenciá-las.

2.4 A solução OPC

2.4.1 Arquitetura .NET e o modelo COM/DCOM

Da mesma maneira que a OMG trabalha para o desenvolvimento de uma arquitetura aberta e padronizada para o trabalho com objetos distribuídos, também a Microsoft investe recursos no desenvolvimento de seu próprio padrão, o modelo DCOM (*Distributed Component Object Model*).

A arquitetura DCOM foi concebida inicialmente como parte integrante da especificação Windows DNA (*Distributed Internet Applications*), que foi a tentativa original da Microsoft de criar uma especificação técnica para viabilizar a interoperabilidade em sistemas distribuídos [FLASH, 2002]. Posteriormente, com a evolução do Windows DNA para a atual arquitetura .NET, em que a Microsoft propõe um paradigma de computação distribuída baseado em serviços com baixo grau de acoplamento [MICROSOFT, 2002], o modelo DCOM foi mantido como parte integrante da especificação.

Como pode ser visto na figura 12, o padrão COM é uma infra-estrutura de integração, usada para implementar componentes que interagem dentro de um único espaço de endereços, ou em processos em um único *host*. COM é a base sobre a qual estão o OLE (*Object Linking and Embedding*), ActiveX, MTS (*Microsoft Transaction Services*) e um número crescente de serviços providos pelos sistemas operacionais Microsoft, como os serviços multimedia DirectX [TALLMAN, 1998].

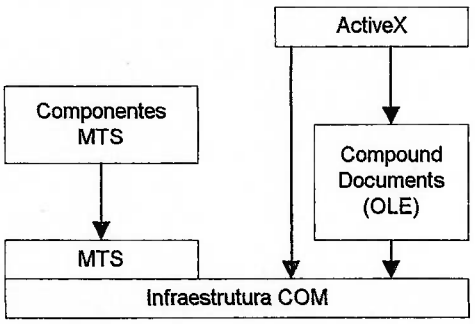

Figura 12 - Modelo COM[10]

[10] Fonte: [TALLMAN, 1998]

Já o padrão DCOM é uma extensão do modelo de componentes COM e constrói um RPC (*Remote Procedure Call*) para objetos (ORPC) sobre o RPC DCE (*Distributed Computing Enviroment*) já existente, a fim de suportar o uso de objetos remotos. Desse modo, DCOM possibilita a interação entre objetos sendo executados em *hosts* diferentes em uma rede. O padrão DCOM foi introduzido com o sistema operacional Windows NT 4.0 e como um adicional separado ao Windows 95 ao final de 1996.

Um objeto COM pode suportar múltiplas interfaces, cada qual representando uma visão diferente do comportamento do objeto e através dessas interfaces, os objetos COM cliente podem relacionar-se com os objetos COM servidores, de uma maneira análoga ao que ocorre com o CORBA. Ainda como no caso do CORBA, o padrão COM permite a integração de componentes escritos em diferentes linguagens de programação, como C++, Java e Visual Basic [CHUNG, 1998].

Da mesma maneira que o CORBA, o modelo DCOM também implementa comunicação usando um esquema cliente/servidor sendo que, para requisitar um serviço, um cliente invoca um método implementado por um objeto remoto, que passa então a agir como um servidor. De uma maneira diferente do modelo CORBA, o DCOM não suporta herança múltipla de interfaces, mas possibilita o uso de diversas interfaces por um mesmo objeto, obtendo o mesmo resultado.

Comparando-se as arquiteturas DCOM e CORBA, percebe-se que as duas são muito similares, sendo que ambas proporcionam a infra-estrutura de objetos distribuídos necessária para ativações transparentes e acesso a objetos remotos. Entretanto, algumas diferenças podem ser observadas, conforme segue:

- O modelo DCOM suporta objetos com múltiplas interfaces e fornece um método padrão QueryInterface() para a navegação entre elas, o que não existe no modelo CORBA;
- Toda interface CORBA herda o construtor que realiza tarefas implícitas como registro de objetos, geração de referências e instanciação de esqueletos. No DCOM, essas tarefas precisam ser realizadas explicitamente pelos servidores ou dinamicamente pelo run-time DCOM;

- O protocolo DCOM é amarrado fortemente ao RPC, ao contrário do que ocorre com o CORBA;
- A especificação DCOM inclui diversos itens que são considerados detalhes de implementação no CORBA, não estando explícitos na descrição da arquitetura, que se restringe ao nível de definição das interfaces em IDL;
- A especificação MIDL (*Microsoft's Interface Definition Language*) não apresenta um mecanismo para o tratamento de exceções, ao contrário da especificação CORBA IDL, conforme observado em [EXTON, 1997].

Ao final de 1998, a Microsoft lançou uma versão aprimorada do modelo COM, batizado de COM+. O padrão COM+ implementa uma série de propriedades adicionais, como o gerenciamento de objetos em tempo de execução, e novas ferramentas de desenvolvimento.

No ramo de automação industrial, diversas empresas já estão utilizando o modelo DCOM, principalmente em função do papel da Microsoft no contexto atual de sistemas de informação. Algumas das grandes e tradicionais empresas de automação industrial, como é o caso da Rockwell Automation e várias novas empresas de software para automação como a ObjectAutomation, já incorporam o modelo DCOM em seus produtos e estão lançando aplicações baseadas nele.

2.4.2 A especificação OPC

A partir de 1998, a OPC Foundation, uma organização sem fins lucrativos, estabeleceu um conjunto de protocolos padrões de interfaces OLE/COM destinados a facilitar uma maior interoperabilidade entre aplicações de controle/automação, sistemas e dispositivos de campo e aplicações de escritório na indústria de controle de processos. Esse conjunto de protocolos forma a especificação OPC (*OLE for Process Control*), baseada nos requisitos funcionais da tecnologia OLE/COM da Microsoft [OPC, 1998a].

A especificação OPC define objetos padrão, métodos e propriedades para servidores de informação em tempo-real, como PLC's, dispositivos de campo inteligentes e sensores, a fim de comunicar as informações desses servidores a dispositivos

compatíveis com a tecnologia OLE/COM, como softwares de supervisão ou geradores de relatórios (figura 13). Na visão da OPC Foundation, há a necessidade de um método comum para que os aplicativos possam acessar dados de qualquer fonte, seja ela um dispositivo ou um banco de dados [OPC, 1998b].

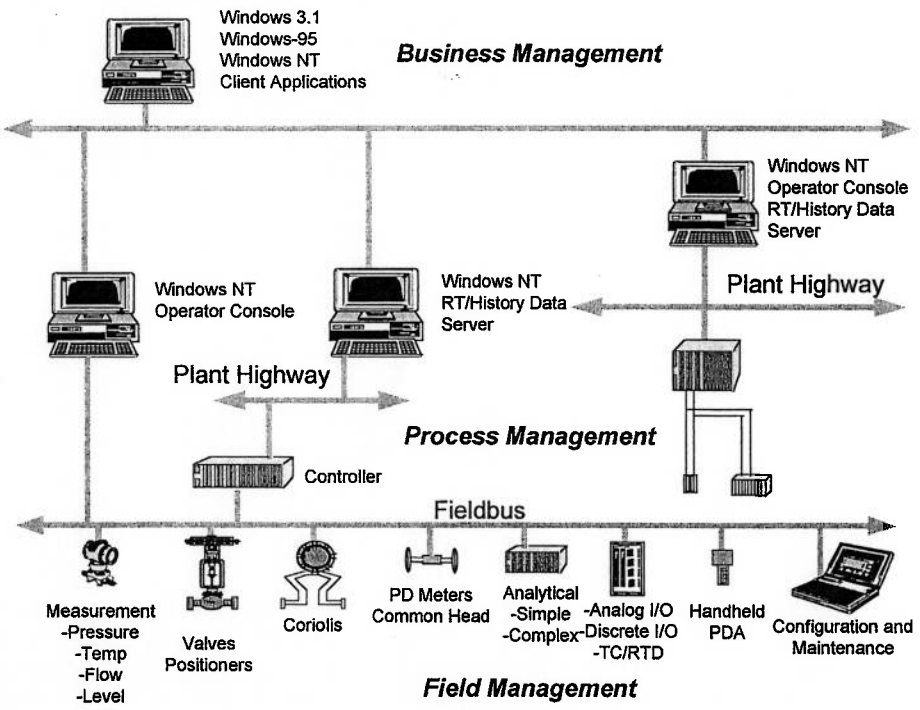

Figura 13 - Arquitetura do fluxo de informações de processo na visão da OPC Foundation[11]

Existem hoje desenvolvidas muitas aplicações cliente de supervisão e controle que precisam adquirir dados de fontes externas, como um sensor ou um PLC e que acabam obrigando o desenvolvimento de *drivers* específicos que realizem a interface entre elas [OPC, 1998b]. Esse desenvolvimento de *drivers* particulares acaba causando aos seguintes problemas:

- Esforços duplicados, uma vez que todos os fornecedores de software precisam escrever um *driver* para um determinado *hardware*;

- Inconsistências entre drivers de diferentes fornecedores, sendo que nem todos os *drivers* acabam utilizando as mesmas características de um determinado *hardware*;

[11] Fonte: [OPC, 1998b]

- Falta de suporte às mudanças no *hardware*, uma vez que qualquer alteração no *hardware* pode levar a problemas de compatibilidade com os *drivers*;

- Conflitos de acesso, já que dois pacotes diferentes não podem acessar o mesmo dispositivo simultaneamente, caso estejam utilizando dois *drivers* independentes.

Uma das alternativas de solução seria os fabricantes do hardware desenvolverem os drivers, mas esse trabalho é dificultado pelas diferenças entre os protocolos usados pelas aplicações cliente. Os fabricantes de hardware não podem desenvolver um driver eficiente que possa ser usado por todos os clientes [OPC, 1998b].

Desse modo, o objetivo do OPC é delimitar a fronteira entre os fabricantes de hardware e os desenvolvedores de software proporcionando um mecanismo que forneça os dados de uma determinada fonte a qualquer aplicação cliente de uma maneira padronizada. Seguindo esse conceito, um fornecedor pode desenvolver um componente servidor reutilizável e otimizado para se comunicar com suas fontes de dados (normalmente o hardware por ele desenvolvido) e disponibilizar uma interface OPC nesse servidor, a fim de que qualquer cliente possa acessar os dados dessas fontes.

A primeira versão da especificação OPC focou em funcionalidades comuns à maior parte dos fornecedores de hardware para aplicações de controle industrial, a saber:

- Acesso a dados em tempo-real, ou seja, a leitura e escrita eficiente de dados entre uma aplicação e um dispositivo de controle de processo;

- Alarmes e manipulação de eventos, ou seja, mecanismos para que os clientes OPC sejam notificados da ocorrência de eventos específicos e de situações de alarme;

- Acesso a históricos de dados, ou seja, mecanismos de leitura, processamento e edição de dados dos históricos de cada dispositivo.

As especificações OPC incluem um conjunto de interfaces COM customizadas para o uso por clientes e servidores, bem como referências a um conjunto de interfaces de Automação OLE para suportar clientes desenvolvidos a partir de aplicativos de nível alto, como Excel ou Visual Basic. A arquitetura também faz uso da tecnologia DCOM para facilitar as interfaces entre clientes e servidores remotos.

Um cliente OPC pode se comunicar com servidores OPC de diferentes fornecedores, sendo que o código fornecido pelo fabricante do hardware ou fonte de dados determina quais os dispositivos e dados aos quais cada servidor tem acesso e detalhes sobre como o servidor pode acessar fisicamente os dados (ver figura 14).

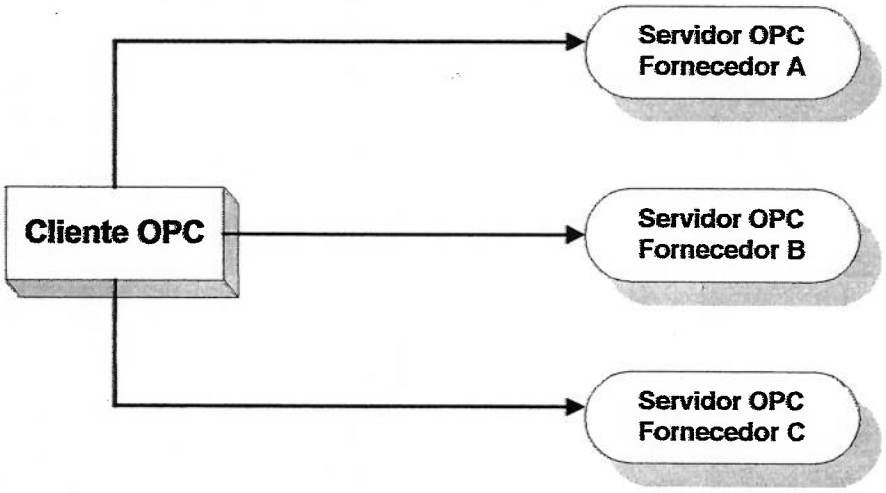

Figura 14 - Cliente OPC[12]

As interfaces OPC podem ser usadas de diversas formas em um ambiente de controle industrial: desde a extração direta de dados brutos de dispositivos físicos para um sistema SCADA ou DCS até a extração de dados de um SCADA ou DCS para uma aplicação de nível mais alto. A arquitetura possibilita a construção de um servidor OPC que permita a uma aplicação cliente acessar dados de diversos servidores OPC de fornecedores diferentes, rodando em diferentes locais da rede, através de um único objeto, conforme pode ser visto na figura 15.

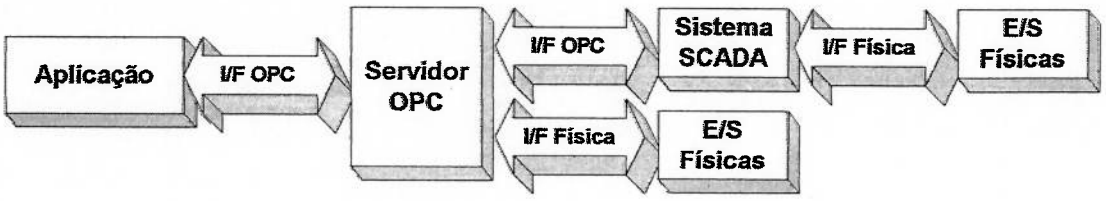

Figura 15 - Relacionamento entre clientes e servidores OPC[13]

---

[12] Fonte: [OPC, 1998b]

[13] Fonte: [OPC, 1998b]

A especificação OPC define dois conjuntos de interfaces: interfaces customizadas (*Custom Interfaces*) e interfaces de automação (*Automation Interfaces*), conforme exposto na figura 16.

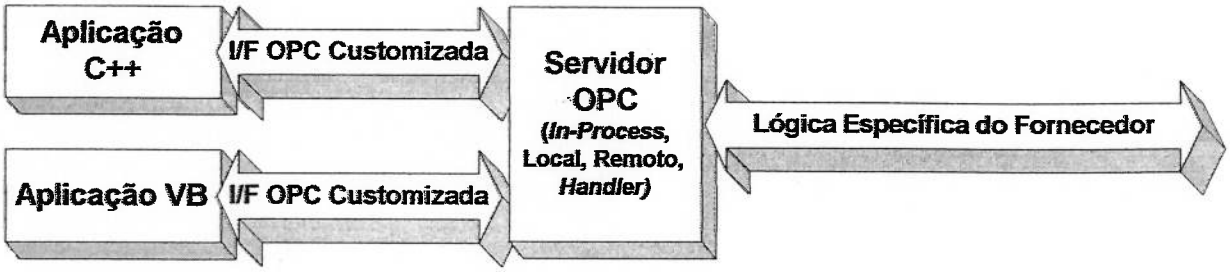

Figura 16 - Interfaces OPC[14]

De uma maneira semelhante à maneira pela qual a OMG define os componentes e serviços da arquitetura CORBA, a especificação OPC define apenas interfaces COM, sem entrar nos detalhes de implementação. Ela especifica o comportamento esperado das interfaces pelas aplicações cliente que irão acessá-las.

Da mesma forma que outras implementações COM, a arquitetura OPC segue o modelo cliente-servidor, onde o componente servidor OPC fornece uma interface para os objetos OPC e os gerencia. Uma aplicação OPC cliente se comunica com o servidor através de interfaces customizadas ou de automação e, em alguns casos, a OPC Foundation fornece um *wrapper* padrão específico para uma interface de automação. Esse *wrapper* (normalmente uma DLL) pode ser usado por qualquer servidor customizado de um fornecedor (ver figura 17).

---

[14] Fonte: [OPC, 1998b]

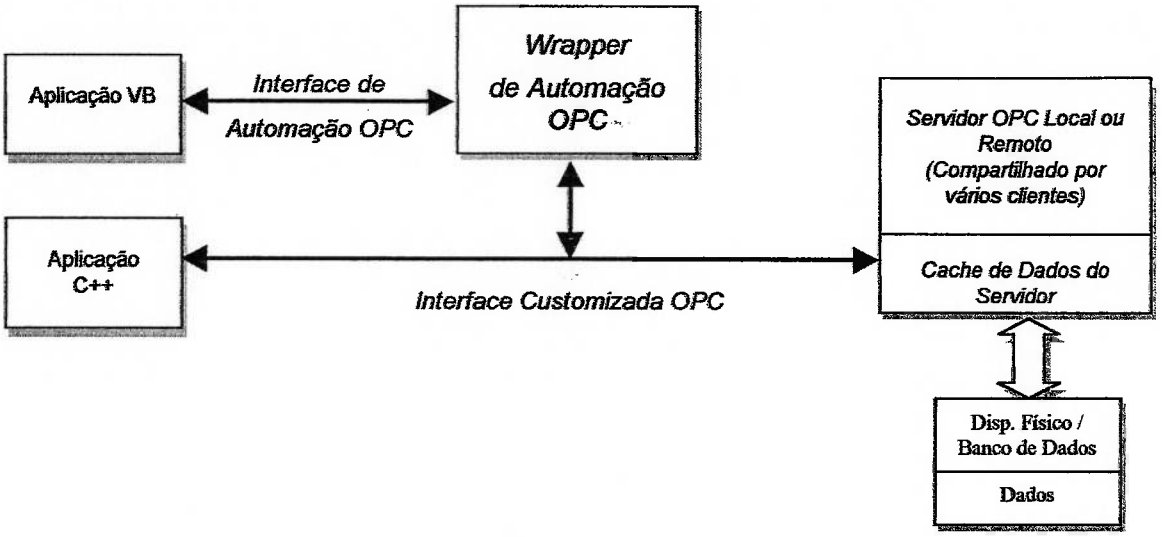

Figura 17 - Arquitetura OPC típica[15]

Existem hoje no mercado diversos pacotes SCADA e DCS que fazem uso dos protocolos OPC, como por exemplo as soluções oferecidas pelas empresas Iconics, Measurex, SAP, GE, OSI PI, Modbus, Siemens e Elipse Software [STROTHMAN, 2002a].

A OPC Foundation trabalha atualmente em uma nova geração da especificação OPC, baseada na arquitetura .NET da Microsoft e no padrão XML (*Extensible Markup Language*). Entretanto, por tratar-se de um trabalho ainda em andamento, a especificação completa não se encontra disponível, razão pela qual a arquitetura .NET não será abordada no presente texto.

Entre outros aspectos de evolução tecnológica, a nova versão da especificação OPC, conhecida por OPC XML tenciona resolver alguns problemas relacionados com segurança da versão anterior. Um exemplo simples é o bloqueio dos dados binários usados pela arquitetura COM pelos diversos *firewalls* utilizados pelas empresas: uma vez que o padrão XML utiliza código ASCII, os dados OPC deixam de ser bloqueados.

---

[15] Fonte: [OPC, 1998b]

A previsão de lançamento da especificação OPC XML é para o final do segundo semestre de 2002, embora a OPC Foundation já tenha apresentado uma prévia no evento National Manufacturing Week, ocorrido em Março de 2002.

Uma outra versão do OPC, a *OPC Data Exchange* (OPC-DX), focada na eliminação das "pontes" de dados entre os diversos protocolos de *bus* industriais (como DeviceNet, Profibus, Fieldbus e ControlNet), também encontra-se em desenvolvimento e teve uma versão preliminar apresentada na Hannover Fair alemã em Abril de 2002. Entretanto, sua conclusão não está prevista para antes do final do ano de 2002.

Embora diversos protocolos industriais de comunicação baseados em Ethernet existam, coexistam e compartilhem o mesmo meio físico, como é o caso do High-Speed Ethernet (HSE) da Fieldbus Foundation, o Ethernet/IP e o Modbus/TCP, eles são incompatíveis e incapazes de trocar informações. O padrão OPC-DX está sendo desenvolvido para resolver esta incompatibilidade, trabalhando como um *gateway* por software e substituindo os raros e caros *gateways* por hardware disponíveis hoje no mercado [BERGE, 2002].

O mecanismo de comunicação OPC-DX é do tipo servidor-servidor. Nele, os dispositivos, redes ou subsistemas conectam-se a componentes servidores OPC-DA (*Data Acquisition*[16]), que, por sua vez, conectam-se uns aos outros diretamente, através de interfaces OPC-DX (figura 18). Dessa forma, a comunicação ocorre em um esquema peer-to-peer, eliminando a necessidade de aplicações ponte para transferir os dados.

---

[16] OPC-DA – Componentes OPC voltados à aquisição de dados, ou seja, os componentes OPC tradicionais.

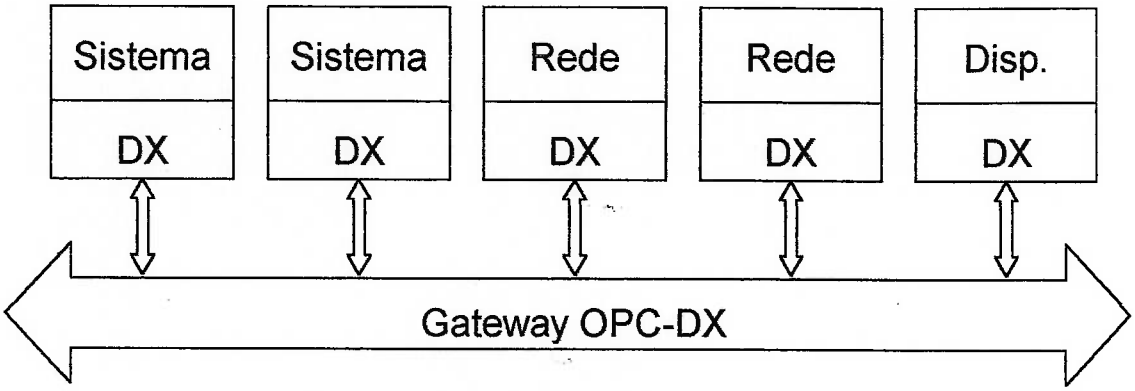

Figura 18 - Esquema de comunicação OPC-DX

A interface DX é, portanto, cliente e servidora ao mesmo tempo, agindo como fonte e consumidora de dados. Um *gateway* OPC-DX possui um componente servidor OPC-DA e um componente cliente OPC-DA, com uma interface DX. O componente servidor produz dados para as outras aplicações, e a parte cliente consome os dados de outras aplicações.

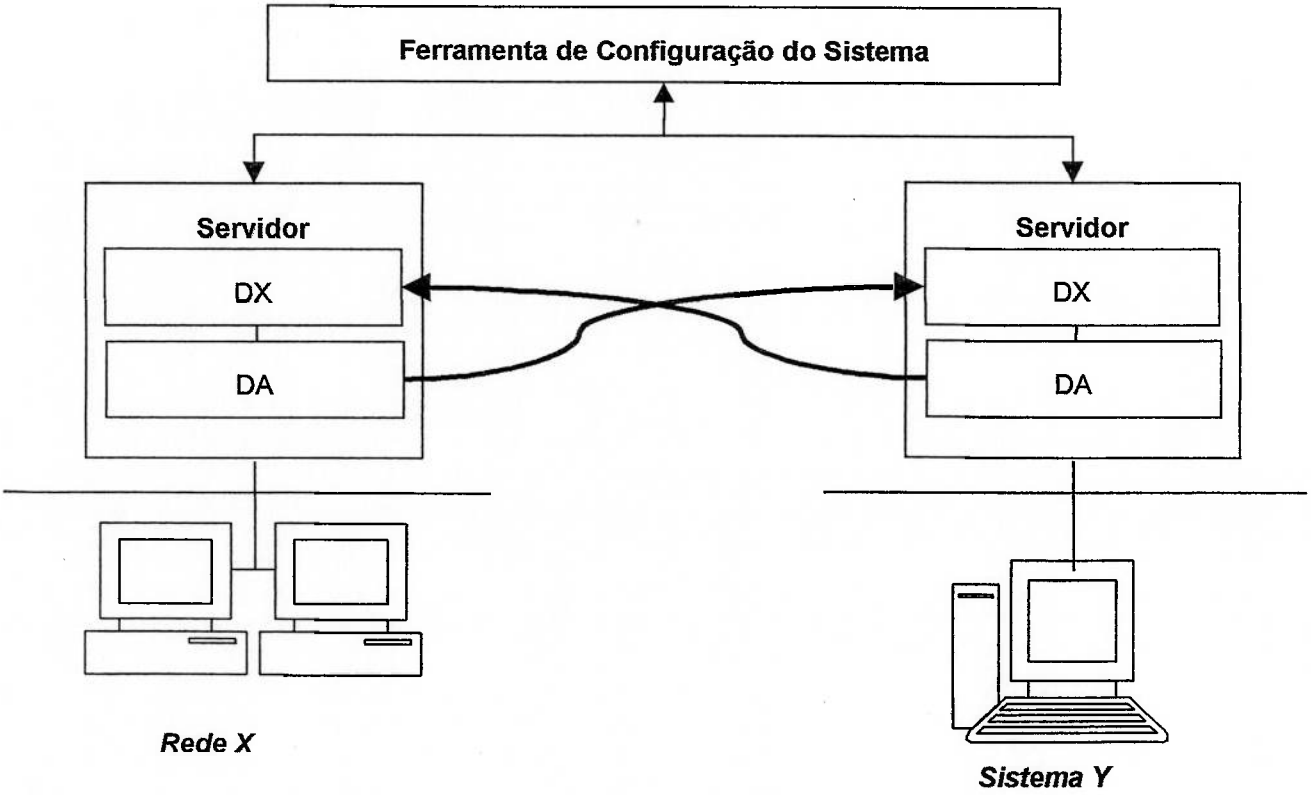

Figura 19 - Relacionamento entre OPC-DA e OPC-DX

Sendo assim, o padrão OPC-DX não é uma alternativa ao OPC clássico, mas sim um complemento. Enquanto o OPC-DA traz dados verticalmente entre um dispositivo e o software, o OPC-DX comunica dados de forma horizontal entre os dispositivos.

O padrão OPC-DX baseia-se sobre a tecnologia OLE (hoje ActiveX), COM e DCOM da Microsoft, encontradas hoje apenas em sistemas operacionais Microsoft Windows e alguns sistemas operacionais *embedded*, o que significa que uma imensa maioria de dispositivos não serão capazes de rodar o padrão OPC-DX, e precisarão de um servidor Windows operando como proxy [BERGE, 2002]. Desse modo, o OPC-DX é, hoje, um complemento aos protocolos industriais Ethernet, e não um substituto.

2.5 CORBA

2.5.1 O Paradigma da Orientação a Objetos

2.5.1.1 Classes e Objetos

Um dos conceitos básicos necessários para a correta compreensão da arquitetura CORBA, suas capacidades e suas vantagens reside no do paradigma da Programação Orientada a Objetos.

A Orientação a Objetos é uma evolução da metodologia estruturada de programação (implementada por linguagens de programação mais antigas como Pascal e C) e consiste em uma forma de se conceituar um programa de computador como sendo um conjunto de objetos, que trabalham juntos, de maneiras predefinidas, para realizar tarefas [RUMBAUGH, 1994].

Uma analogia interessante para o modo pelo qual a Orientação a Objetos vê um programa é a montagem de um computador. É possível, hoje, entrar-se em uma loja e comprar uma série de componentes independentes, para se montar um computador, como uma placa-mãe, uma CPU, um monitor, uma fonte ou um disco rígido. Cada um desses componentes é fabricado por empresas diferentes, com tecnologias diferentes, mas trabalham juntos para formar o conjunto do computador.

Com o paradigma da Orientação a Objetos ocorre algo semelhante: os programas são compostos por vários componentes (chamados objetos), que atuam de forma articulada para atender a uma determinada aplicação. Entretanto, a capacidade de combinar objetos é apenas um aspecto geral da programação orientada a objetos. Ela inclui conceitos e recursos que auxiliam na criação e o uso de objetos, como por exemplo o conceito de classes.

As classes são modelos utilizados para criar objetos com características semelhantes, e incorporam todos as propriedades de um objeto em particular. Na verdade, quando se modela em OO, não se define cada um dos objetos, mas sim as classes que servem como base para a criação desses objetos.

Por exemplo, um programa pode possuir uma classe chamada "Árvore", e que representa as características de todas as árvores:

- Possuem folhas e raízes;
- Crescem;
- Realizam fotossíntese.

Essa classe Árvore serve, portanto, como uma representação abstrata do conceito de árvore. Para ser possível usar-se um objeto do tipo Árvore em um programa, deve-se instanciar a classe Árvore, ou seja, criar-se um objeto Árvore específico, com parâmetros definidos para cada uma de suas características, como altura, formato e cor. Dessa forma, embora cada um dos objetos Árvore criados tenha suas próprias características, todos eles possuem as características básicas definidas acima, ou seja, possuem folhas e raízes, crescem e realizam fotossíntese, podendo, portanto, ser reconhecidos como árvores independentemente de sua altura ou cor.

Quando se programa de modo Orientado a Objetos, projeta-se e constrói-se um conjunto de classes. Quando o programa é executado, os objetos serão criados a partir dessas classes e usados conforme necessário.

Cada classe compõe-se de dois elementos fundamentais: os atributos e o comportamento. Os atributos são as propriedades específicas que diferenciam uma classe de objetos de outra e o comportamento é o modo pelo qual uma determinada classe pode realizar uma atividade.

No caso da classe Árvore, os atributos seriam suas propriedades fundamentais, como a altura, o número de folhas, o formato das folhas, o diâmetro do tronco e assim por diante, enquanto que o comportamento seria o conjunto de ações que podem ser realizadas pela árvore, como fazer fotossíntese, crescer e morrer. No jargão da Orientação a Objetos, essas ações seriam os métodos que compõem a classe Árvore.

Os métodos são grupos de instruções relacionadas em uma classe de objetos, que atuam sobre eles mesmos e sobre outras classes e objetos. São usados para realizar

atividades específicas, da mesma forma que as funções das linguagens de programação estruturada.

2.5.1.2 Herança

Através do conceito de herança, uma classe recebe imediatamente toda a funcionalidade de uma classe preexistente. Desse modo, uma nova classe pode ser criada apenas determinando-se as suas diferenças com relação à anterior, sendo possível organizar-se todas as classes de um sistema em uma hierarquia rigorosa.

À classe que herda as propriedades de uma outra, dá-se o nome de subclasse. Já à classe que tem suas propriedades herdadas, dá-se o nome de superclasse. Em algumas linguagens de programação (como C++), é possível que uma classe tenha mais de uma superclasse (ou seja, apresente herança múltipla), mas, em outras linguagens (como Java), cada classe pode herdar as propriedades de apenas uma superclasse (herança simples).

No exemplo da classe Árvore, podem-se definir várias subclasses, como por exemplo Árvores de Clima Quente e Árvores de Clima Frio. Ambos os tipos de árvores possuem todas as propriedades da classe Árvore, mas apresentam suas próprias características que as diferenciam (ou especializam). Quando a classe Árvore de Clima Quente for ser criada, basta que se informe que ela herda da classe Árvore, e definir-se suas propriedades específicas, sem que seja necessário informar novamente que ela realiza fotossíntese, por exemplo.

2.5.1.3 Interfaces e Encapsulamento

As interfaces consistem em pequenos pacotes de código, que definem os modelos de comportamento esperados na implementação das classes. Basicamente, as interfaces definem quais os métodos e os atributos das classes que constituem o sistema, bem como seus parâmetros de entrada e saída, sem entrar em detalhes de implementação como os algoritmos que devem ser usados ou a forma como uma determinada operação deve ser realizada.

Em algumas linguagens, como é o caso da linguagem Java, as interfaces permitem definir-se uma hierarquia completamente separada da hierarquia de classes principal.

Essa independência viabiliza a criação de classes que, embora possuam apenas uma superclasse, podem selecionar e implementar diversas interfaces com comportamentos diferentes e não fornecidos por sua superclasse.

Como as interfaces fornecem apenas definições de método abstratas, é necessário implementar-se os métodos que definem, usando-se as mesmas assinaturas de método das interfaces. Este conceito é necessário para a compreensão do papel da OMG na definição da arquitetura CORBA.

Como será visto mais à frente, toda a arquitetura de objetos distribuídos CORBA é definida apenas através das interfaces de suas classes, ficando a implementação a cargo dos programadores. Entretanto, como as interfaces usadas como base de implementação são sempre as mesmas (ou seja, as interfaces definidas pela OMG), é possível garantir-se a interoperabilidade entre as implementações de cada um dos programadores.

Sendo assim, pode-se perceber que um dos papéis mais importantes do conceito de interfaces na orientação a objetos é o de possibilitar o encapsulamento dos objetos, ou seja, não interessa para os outros objetos de um sistema saber como um determinado objeto processa cada uma de suas funções, mas sim apenas saber quais as funções que ele pode executar e quais os parâmetros de entrada e saída de cada uma delas.

2.5.1.4 Polimorfismo

Algumas vezes, uma operação pode ter o mesmo nome em classes diferentes. Por exemplo, pode-se "abrir" uma lata, uma porta, uma janela ou uma conta em um banco. Em cada caso, está sendo realizada um operação diferente mas, através da Orientação a Objetos, cada classe sabe como cada uma das operações "abrir" deve ser realizada.

Ou seja, as classes em um sistema podem possuir métodos com o mesmo nome, mas que atuam de maneira diferente sobre cada tipo de objeto. A essa propriedade dá-se o nome de polimorfismo.

Uma das utilizações mais importantes do polimorfismo está associada ao conceito de herança. Subclasses podem possuir métodos com o mesmo nome de métodos de suas

superclasses, mas com funcionalidades diferentes, de modo que as subclasses possam desempenhar atividades específicas de acordo com suas especializações.

Além disso, uma mesma classe pode possuir mais de um método com o mesmo nome, mas com parâmetros diferentes, o que possibilita que o comportamento da classe (ou do objeto) varie em função do tipo de parâmetro que será enviado ao mesmo. Por exemplo, uma função "soma" de uma determinada classe pode realizar uma adição matemática quando receber dois números inteiros como parâmetro, ou concatenar strings quando receber caracteres texto.

2.5.2 Arquitetura em camadas

Antes de serem apresentados os conceitos e definições referentes à arquitetura CORBA, faz-se necessária uma breve apresentação da história dos sistemas distribuídos, a fim de que possa-se ter a visão exata de como a arquitetura CORBA encaixa-se dentro destes sistemas.

Os primeiros sistemas de software corporativos a serem usados em larga escala no mercado eram baseados em *mainframes*, banços de dados hierárquicos e terminais "burros", sem processamento local (ver figura 20). Ainda que os *mainframes* custassem caro e demandassem muita manutenção, os mesmos eram capazes de servir a um grande número de usuários com a vantagem (ou desvantagem, dependendo do ponto de vista), de poderem ser gerenciados de uma maneira centralizada [DOBRIVOJE, 1990].

Sistemas de software escritos para *mainframes* eram normalmente monolíticos, ou seja, tanto a interface com o usuário, quanto as funcionalidades de acesso a dados, quanto a lógica do negócio estavam localizadas em uma única grande aplicação. Essa arquitetura era razoável na época, uma vez que os terminais não possuíam capacidade de processamento local.

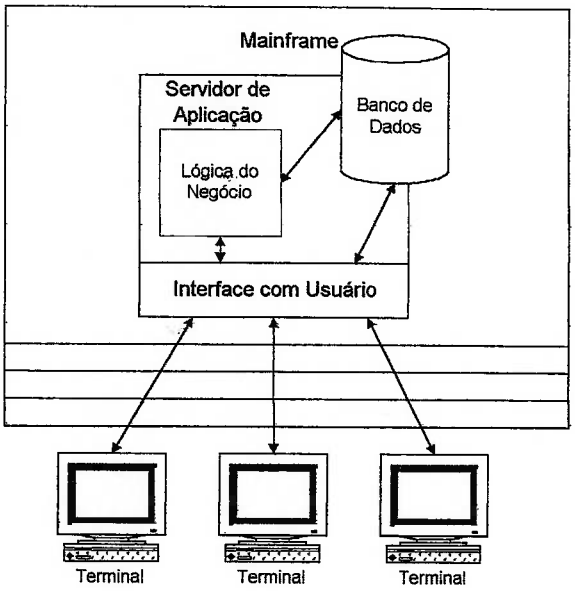

Figura 20 - A arquitetura monolítica [17]

Com o advento dos PC's, tornou-se possível a transferência de parte do processamento para dentro dos terminais, alterando-se o paradigma da arquitetura monolítica para a arquitetura conhecida por cliente/servidor. Essa fase correspondeu à proliferação dos servidores UNIX, onde residia o banco de dados, e dos PC's, onde residiam as interfaces com o usuário. A lógica do negócio passou a residir em ambos, parte no servidor de dados, parte no cliente.

A arquitetura cliente/servidor original é atualmente conhecida por arquitetura de duas camadas (*two-tier client/server*), e está representada na figura 21.

[17] Adaptado de [ROSEMBERG, 1998]

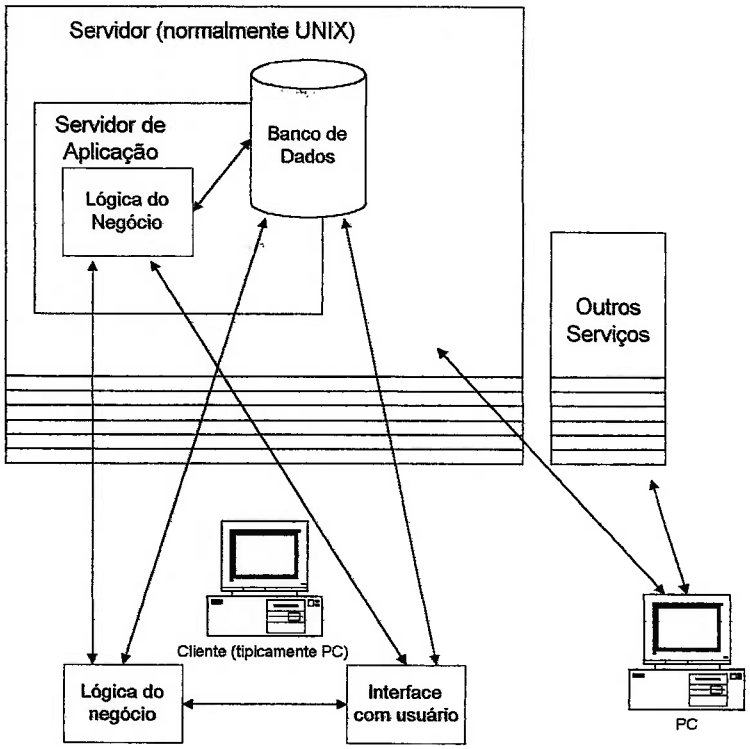

Figura 21 - Arquitetura cliente/servidor de duas camadas[18]

A arquitetura cliente/servidor de duas camadas, entretanto, trouxe consigo um novo problema: uma vez que a lógica do negócio e as funcionalidades de acesso aos bancos de dados residiam no cliente, quaisquer alterações em um desses componentes ou mesmo no banco de dados, freqüentemente exigiam alterações de versão em todos os terminais usuários da aplicação (clientes), que não raro acarretavam problemas com as versões anteriores, resultando numa fragilidade da aplicação.

Para resolver-se esse aspecto, propôs-se uma nova arquitetura multicamada. A diferença principal com relação à anterior reside no fato de que essa nova arquitetura previa o uso de qualquer número de camadas, que acabavam criando um "isolamento" entre os clientes e os bancos de dados. Dentre os esquemas possíveis de multicamada, o mais popular é o de três camadas, que particiona o sistema em três níveis lógicos: a

[18] Adaptado de [ROSEMBERG, 1998]

camada de interface com o usuário, a camada de regras de negócio e a camada de acesso à base de dados.

A arquitetura de três níveis é superior à de dois em pelo menos dois aspectos: torna a aplicação menos frágil, através do isolamento criado entre o cliente e o resto da aplicação e aumenta a granularidade da aplicação, possibilitando uma maior flexibilidade [ROSEMBERG, 1998]. Uma visão geral dessa arquitetura está apresentada na figura 22.

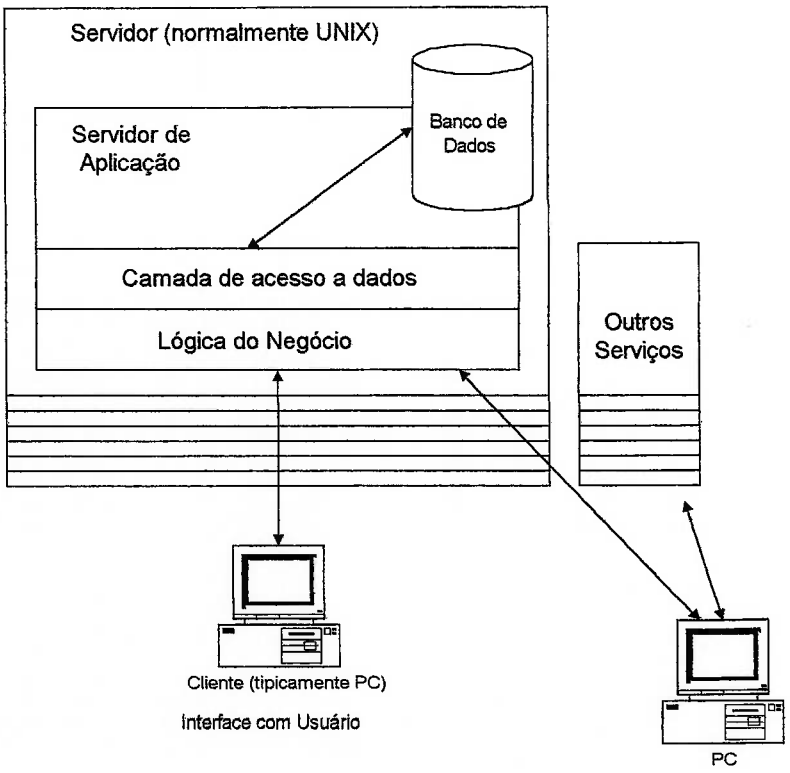

Figura 22 - A arquitetura cliente/servidor multicamada[19]

A partir da arquitetura cliente/servidor multicamada, atingiu-se finalmente o conceito de Sistemas Distribuídos conforme aceito atualmente. Nesses sistemas, cada uma das funcionalidades da aplicação é descrita como um objeto, que pode disponibilizar ou

[19] Adaptado de [ROSEMBERG, 1998]

requisitar serviços de outros objetos do sistema ou mesmo de outros sistemas, e os conceitos de "cliente" e "servidor" tendem a se misturar.

O funcionamento e a flexibilidade de sistemas distribuídos dependem das definições das interfaces dos objetos dos sistemas. Caso não se tenha uma padronização dessas interfaces, torna-se inviável a comunicação entre os diversos objetos já que, em um sistema distribuído, o número de clientes e servidores é potencialmente alto, impossibilitando a implementação de interfaces específicas para cada uma das ligações.

Os sistemas distribuídos devem incluir, ainda, serviços de diretório de objetos a fim de facilitar a localização de serviços pelos diversos componentes do sistema, e gerenciadores de transações, com a função de garantir a integridade dos objetos e dos dados passados entre eles.

2.5.3 A OMG e a arquitetura CORBA

Tendo em vista o advento das aplicações baseadas em objetos e dos sistemas distribuídos e a conseqüente necessidade de padronização de interfaces e de uma série de serviços essenciais para que essas aplicações pudessem efetivamente atingir seus objetivos, criou-se em 1989 a OMG (*Object Management Group*).

O objetivo central da OMG consiste em desenvolver uma arquitetura comum para aplicações orientadas a objetos com base em especificações de interfaces [OMG, 1995a], através do estabelecimento da OMA (*Object Management Architecture*), da qual a arquitetura CORBA faz parte. Basicamente a OMA fornece um grupo de padrões sobre os quais as aplicações são desenvolvidas, consistindo de uma função ORB (*object request broker*), serviços de objetos (CORBAservices) facilidades comuns (CORBAfacilities) [OMG, 1995b], interfaces de domínios e objetos de aplicação.

A função da arquitetura CORBA dentro da OMA é implementar a função ORB, fornecendo um mecanismo padronizado para a definição de interfaces entre componentes, além de ferramentas para facilitar a implementação dessas interfaces utilizando a linguagem de programação preferida pelo usuário. Duas das grandes propriedades da CORBA são, sem dúvida, a independência da plataforma e a independência de linguagem.

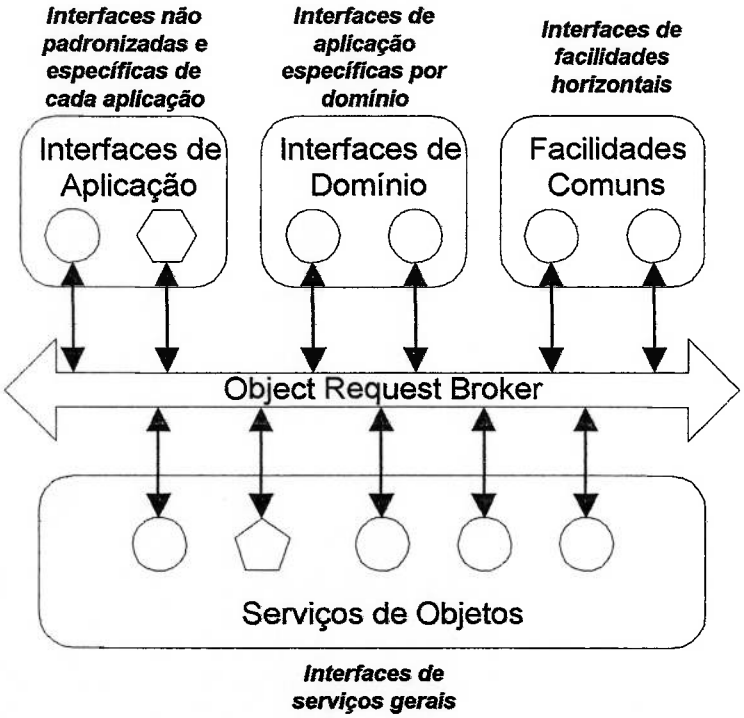

Figura 23 - Modelo de referência OMA: categorias de interfaces[20]

Em [OMG, 1995a], a OMG apresenta seus objetivos técnicos e terminologia, além de detalhar a infra-estrutura conceitual sobre a qual as especificações são detalhadas. O guia inclui o modelo de objetos da OMG, que define a semântica comum para se especificarem as características externas dos objetos de uma forma independente da implementação e o modelo de referência OMA.

A fim de complementar a OMA com especificações detalhadas de cada um dos componentes do modelo de referência, a OMG publicou uma série de RFP's (*Request For Proposals*), tendo-se estabelecido, assim, especificações para a arquitetura CORBA, os serviços CORBAservices e as facilidades CORBAfacilities.

O modelo de referência identifica e caracteriza os componentes, interfaces e protocolos que compõem a OMA, incluindo-se o *Object Request Broker* (ORB), o componente que possibilita que clientes e objetos se comuniquem em um ambiente distribuído. Além disso, o modelo de referência apresenta quatro categorias de interfaces de objetos:

- Serviços de Objetos – interfaces para serviços gerais, com potencial de uso em qualquer aplicação baseada em objetos distribuídos;
- Facilidade Comuns – interfaces para facilidades horizontais (aplicações genéricas reutilizáveis), aplicáveis à maioria dos domínios de aplicação e com foco em auxiliar o usuário na implementação;
- Interfaces de Domínio – interfaces específicas para cada domínio de aplicação;
- Interfaces de Aplicação – interfaces não-padronizadas, específicas de cada aplicação.

Todas estas categorias estão representadas no modelo de referência da figura 23.

A segunda parte do modelo de referência dedica-se ao uso destas interfaces e introduz a noção dos *Frameworks* de Objetos específicos por domínio. Esses *frameworks* são coleções de objetos que fornecem uma solução integrada dentro de uma aplicação ou domínio de aplicação, podendo ser customizados pelo desenvolvedor.

Uma das características da OMA é que as aplicações não precisam nem mesmo ser desenvolvidas segundo o paradigma da Orientação a Objetos para participar ou interagir com a arquitetura proposta, bastando para tanto que essas aplicações implementem ou suportem as interfaces estabelecidas pela OMG, de modo a serem encapsuladas em objetos que participam da OMA.

As Facilidades Comuns são interfaces para facilidades horizontais, aplicáveis à maioria dos domínios, como por exemplo ferramentas para gerenciamento de interfaces com usuário e operações com datas e horários. As Facilidades Comuns adotadas pela

---

[20] Adaptado de [HIGHLANDER, 1998]

OMG recebem o nome de CORBAfacilities e incluem uma facilidade para distribuição de documentos baseada em OpenDoc.

Tanto a especificação de uma Facilidade Comum quanto de um Serviço incluem um conjunto de definições de interfaces expressas em IDL, que os objetos devem suportar para serem capazes de proporcionar, usar ou participar da facilidade ou serviço. Como todas as especificações definidas pela OMG, as especificações de Facilidades ou Serviços não contemplam detalhes de implementação, restringindo-se a definir as interfaces.

A Interfaces de Domínio são outro exemplo. Essas interfaces aplicam-se a domínios como Finanças, Saúde, Manufatura, Telecomunicações, Transportes e Comércio Eletrônico e também são definidas apenas por meio da IDL.

2.5.4 O Object Request Broker (ORB)

O ORB, componente fundamental da arquitetura CORBA, consiste em um elemento de software cujo propósito é realizar a comunicação entre objetos. Isso é conseguido através da disponibilização de uma série de capacidades [OMG, 1998a], entre elas a capacidade de localizar um objeto remoto a partir de uma referência.

Um ORB é um mecanismo através do qual os objetos fazem requisições e recebem respostas uns dos outros, seja em uma mesma máquina, seja através de uma rede. Com o ORB, o cliente não precisa estar ciente dos mecanismos usados para a comunicação ou ativação de um objeto, as propriedades de sua implementação ou mesmo sua localização. O ORB forma, portanto, a base para se construir aplicações distribuídas e para a interoperabilidade entre aplicações em ambientes homogêneos ou heterogêneos.

2.5.5 A Interface Definition Language (IDL)

Outro componente fundamental da arquitetura CORBA é a IDL, cuja função é especificar interfaces entre objetos CORBA, sendo um dos fatores responsáveis pela independência de linguagem proposta pelo padrão.

A IDL é a linguagem padrão utilizada para definir-se interfaces entre componentes das aplicações. Entretanto, a IDL não é uma linguagem procedural, ou seja, pode

definir apenas as interfaces, e não as implementações. Pode-se comparar a IDL aos arquivos de cabeçalhos (*headers*) utilizados na programação em C++. Um arquivo de cabeçalho normalmente não possui a implementação das classes, restringindo-se a descrever suas interfaces.

A especificação da IDL é responsável por assegurar que os dados sejam corretamente trocados entre linguagens de programação diferentes [ROSEMBERG, 1998]. Por exemplo, a IDL possui o tipo *long*, que consiste em um número inteiro de 32 bits com sinal e que pode ser mapeado para um tipo *long* da linguagem C++ ou para um tipo *int* na linguagem Java.

Uma vez que todas as interfaces definidas pela arquitetura CORBA são descritas via IDL, podem ser mapeadas para qualquer linguagem de implementação, ou seja, servidores escritos em C++ podem comunicar-se com clientes escritos em Java, que podem comunicar-se com servidores escritos em COBOL, e assim por diante.

A IDL, da mesma maneira que a maioria das linguagens de programação, define tipos primitivos que incluem números inteiros, caracteres, números de ponto flutuante, *strings*, tipos booleanos e constantes, entre outros. Além disso, os tipos primitivos podem ser agregados em outros tipos, como *unions* e *structs*.

São definidos também pela especificação da IDL formas de representação dos parâmetros de entrada e saída, métodos, módulos, atributos, herança entre classes e diversas outras ferramentas necessárias para a descrição completa das interfaces entre os diversas classes de um sistema [EXTON, 1997].

Por exemplo, caso se deseje implementar um servidor que simplesmente conte o número de vezes em que ele é acessado, pode-se definir a seguinte interface através da IDL definida na figura 24.

Esse exemplo de IDL define uma interface "Conta" dentro de um módulo CORBA chamado "Contador" possuindo um método "incrementa", que poderia ser responsável por incrementar a contagem dos acessos e um atributo "soma", com a função de armazenar o resultado da contagem.

```
module Contador
{ interface Conta
  { attribute long soma;
    long incrementa ();
  };
};
```

Figura 24 - Exemplo de IDL

Uma vez que se compile essa interface através dos pré-compiladores de IDL disponíveis em qualquer ferramenta CORBA, serão criadas automaticamente uma série classes implementando a infra-estrutura básica necessária para a comunicação desse servidor com seus clientes através da arquitetura CORBA. Após esta pré-compilação pode-se então implementar as porções de código necessárias para que o servidor de contagem de acessos realize a tarefa a que se destina.

Um caso real detalhado, apresentando como a definição clara das IDL's pode ser usada para viabilizar a interoperabilidade entre componentes de software desenvolvidos por equipes de programadores distintas, utilizando linguagens de programação e implementações CORBA distintas pode ser encontrado em [XING, 1999].

2.5.6 Stubs e Esqueletos

Da mesma maneira que outras arquiteturas, a CORBA mantém as noções de cliente e servidor, sendo que qualquer objeto CORBA pode trabalhar em ambas as condições. No caso de um objeto que esteja disponibilizando serviços para outros objetos, ele é considerado um servidor e, no caso de um objeto que acessa serviços de outro, ele é considerado cliente.

Para a implementação de clientes, utiliza-se o conceito de stubs. Um stub é um trecho de código que permite a um cliente ter acesso a um componente servidor, sendo compilado juntamente com a porção "cliente" da aplicação. Similarmente, os esqueletos são trechos de códigos que devem ser preenchidos quando um servidor é

implementado. Não é necessário codificarem-se manualmente os stubs e os esqueletos; eles são gerados quando a interface IDL é compilada.

2.5.7 Os Modelos de Comunicação e de Objetos CORBA

A fim de facilitar a comunicação entre objetos, a arquitetura CORBA utiliza a noção de referências de objetos, ou IOR's (*Interoperable Object References*). Quando um componente de uma determinada aplicação deseja acessar um objeto CORBA, ele inicialmente obtém um IOR para esse objeto, através do qual irá ser capaz de invocar métodos no objeto. Nesse momento, o objeto torna-se um servidor.

Para a CORBA, um cliente é simplesmente uma aplicação que esteja usando os serviços de um objeto CORBA, ou seja, que esteja invocando métodos em outro objeto.

Os ORB's normalmente comunicam-se através do Internet Inter-ORB Protocol (IIOP). Existem outros protocolos para comunicação entre ORB's, mas o IIOP é o mais popular deles [ORFALI, 1998], devido à popularidade do protocolo TCP-IP, uma camada que se localiza logo abaixo da camada IIOP. Todavia, a CORBA é independente de protocolo de rede e poderia trabalhar com qualquer outro deles.

A visibilidade dos objetos CORBA é realizada apenas através da passagem de referências aos objetos, não podendo ser possíveis as passagens por valor (pelo menos até a versão 2.0). Sendo assim, os objetos remotos CORBA permanecem remotos, não sendo possível atualmente que um objeto copie-se para outro local.

A arquitetura CORBA define o BOA (*Basic Object Adapter*), que realiza a interface entre os objetos e seus ORB's, através da implementação de uma série de métodos para o acesso às funções do ORB, como a autenticação da ativação de um objeto ou a persistência de um objeto.

Na terminologia CORBA, o BOA é um pseudo-objeto, ou seja, é um objeto que é criado diretamente pelo ORB, mas que pode ser invocado diretamente por qualquer outro objeto. O BOA disponibiliza operações que os objetos servidores de um sistema podem acessar e também faz a interface com o ORB e com os esqueletos. Entretanto, esta comunicação do BOA com o ORB e os esqueletos é feita através de interfaces

privadas, o que significa que o BOA é específico para cada implementação de ORB. Esse foi um ponto que a OMG solucionou com a introdução do POA na versão 3.0 da especificação da arquitetura CORBA.

As limitações do BOA decorriam do fato de ele haver sido subespecificado na versão 2.0 da arquitetura CORBA [ORFALI, 1998]. Para a versão 3.0 da especificação, a OMG optou por reescrever completamente a especificação do BOA, uma vez que as diferenças entre as visões e versões do BOA de cada fornecedor de software eram muito grandes e uma tentativa de consertar o modelo original levaria a incompatibilidades entre os ORB's então disponíveis [ORFALI, 1998].

Desse modo, ao invés de correr o risco de causar um imenso problema de migração do BOA da versão 2.0 para uma nova versão, a OMG decidiu deixar o BOA do jeito que estava e criar uma versão portável, o POA (*Portable Object Adapter*). Atualmente, grande parte dos fornecedores de ferramentas CORBA mantém os dois adaptadores (BOA e POA), de modo a manter a compatibilidade de sistemas antigos com as novas versões de ORB's e também atender à nova especificação da OMG.

2.5.8 Serviços de Objetos

Caso a OMG houvesse definido apenas o ORB e a especificação de interfaces para os objetos com a IDL, ainda haveria diversos aspectos em aberto para que fosse possível o desenvolvimento de aplicações CORBA úteis com objetos distribuídos, e que precisariam ser solucionados pelos usuários do padrão. Para auxiliar o trabalho de desenvolvimento, foram criados e definidos os chamados Serviços de Objetos, que consistem em serviços fundamentais que fornecem a base para o desenvolvimento de aplicações.

Os Serviços de Objetos são os elementos básicos para aplicações baseadas em objetos distribuídos, podendo ser combinados de diversas formas, atendendo a necessidades como a localização de objetos, controle de persistência, controle de concorrência, tratamento de eventos, e outras mais. Os Serviços de Objetos já adotados pela OMG são chamados de uma maneira geral de CORBAservices, e incluem os serviços de Nomes, Eventos, Ciclo de Vida, Persistência, Transações, Controle de Concorrência,

Relacionamentos, Externalização, Licenciamento, Propriedades, Segurança e Tempo. [OMG, 1998b] e [OMG, 2000a].

Alguns desses serviços, relevantes para o presente trabalho, serão descritos com maior detalhe a seguir.

2.5.8.1 Serviço de Nomes

Quando um componente de uma determinada aplicação deseja acessar um objeto CORBA, ele inicialmente precisa obter um IOR para este objeto, através do qual será capaz de invocar seus métodos. A maneira mais básica de se trabalhar com os IOR's é gerando pequenos arquivos texto com nomes predefinidos e escrevendo-se os IOR's nesses arquivos. Quando um objeto necessitar identificar um segundo objeto, pode abrir o arquivo gerado e ler seu IOR, para então ser capaz de interagir com ele. Entretanto, para que esse método funcione, todos os objetos interessados em escrever IOR's devem fazê-lo em locais predeterminados e a que todos os outros objetos interessados em localizá-los tenham acesso. Ou seja, o objeto servidor tem sempre que escrever seu IOR em uma área de disco conhecida pelo cliente e à qual o cliente tenha acesso de leitura.

Para solucionar o problema da localização de objetos, foi definido pela OMG o Serviço de Nomes [OMG, 1998b]. Esse serviço visa associar nomes aos objetos, através das chamadas "*name bindings*", ou associações de nomes. Essas associações são sempre definidas dentro dos chamados "contextos de nomes", que são objetos que contêm um grupo de associações de nomes, sendo que cada associação deve ser única dentro de um dado contexto. Nomes diferentes podem ser associados ao mesmo objeto, em um mesmo contexto ou em contextos diferentes, mas não há a obrigatoriedade de que todos os objetos recebam nomes.

Existem duas ações básicas disponibilizadas aos objetos pelo serviço de Nomes. A primeira delas é a de associação de um nome (ou *bind*), que consiste em associar-se um nome a um objeto em um determinado contexto; a segunda é a de localização de um nome (ou *resolve*), que consiste em determinar-se o objeto associado a um nome

em um dado contexto. Cada nome sempre é localizado em função de um contexto, sendo que não pode haver nomes absolutos.

Uma vez que um contexto é um objeto como outro qualquer, ele também pode ser associado a um nome em um contexto de nomes. Essa associação de nomes a contextos em outros contextos cria um grafo de nomes, em que os nós são contextos, e que possibilita a criação de nomes compostos, de modo que uma seqüência de nomes possa referenciar um objeto. Essa estrutura de nomes compostos funciona de maneira análoga à árvore de diretórios utilizada por sistemas operacionais como DOS, Windows ou Unix, sendo que pode-se fazer uma analogia clara entre os contextos e os diretórios, e os nomes dos arquivos e os nomes dos objetos.

Como todos os serviços definidos pelos CORBAservices da OMG, o serviço de Nomes só é especificado através de sua interface. Isso significa que a OMG não fornece código-fonte aos fornecedores e desenvolvedores de software, fornece apenas as IDL para que esses implementem suas próprias soluções para o serviço. Ao fornecer as IDL e definir algumas regras básicas de implementação que precisam ser seguidas, a OMG garante que as empresas interessadas em desenvolver produtos baseados no Serviço de Nomes façam-no mantendo a padronização das funções básicas, dos nomes dos módulos e dos nomes das funções de cada módulo, viabilizando a interoperabilidade entre os diversos fornecedores de soluções.

2.5.8.2 Serviço de Eventos

A cada chamada CORBA entre um objeto e outro são normalmente passados parâmetros e retornadas respostas, ou seja, há comunicação de dados entre o objeto cliente e o objeto servidor. Uma chamada padrão resulta na execução de uma operação pelo objeto servidor, realizada de forma síncrona, e, para que a chamada seja bem-sucedida, tanto o objeto cliente quanto o servidor devem estar "no ar" e disponíveis, havendo exceções a serem levantadas caso essa condição básica não se verifique.

Entretanto, as chamadas síncronas não resolvem a totalidade das necessidades de um programador ou desenvolvedor que esteja trabalhando em um sistema distribuído

[OMG, 1998b]. Existem muitas situações em que é importante desacoplar-se a comunicação entre os objetos cliente e servidor, seja para melhorar-se o desempenho do sistema (tornando o processamento assíncrono), seja para melhorar-se sua confiabilidade, ou por particularidades do problema a ser solucionado pelo sistema.

A fim de promover o desacoplamento entre os objetos cliente e servidor na comunicação entre os mesmos, a OMG definiu o Serviço de Eventos [OMG, 1998b] que possibilita a comunicação assíncrona entre objetos. O Serviço de Eventos define dois novos papéis para os objetos: o de Fornecedor e o de Consumidor.

Os objetos Fornecedores são aqueles que geram os eventos (que produzem os dados) e os Consumidores são aqueles que processam esses eventos ou dados, que lhes são passados através de requisições CORBA.

A comunicação de eventos entre Fornecedores e Consumidores pode ocorrer de duas formas distintas: através do modelo *Push* e do modelo *Pull*. O modelo *Push* (figura 25) dá ao Fornecedor a capacidade de iniciar a transferência de dados de eventos aos Consumidores, sendo que é o Fornecedor quem toma a iniciativa da comunicação. Já o modelo *Pull* (figura 26) dá ao Consumidor a capacidade de requisitar dados do Fornecedor a qualquer momento, sendo que é o Consumidor quem toma a iniciativa da comunicação.

Todo o gerenciamento da comunicação de dados entre Fornecedores e Consumidores, bem como o gerenciamento das requisições é realizado por um objeto independente chamado de *Event Channel* (ou Canal de Eventos). O Canal é simultaneamente um Consumidor e um Fornecedor e viabiliza o assincronismo das trocas de dados, sendo que toda a comunicação é feita através desse e não diretamente entre os Consumidores e os Fornecedores. Os Canais são objetos CORBA padrão e a comunicação com eles é feita através de chamadas CORBA comuns.

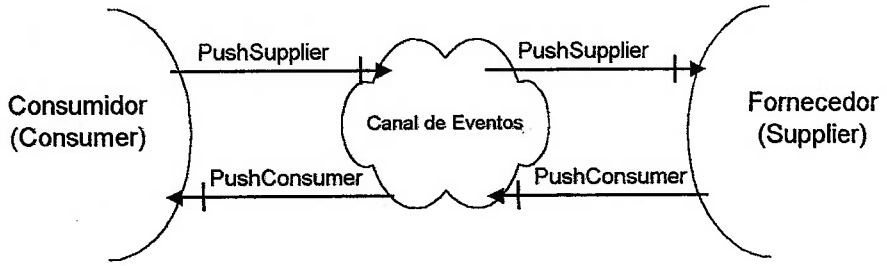

Figura 25 - Comunicação tipo *Push*[21]

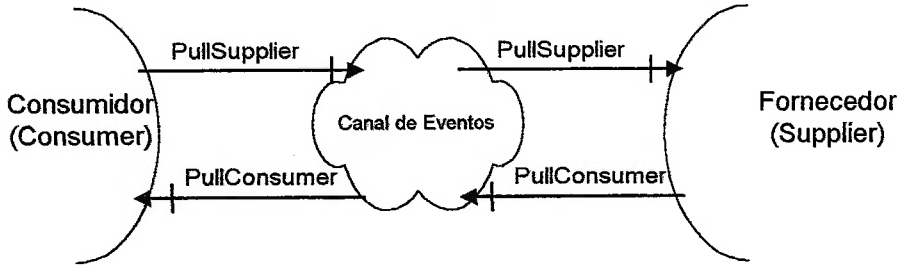

Figura 26 - Comunicação tipo Pull[22]

O processo de criação do mecanismo de comunicação através do serviço de Eventos é iniciado com a instanciação de um canal de eventos, que suporta a interface padrão *EventChannel* definida pela especificação do serviço de Eventos escrita pela OMG. Esse objeto suporta basicamente 3 operações distintas: uma operação que retorna um administrador de clientes (ou *ConsumerAdmin*), uma operação que retorna um administrador de fornecedores (ou *SupplierAdmin*), e uma operação que destrói o canal.

Os administradores atuam no sentido de possibilitar a adição de clientes e fornecedores ao canal de eventos. Isso é feito através de *proxies*, que são conectores usados para ligar-se os objetos externos (clientes e fornecedores) ao canal, sem que eles tenham contato direto com o objeto canal.

O administrador de clientes possui um método que retorna um *proxy* fornecedor (definido pelas interfaces ProxyPushSupplier ou ProxyPullSupplier, dependendo do

---

[21] Adaptado de [OMG, 1998b]

[22] Adaptado de [OMG, 1998b]

modelo de comunicação escolhido). Esse proxy atua como se fosse um fornecedor de eventos normal (inclusive herdando a interface de um fornecedor), mas inclui um método que permite a conexão de um cliente a ele.

O mesmo ocorre com o administrador de fornecedores. O SupplierAdmin possui um método que retorna um proxy consumidor (definido pelas interfaces ProxyPushConsumer ou ProxyPullConsumer), que atua como o cliente normal, mas possui um método que possibilita a conexão de um fornecedor.

Desse modo, o processo de conexão de uma aplicação fornecedora ao canal de eventos transcorre em duas etapas: inicialmente a aplicação geradora de eventos obtém um proxy consumidor do canal, através do administrador de fornecedores, e depois conecta-se a esse proxy. Da mesma forma, a aplicação cliente primeiro precisa obter um proxy fornecedor do canal e depois conectar-se a ele para ter acesso ao canal de eventos.

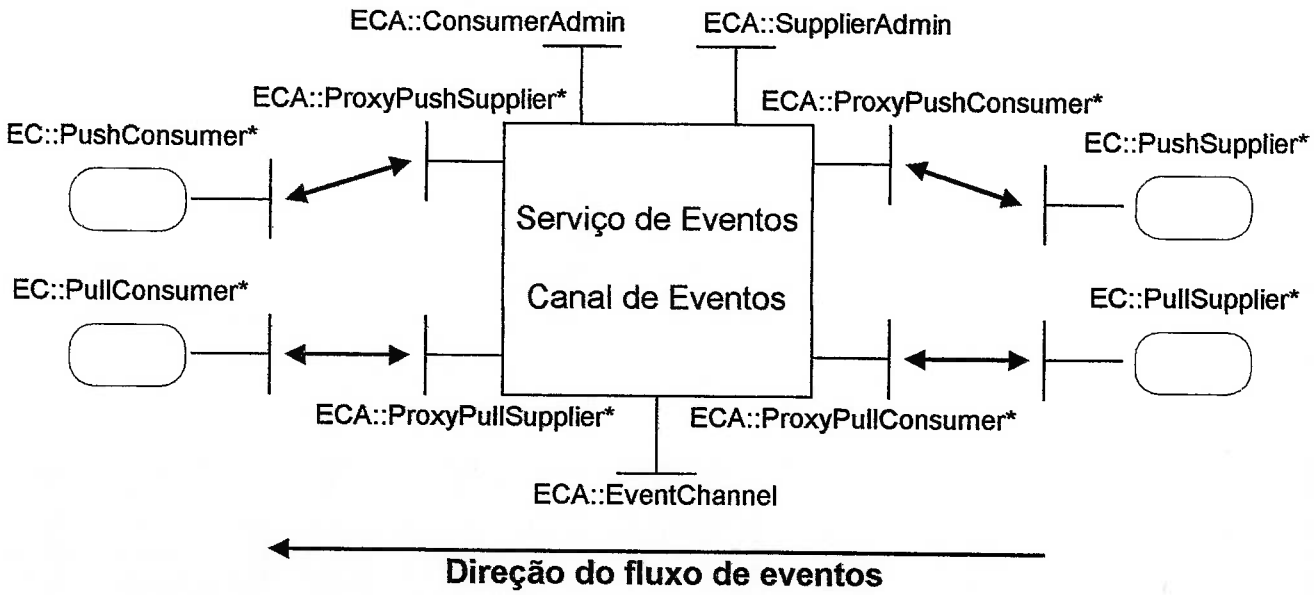

Figura 27 - Arquitetura de um canal de eventos (sem tipos definidos) definida pela OMG[23]

Como pode ser visto na figura 27, a especificação da OMG define as interfaces para a obtenção dos *proxies* consumidores e fornecedores, para os modelos de comunicação do tipo *Push* e *Pull*, resultando num total de 4 interfaces de *proxies* diferentes a serem implementadas pelo serviço de Eventos. Entretanto, existe ainda o processo de comunicação de eventos com tipos definidos, o que implica a definição de uma outra estrutura, análoga à acima, resultando em mais quatro interfaces de *proxies*. Toda a comunicação dos eventos é realizada através desses proxies, de modo que os clientes e fornecedores permanecem completamente desacoplados, ou seja, os objetos clientes e fornecedores não executam diretamente métodos uns dos outros.

[23] Adaptado de [OMG, 1998b]

Um exemplo de aplicação do serviço de eventos seria uma situação em que um determinado sistema precisa fornecer uma leitura de um sensor a diversos painéis espalhados por uma fábrica. Em vez de se realizarem as leituras a cada painel individualmente pode-se, através do serviço de Eventos, criar-se um Canal de Eventos em que cada um dos painéis é um Consumidor, o sistema que lê o sensor é o Fornecedor, e a comunicação é realizada no modelo *Push*. Nesse sistema, o Fornecedor tem apenas que enviar a informação da leitura para o Canal, através de um *proxy* consumidor e o mesmo se encarrega de distribuí-la entre todos os Consumidores.

A estrutura de relacionamento entre as classes definidas na especificação do Serviço de Eventos pode ser vista na figura 28.

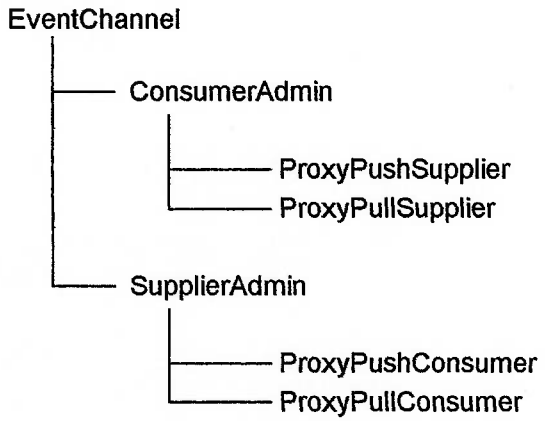

Figura 28 - Relacionamento entre as classes definidas na especificação do Serviço de Eventos[24]

Uma das características que pode ser percebida nesse exemplo, quando comparado a uma chamada CORBA, reside no assincronismo do processo. O Fornecedor não precisa invocar métodos em cada Consumidor para informá-los da nova leitura, bastando enviar o dado ao Canal. Isso evita problemas potenciais como a falha na comunicação com um Consumidor, que poderia levar o Fornecedor a ter de aguardar o retorno da comunicação (ou um período de *timeout*), antes de passar para a comunicação com o Consumidor seguinte.

---

[24] Adaptado de [OMG, 1998b]

Cabe aqui uma observação com relação ao serviço de Eventos. Nas especificações da OMG para esse serviço [OMG, 1998b] não fica definida qual a técnica a ser empregada para o controle das entregas das mensagens, ou seja, para o controle da Qualidade do Serviço (*Quality of Service – QoS*). Isso significa que cada empresa que implementa o serviço de eventos pode utilizar um método diferente, embora todas elas devam implementar ao menos a técnica da entrega de mensagens "o mais rapidamente possível". Desse modo, fica a critério da empresa ou pessoa que implementar o serviço o que fazer quando uma mensagem não puder ser entregue de imediato, como gerenciar filas de mensagens, e outras questões inerentes ao serviço.

2.5.8.3 Serviço de Notificação

Apesar de o Serviço de Eventos contribuir substancialmente para a agilização da implementação de comunicação assíncrona entre fornecedores e consumidores, há situações em que esse serviço apresenta deficiências de especificação. Especialmente em uma aplicação com fortes requisitos de confiabilidade, como é o caso de um sistema de controle e supervisão de chão-de-fábrica, existem algumas necessidades que não podem ser ignoradas, entre elas:

- A garantia da Qualidade de Serviço, ou seja, a garantia de que os eventos serão entregues aos consumidores segundo uma prioridade predefinida, e que o sistema será informado caso seja impossível entregar-lhes no prazo;
- Um sistema de filtragem, de modo que cada cliente só receba os eventos que realmente lhe interessem. Por exemplo, um cliente que supervisiona o nível de um tanque pode desejar receber o valor do nível constantemente, apenas a cada variação, ou apenas quando um dado valor for excedido. Isso reduz consideravelmente o tráfego de dados na rede, colaborando com a velocidade da comunicação e descarregando os clientes;
- A capacidade de se transmitir dados estruturados, e não apenas tipos simples, como acontece no serviço de Eventos;

Todas essas necessidades são atendidas pelo Serviço de Notificação [OMG, 2000a], que é uma extensão do serviço de Eventos.

O objetivo principal do Serviço de Notificação é o de aprimorar o Serviço de Eventos, introduzindo os conceitos de filtragem e configurabilidade, de acordo com vários requisitos de Qualidade de Serviço. Os clientes do Serviço de Notificação podem assinar eventos específicos, associando filtros aos *proxies* que usam para se comunicar com os canais de eventos. Os filtros encapsulam restrições que determinam os eventos em que o cliente está interessado, de modo que o canal apenas entrega eventos correspondentes aos interesses expressos pelos clientes.

O Serviço de Notificação foi projetado para ser completamente compatível com o Serviço de Eventos, suportando todas as interfaces nele definidas. Os modelos de entrega do Serviço de Notificação também permanecem os mesmos (Push e Pull), mas existe uma série de novas interfaces, como pode ser visto no diagrama de hierarquia dos objetos apresentado na figura 29, onde os elementos marcados com "*" são elementos já presentes no Serviço de Eventos :

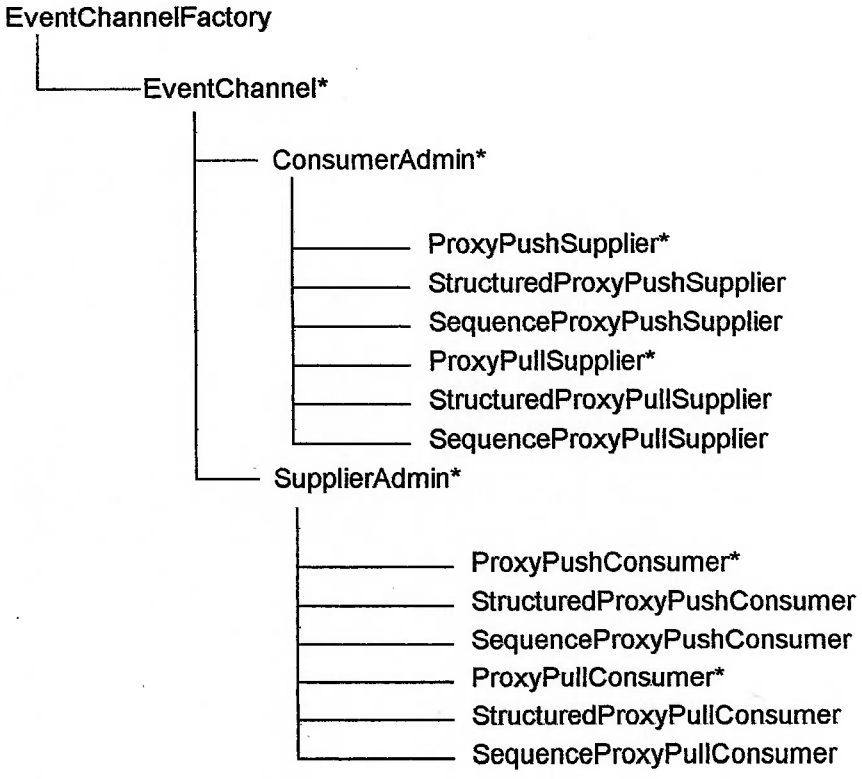

Figura 29 - Relacionamento entre os objetos definidos na especificação do Serviço de Notificação[25]

[25] Adaptado de [OMG, 2000a]

A primeira diferença que pode ser identificada quando se compara esse diagrama com o diagrama apresentado para o Serviço de Eventos está na presença de uma interface chamada EventChannelFactory, ou a Fábrica de Canais de Eventos. Na verdade, a especificação do serviço de Eventos já deixava subentendida a necessidade para uma fábrica, responsável por criar as instâncias dos Canais de Eventos, mas não definia explicitamente suas interfaces. Isso implicava o desenvolvimento de Fábricas com interfaces próprias por parte de cada um dos desenvolvedores de software que desejassem implementar e comercializar uma versão do Serviço de Eventos.

No Serviço de Notificação há uma interface chamada EventChannelFactory, com a função de criar instâncias dos canais de notificação, suportando a definição de parâmetros de Qualidade de Serviço (QoS) e de propriedades administrativas, como o número máximo de eventos que um canal pode acumular ou o número máximo de clientes e fornecedores que podem se conectar ao canal simultaneamente.

Uma particularidade introduzida pelo serviço de Notificação é que todos os objetos que criam outros objetos (como a Fábrica de Canais, os Administradores e os próprios Canais), devem associar identificadores numéricos únicos aos objetos que criarem e também devem possuir uma operação que receba como parâmetro de entrada o número de identificação de um objeto criado por eles e retorne uma referência para este objeto. Estas duas ferramentas contribuem para a melhor administração do serviço de Notificação.

Ao compararem-se as interfaces definidas pelos serviços de Notificação e de Eventos, verifica-se que o canal de Eventos, definido pela interface EventChannel, existe em ambos os casos. Desse modo, uma interface EventChannel pode ser tratada simplesmente como um canal definido pelo Serviço de Eventos, ou possuir parâmetros de QoS e propriedades administrativas associadas, como é esperado em um canal do Serviço de Notificação.

Uma outra particularidade no serviço de Notificação é a sua capacidade de suportar instâncias múltiplas das interfaces ConsumerAdmin e SupplierAdmin (ou administradores). Os administradores podem ser vistos como fábricas de proxies e podem possuir suas próprias propriedades de QoS e elementos de filtro. Os

parâmetros de QoS são associados a cada Proxy em sua criação, mas podem ser modificados caso a caso, em cada Proxy criado pelo administrador, enquanto que os filtros são sempre os mesmos para todos os Proxy criados por um determinado administrador. Filtros adicionais podem ser associados diretamente a cada Proxy, mas os filtros criados por um Administrador sempre se aplicam a todos os seus Proxy "filhos".

O ganho principal em terem-se múltiplos administradores para cada canal está na otimização do tratamento de clientes com requisitos idênticos. Por exemplo, se há a necessidade de conectarem-se múltiplas aplicações clientes com interesse em receber um mesmo conjunto de eventos a um canal de notificação, isso poderia ser conseguido através da criação de um único objeto do tipo ConsumerAdmin, e associando ao mesmo os filtros que definem os requisitos dos clientes. Isso feito, cada cliente poderia então conectar-se ao canal usando este objeto ConsumerAdmin para criar os seus objetos Proxy Supplier.

2.6 Produtos MES disponíveis no mercado

Existe hoje disponível no mercado uma grande variedade de soluções MES, dedicadas à utilização nos mais variados tipos de processos de manufatura, da manufatura discreta aos sistemas contínuos e por batelada.

Em uma primeira pesquisa foram identificados 28 empresas com oferta de produtos MES "de prateleira", ou seja, pacotes já disponíveis comercialmente para implantação. São elas:

Tabela 2 - Relação de produtos MES disponíveis no mercado

	Empresa	*Produto*	*Website*	*OMG*	*OPC*
1	ABB	Industrial[11]	www.abb.com	X	X
2	Datasweep	Advantage	www.datasweep.com		
3	AspenTech	AspenTech Mfg. Suite	www.aspentech.com		X
4	GE Fanuc	Cimplicity	www.gefanuc.com	X	X
5	Ci Technologies	Citect / Plant2Business	www.citect.com		X
6	Siemens ORSI	Cube Ind. Framework Suite	www.siemens-orsi.com	X	X
7	Brooks Automation	FactoryWorks	www.brooks-pri.com	+	
8	Teradyne	GR Force/SCE	www.teradyne.com	+	
9	Meteor-IT	Hacon Primas	www.meteor-it.nl		
10	HMS Software	HMS	www.hmssoftware.com		
11	Cimnet Inc.	Folders / Factelligence	www.cimnetinc.com		
12	Camstar	InSite	www.camstar.com		
13	Intellution	Intellution Dynamics / iFIX	www.intellution.com		X
14	Mware	Mware MES	www.mware-mes.com		
15	Real World Tech.	OnTrack	www.rwtcorp.com		
16	Optal Ltd.	OPTAL	www.optalmes.com		
17	OSIsoft	OSI-PI	www.osisoft.com		X
18	Interact Indl. Aut.	PASSy	www.interact-automation.nl		
19	Werum Software	PAS-X	www.werum.de		
20	Rockwell Propack	PMX	www.propack-data.com	X	X
21	Honeywell	POMS	www.poms.com	X	X
22	Mountain Systems	Proficy for Manufacturing	www.mountainsystems.com		
23	IBASEt	Solumina	www.ibaset.com		
24	IBM	SuperPoseidon	www.ibm.com	X	
25	SynQuest	Virtual Production Engine	www.synquest.com		
26	Invensys	Wonderware InTrack	www.wonderware.com		X
27	USData	Xfactory / FactoryLink	www.usdata.com		X
28	Elan Software	XFP	www.elansoftware.com		

Procurou-se identificar nas empresas relacionadas acima a utilização de tecnologias CORBA ou COM/DCOM nos produtos disponíveis. Entretanto, como todas elas fazem uso de algum modo das tecnologias COM/DCOM, optou-se por comparar o uso da tecnologia CORBA com o uso de OPC.

Desse modo, as duas últimas colunas da tabela identificam as empresas filiadas à OMG, à OPC Foundation, ou a ambas. As empresas identificadas com "X" são filiadas à entidade e oferecem pelo menos um produto baseado na tecnologia em questão. Já as indicadas com "+" não são filiadas à entidade em questão, mas foi possível identificar-se pelo menos um produto baseado em OPC ou CORBA.

Todas essas empresas classificam seus produtos como *Manufacturing Execution Systems*, e diversas oferecem também soluções SCADA, normalmente como um subconjunto dos pacotes MES.

Entretanto, em algumas situações, os pacotes SCADA podem ser dedicados a um mercado vertical específico, como é o caso dos produtos voltados ao mercado de geração e distribuição de energia elétrica. Em função das particularidades envolvidas no gerenciamento das fases de geração e distribuição de energia elétrica, os pacotes SCADA para essa finalidade apresentam arquitetura particular e específica, como pode ser observado em [TOMOMICHI, 1997]. Por não tratar-se de um mercado que envolva manufatura de produtos, esses produtos SCADA não são, nem podem ser considerados subconjuntos de ferramentas MES.

Uma situação análoga ocorre também no domínio da automação predial, como pode ser verificado através do modelo básico de sistema proposto em [PENNER, 2002].

Das empresas relacionadas na tabela acima pode-se obter uma idéia da distribuição de mercado entre as duas tecnologias, no que se refere ao número de desenvolvedores de sistemas MES que fazem uso de cada uma.

Enquanto todas as empresas relacionadas na tabela oferecem produtos baseados na tecnologia COM/DCOM, apenas uma parcela oferece soluções baseadas na arquitetura OPC, como é o caso dos pacotes Wonderware InTrack (da multinacional Invensys) e Xfactory (da USData). Isso se deve, entre outros aspectos, ao fato da

tecnologia OPC estar mais difundida no nível de instrumentação e SCADA das plantas do que especificamente no nível dos MES.

A OPC Foundation conta atualmente com um total de 318 empresas filiadas. Das 28 empresas relacionadas, 11 são filiadas à OPC Foundation, sendo que em alguns casos mais de uma unidade de negócio de uma mesma empresa pode estar filiada, como é o caso da Siemens, que pode ser encontrada na relação de membros da OPC Foundation sob quatro diferentes nomes (Siemens, Siemens AG, Siemens Automation & Drives e Siemens Building Technologies Inc.).

Uma particularidade da relação de membros da OPC Foundation é que nela podem ser encontradas duas empresas nacionais: a Smar, do interior de São Paulo, fabricante e exportadora de equipamentos e software para automação industrial e a Elipse Software, desenvolvedora de sistemas SCADA e soluções para interface Homem-Máquina.

Cabe observar que entre as empresas filiadas à OMG pode-se encontrar também uma empresa nacional, a Visionnaire, voltada ao desenvolvimento de software, mas não ligada diretamente ao ramo da automação industrial.

Dentre as 28 empresas levantadas, apenas 6 são de algum modo filiadas à OMG, embora sejam 6 multinacionais bastante significativas: ABB (Asea Brown Bovery), GE (General Electric), Siemens, Honeywell, Rockwell e IBM. Entretanto, em alguns desses casos, as unidades de negócio associadas à OMG não são as unidades voltadas para a automação industrial relacionadas na tabela, como é o caso da Rockwell, cuja unidade filiada à OMG é a Rockwell Collins (aviação), e não a Rockwell Automation (automação).

Ou seja, mesmo no caso dessas 6 empresas filiadas à OMG, existem produtos MES fazendo uso da tecnologia OPC, como por exemplo os produtos Rockwell Automation e Siemens ORSI.

Todavia, o uso da tecnologia CORBA para implementação de produtos MES não é restrito às 6 multinacionais filiadas à OMG. Também empresas de empresas de menor

porte, como é o caso da Brooks Automation e da Teradyne, possuem produtos de prateleira que fazem uso da arquitetura CORBA.

Entretanto, a arquitetura CORBA ainda é relativamente pouco utilizada na automação de processos e na comunicação de dados com os sistemas MES, sendo que esse mercado tem sido efetivamente dominado de maneira desproporcional pelas arquiteturas DCOM e OPC [KOSHIKEN, 2000] e [MORLEY, 1998]. Para sistemas baseados em Windows NT, a tecnologia OPC é o padrão dominante, conforme observado pela própria OMG em [OMG, 2001a].

Um ponto que demonstra a predominância das ferramentas para automação industrial baseadas na tecnologia OPC é o fato de já existirem atualmente disponíveis interfaces prontas para a comunicação bidirecional de dados entre os sistemas MES e os sistemas ERP, através da tecnologia OPC. Um exemplo é a ferramenta SAP ODA (*SAP OPC Data Access*), que viabiliza a troca de dados entre o SAP R/3 (mySAP.com) e qualquer servidor compatível com o padrão OPC. Na verdade, a ferramenta SAP DOA é um cliente OPC que possibilita que qualquer dado de um sistema SAP possa ser integrado com os dados do chão-de-fábrica. Cabe ainda dizer que a própria SAP também é uma empresa membro da OPC Foundation.

Claramente o domínio da arquitetura DCOM, e consequentemente da tecnologia OPC, está baseado na predominância dos sistemas operacionais Microsoft no mercado. Já em 1998 haviam mais de 150 milhões de computadores rodando produtos baseados na tecnologia COM [LANGE, 1998]. Como em qualquer outro tipo de disputa por mercado, o momento (ou *timing*) é um fator decisivo e nisso a arquitetura Microsoft COM/DCOM é claramente superior à arquitetura CORBA [MORLEY, 1998].

2.7 UML

O modelo de arquitetura proposto neste texto para o desenvolvimento de sistemas MES foi elaborado segundo o padrão definido pela linguagem UML (*Unified Modeling Language)*. Desse modo, será feito nessa seção um resumo dos conceitos principais envolvidos no desenvolvimento de projetos segundo o padrão UML, a fim de que o modelo proposto possa ser corretamente interpretado.

2.7.1 Breve histórico da UML

A UML consiste em uma linguagem desenvolvida especificamente para a análise e projeto de sistemas computacionais orientados a objetos. Fruto do trabalho conjunto de três grandes pesquisadores da orientação a objetos: Grady Booch, James Rumbaugh e Ivar Jacobson, a UML teve sua primeira versão finalizada em 1997, quando foi então submetida à OMG em resposta a uma solicitação de propostas para uma linguagem padronizada de modelação [SCHMULLER, 1999].

Conforme diversas organizações foram percebendo que a UML serviria a seus objetivos estratégicos, o consórcio responsável pelo suporte à UML foi se desenvolvendo, e passando a abrigar empresas como DEC, HP, Microsoft e Oracle. A OMG assumiu a responsabilidade pela manutenção da UML a partir de 1997, e publicou duas revisões em 1998.

A UML é especialmente útil no modelamento de sistemas complexos e distribuídos, em que a escalabilidade e a capacidade de assimilar alterações e atualizações de maneira transparente são essenciais, como é o caso dos sistemas MES. O modelo proposto no presente trabalho foi desenvolvido segundo as técnicas UML, de modo que se julga apropriado apresentar-se uma visão geral desta técnica de modelamento, para que o modelo proposto possa ser adequadamente compreendido.

2.7.2 Componentes da UML

A UML é uma linguagem que dispõe de uma série de elementos formais e gráficos para representar os elementos de um sistema orientado a objetos e regras para definir os relacionamentos entre esses elementos [SCHMULLER, 1999]. O papel da UML é

dispõe de um conjunto de diagramas, em que os diversos elementos do sistema são apresentados e organizados. Os diagramas utilizados no modelamento segundo o padrão UML são os seguintes:

- Diagrama de Casos de Uso
- Diagramas de Classes
- Diagramas de Comportamento:
  - Diagramas de Estados
  - Diagramas de Atividades
  - Diagramas de Interação:
    - Diagramas de Seqüência
    - Diagramas de Colaboração
- Diagramas de Implementação:
  - Diagramas de Componentes
  - Diagramas de instalação (ou *Deployment*)

2.7.2.1 Diagramas de Casos de Uso

Os diagramas de casos de uso são utilizados para a representação gráfica do comportamento do sistema, sendo compostos por dois elementos principais: os atores (que representam os usuários ou quaisquer elementos externos que precisem interagir com o sistema) e os casos de uso (que indicam as situações de utilização, representando as possíveis interações entre o sistema e seus usuários).

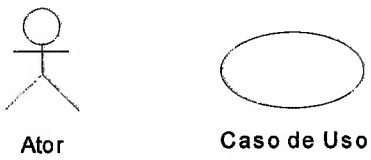

Figura 30 - Elementos gráficos dos casos de uso

Na representação de um diagrama de caso de uso, um ator inicia um caso de uso, e um ator recebe uma saída do caso de uso (podendo este ator ser o mesmo que iniciou a ação ou não). A representação é bastante direta (figura 30): uma elipse representa o caso de uso, e uma figura simples representa o ator, sendo que o ator que inicia a ação está sempre ao lado esquerdo do caso de uso, e o ator que recebe o resultado ao lado esquerdo. As associações entre os atores e os casos de uso são realizadas através de linhas que os conectam (ver figura 31).

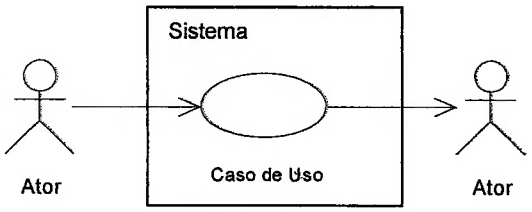

Figura 31 - Modelo de caso de uso

A importância dos casos de uso reside em sua capacidade de representar a utilização do sistema do ponto de vista dos usuários, que podem ser envolvidos no projeto e estimulados a colaborar em sua definição desde os primeiros esboços. Além disso, os casos de uso possibilitam a visualização exata das fronteiras do sistema com o mundo exterior (representadas pela caixa em que estão colocados todos os casos de uso do modelo).

Um exemplo de sistema modelado por um diagrama de casos de uso está apresentado na figura 32, que representa, de forma simplista, uma máquina automática de venda de refrigerantes.

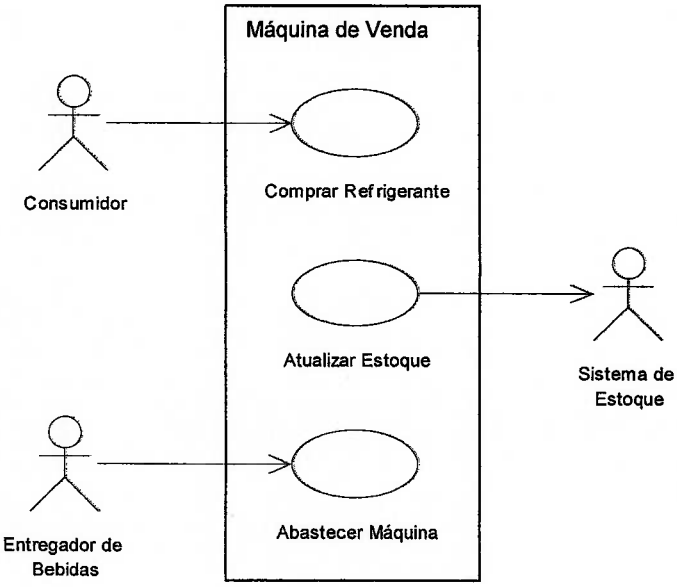

Figura 32 - Exemplo de diagrama de Casos de Uso

2.7.2.2 Diagramas de Classe

Na UML utiliza-se o Diagrama de Classes para representar cada uma das classes que compõe o sistema, juntamente com seus diversos relacionamentos. A função

2.7.2.2 Diagramas de Classe

Na UML utiliza-se o Diagrama de Classes para representar cada uma das classes que compõe o sistema, juntamente com seus diversos relacionamentos. A função principal do Diagrama de Classes é estabelecer a arquitetura que orientará o desenvolvimento do sistema.

As classes são representadas por retângulos, sendo que o seu nome, por convenção, é dado por uma palavra iniciada com uma letra maiúscula que aparece próxima ao lado superior do retângulo. Quando o nome de uma classe for composto, unem-se as palavras, utilizando-se letras maiúsculas para a primeira letra de cada palavra.

Logo abaixo do nome da classe, subdivide-se o retângulo. Nessa nova divisão colocam-se todos os atributos pertinentes a esta classe, iniciando-se seus nomes com letras minúsculas por convenção. Caso os atributos tenham valores predefinidos, esses podem ser indicados ao lado do nome de cada atributo.

Finalmente, cria-se uma última subdivisão no retângulo indicativo da classe para indicarem-se as suas operações, de modo que todas as interfaces possíveis para o trabalho com a classe estejam representadas de maneira simples e objetiva, como pode ser visto na figura 33.

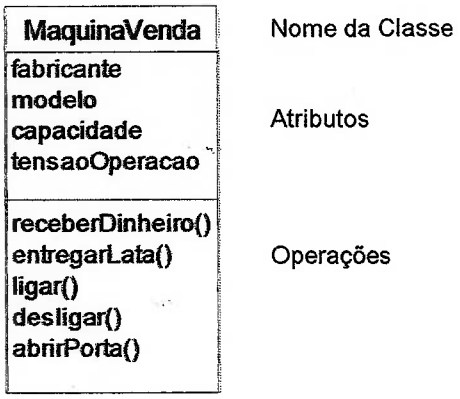

Figura 33 - Representação de uma classe

Tendo-se criado as diversas representações de classes para os elementos do sistema, passa-se agora à definição e representação de seus inter-relacionamentos.

A primeira relação possível entre classes é a relação do tipo *Associativa*, em que as classes são ligadas conceitualmente e é representada por uma linha de ligação entre duas representações de classes. Em uma relação associativa, cada uma das classes desempenha um determinado papel, que pode ser indicado no diagrama. Como exemplo, pode-se observar o relacionamento "utiliza" indicado na figura 34.

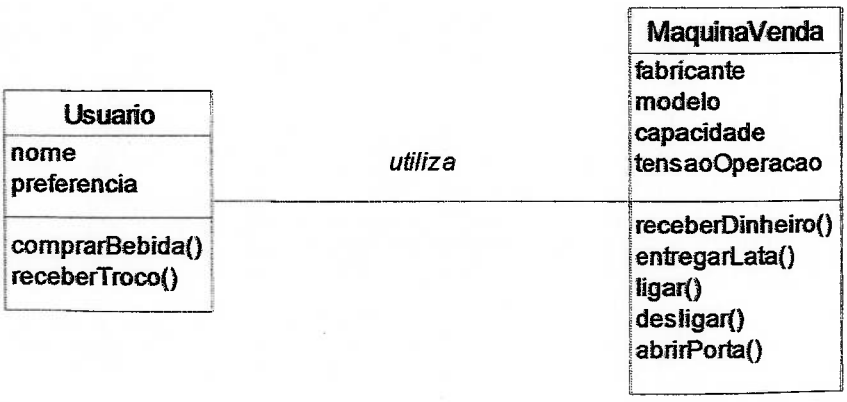

Figura 34 - Exemplo de uma relação associativa

As associações são bidirecionais por natureza. Embora o nome de uma relação de associação costume ser lido em uma determinada direção, ela pode sempre ser percorrida em sentido contrário.

Pode-se também estabelecer relações associativas entre mais de duas classes. Por exemplo, no caso apresentado na figura 19, pode-se também estabelecer uma relação associativa do tipo "utiliza" entre uma eventual classe "Entregador" com a "MaquinaVenda", e não só com a classe "Usuário".

Nas relações associativas pode-se também expressar a noção de multiplicidade, ou seja, definir-se quantas instâncias de uma classe relacionam-se a uma única instância da classe associada, restringindo-se assim a quantidade de objetos relacionados. Por exemplo, podem-se definir relações em que a quantidade de objetos instanciados seja "1" (exatamente um), "1+" (um ou mais), "3-5" (três a cinco, inclusive), "2,4,18" (dois, quatro ou dezoito), e assim por diante. Essas relações são expressas no diagrama através de sua indicação sobre a linha de associação, próxima à classe a que se referem.

As associações também podem ser reflexivas, ou seja, podem relacionar uma dada classe com ela mesma. Essa situação pode ocorrer no caso de uma classe representar diversos papéis diferentes. Por exemplo, uma classe "OcupanteDeCarro" pode representar o papel de "Motorista" ou de "Passageiro", sendo ambos relacionados através de uma associação do tipo "Transporta".

Um outro tipo importante de relacionamento entre classes é o relacionamento do tipo "parte-todo", conhecido como um relacionamento de *Agregação* (figura 35). Nas relações de agregação, as classes que representam componentes ou partes de algo são associados a uma classe que representa o todo, ou o conjunto completo.

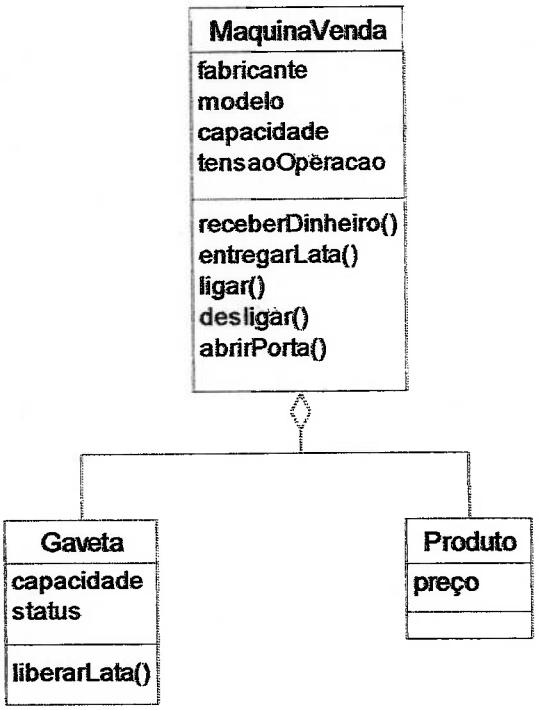

Figura 35 - Exemplo de relação de agregação

Existe também, nos diagramas de classe, a necessidade de representação dos conceitos de herança e generalização. Por *Generalização*, entenda-se o relacionamento entre uma determinada classe e suas versões mais refinadas. Nessa situação, a classe que estiver passando pelo processo de refinamento é chamada de superclasse, e a sua versão refinada é chamada de subclasse. Por exemplo, uma classe do tipo "Estudante" poderia ser criada como uma subclasse da "Usuário", que por sua vez pode ser considerada uma subclasse de uma superclasse "SerHumano".

Nesse processo de refinamento, as subclasses herdam propriedades (métodos e atributos) das suas respectivas superclasses. Por exemplo, a classe "Estudante" herdaria de "Usuário" os atributos "nome" e "preferência", e poderia agregar novos atributos, como "escola", por exemplo. As relações de generalização são representadas através de um triângulo interligando a superclasse aos seus refinamentos (figura 36).

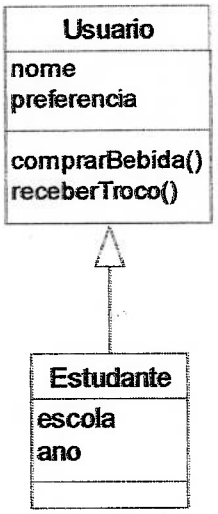

Figura 36 - Exemplo de relação de generalização

2.7.2.3 Diagramas de Estados

Dá-se o nome de "estado" ao conjunto dos valores de atributos e ligações mantidas por um objeto em um determinado instante. Durante a execução de um sistema, os diversos objetos que o compõem excitam-se uns aos outros, ocasionando alterações de estados e gerando, nesse processo, os chamados "eventos", que nada mais são do que estímulos individuais de um objeto sobre outro.

A fim de representar-se o padrão de eventos, estados e transições de estados para uma determinada classe utiliza-se os chamados *Diagramas de Estados*. Esses diagramas consistem de redes de estados e eventos, da mesma maneira que um diagrama de classes é uma rede de classes e relacionamentos. Nas redes (ou grafos), representativos de diagramas de estados, cada um dos nós é um estado e os arcos direcionados são transições rotuladas com nomes de eventos. Cada diagrama de estados descreve o comportamento de uma classe (ver figura 37).

Devido à sua natureza dinâmica e ao seu objetivo básico (representar os estados possíveis de um objeto), os diagramas de estados podem tornar-se bastante complexos com facilidade. Entretanto, eles são úteis para que os analistas possam compreender e simular o comportamento das diversas classes que compõem o sistema, o que não é possível apenas com os diagramas de classes, devido à sua natureza estática [RUMBAUGH, 1994].

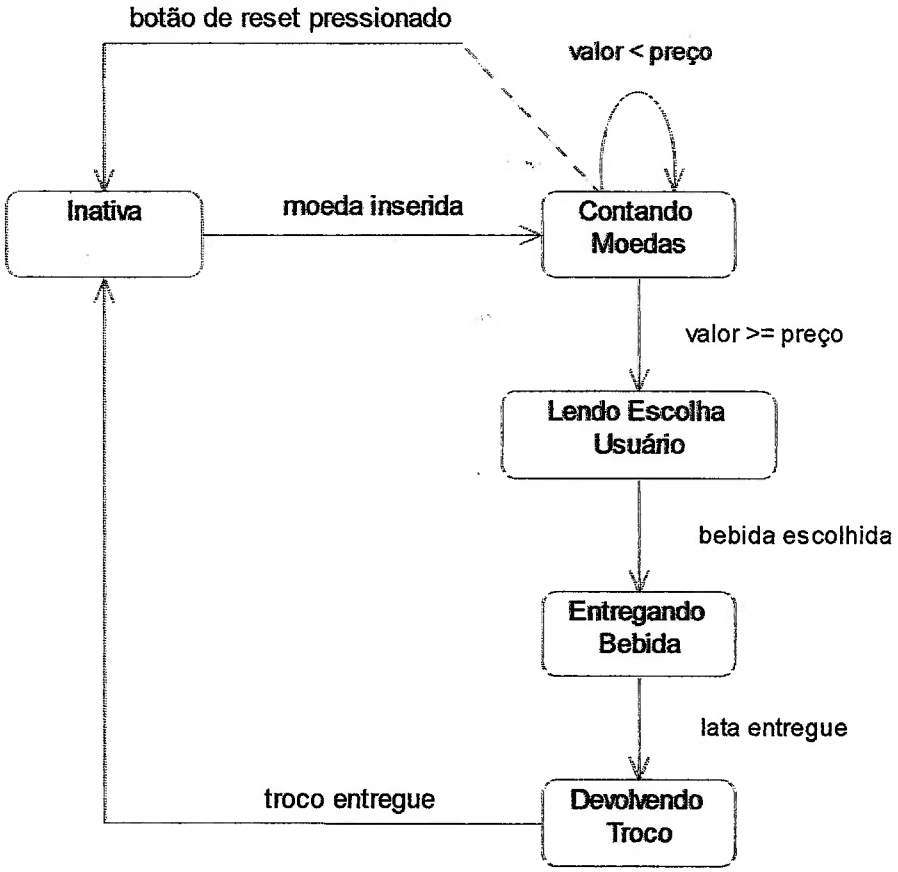

Figura 37 - Exemplo de Diagrama de Estados

2.7.2.4 Diagramas de Seqüência

Os diagramas de seqüência possuem a função de demonstrar a interação dinâmica entre objetos ao longo do tempo. Basicamente, os diagramas de seqüência compõem-se de objetos (representados por retângulos com seu nome sublinhado), mensagens (representadas por setas horizontais sólidas) e tempo (representado por linhas verticais).

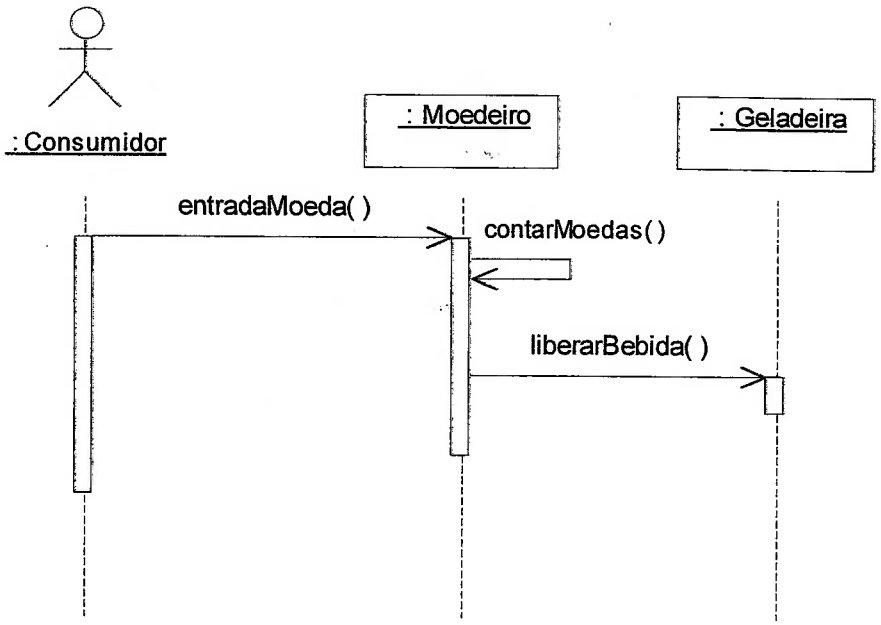

Figura 38 - Exemplo de diagrama de seqüência

No diagrama de seqüência, a linha pontilhada posicionada abaixo de cada objeto é chamada linha da vida daquele objeto. Ao longo das linhas da vida dos objetos, localizam-se um ou mais retângulos denominados ativação, e que representam a execução de uma operação por parte do objeto, sendo o seu comprimento indicativo da duração desta operação (ver figura 38).

As mensagens passadas entre os objetos são representadas por setas horizontais e podem ser de três tipos básicos: simples, síncronas e assíncronas. As mensagens simples representam as transferências de controle de execução do sistema de um objeto para outro; as mensagens síncronas representam situações em que um objeto precisa esperar pela resposta do objeto seguinte para prosseguir; e as mensagens assíncronas representam situações em que o objeto pode continuar a trabalhar independentemente de haver recebido ou não uma resposta. É importante observar-se que um objeto pode também se auto-ativar, passando uma mensagem para si próprio, ou seja, chamando um de seus próprios métodos.

Além dos diagramas apresentados, existem outros diagramas relevantes para o modelamento de sistemas conforme o padrão UML. Entretanto, para os objetivos do presente trabalho, os diagramas apresentados são suficientes, ou seja, possibilitam a

descrição do modelo básico proposto, de modo que os demais diagramas não serão abordados nessa revisão bibliográfica.

2.7.2.5 Interfaces

Em projetos orientados a objetos é comum o uso do conceito de interfaces, conforme já explicado anteriormente na seção 2.5.1.3.

De uma maneira geral, as interfaces são uma forma de que o objeto dispõe de apresentar ao mundo exterior as operações que pode realizar, de modo que outros objetos possam solicitar a execução de determinados métodos mesmo não tendo acesso a seus detalhes de implementação.

Em alguns casos, diversas classes podem vir a dispor de operações análogas às mesmas assinaturas, mesmo sem possuir nenhum tipo de relacionamento por herança. O operador *interface* é a ferramenta UML que possibilita que isso seja implementado. Pode-se pensar em uma interface como uma classe sem atributos, apenas com métodos, ou seja, como o conjunto de operações que uma classe apresenta a outra classe. O relacionamento entre um componente de software e sua interface é do tipo "realiza".

As interfaces possuem um papel de importância na reutilização de componentes de software. Sempre é possível trocar-se um dos componentes por outro equivalente em um sistema, desde que as interfaces sejam as mesmas, o que facilita a manutenção e o desenvolvimento de sistemas.

A representação de uma interface em UML é similar à de uma classe comum, com a diferença da presença do texto "<< Interface >>" no campo em que se apresenta o nome da classe em questão.

2.8 Design Patterns

No modelo proposto neste trabalho faz-se uso de algumas estruturas abstratas, ou padrões para o desenvolvimento de sistemas, já consagradas e extensivamente usadas por equipes de desenvolvimento, a fim de se aproveitar o conhecimento e experiência acumulados por desenvolvedores ao longo de diversos projetos e diminuir-se, dessa forma, o tempo de desenvolvimento e melhorar-se a qualidade do modelo. O conceito envolvido na criação dessas estruturas e o detalhamento das estruturas escolhidas para serem usadas no protótipo estão apresentados nesta seção do texto.

O projeto de software com capacidade de reutilização não é uma atividade trivial: é necessário definir os objetos pertinentes, agrupá-los nas classes corretas (com a granularidade adequada), definir as interfaces, detalhar as relações de herança e estabelecer os relacionamentos entre cada uma delas.

O projeto de software reutilizável deve contemplar a resolução do problema imediato, mas deve ser generalizado a ponto de poder ser usado na resolução de problemas e requisitos futuros, caracterizando-se por uma atividade que demanda muito trabalho para ser concluída. Apenas em raras situações consegue-se desenvolver componentes de software reutilizáveis na primeira tentativa, sendo necessária a sua aplicação em diversos projetos distintos até que se obtenha uma versão realmente genérica e completamente reutilizável [GAMA, 1995].

Existem alguns tipos de componentes que tendem a aparecer com grande freqüência nos projetos orientados a objetos e que resolvem problemas ou requisitos que tendem a aparecer em uma grande variedade de aplicações. Esses componentes podem colaborar com o projetista de aplicações orientadas a objetos a desenvolver software com sucesso em menos tempo e com maior flexibilidade. A esses componentes dá-se o nome de *design patterns*, ou padrões de projeto, e podem ser encontrados na literatura disponível sobre orientação a objetos e componentes, como em [GAMA, 1995].

Um padrão de projeto abstrai e dá um nome a um conjunto de aspectos chave de uma estrutura de projeto, identificando as classes participantes, seus papéis e sua

distribuição de responsabilidades. Cada padrão foca em um problema particular da orientação a objetos e baseia-se em soluções práticas já experimentadas em diversas aplicações anteriormente.

Existem diversas coletâneas de padrões que estão agrupadas em livros, em que pode-se identificar diversos padrões úteis ao projeto de sistemas distribuídos para uso em ambientes industriais, como é o caso do padrão Observer, também conhecido como Publish-Subscribe.

2.8.1 O padrão Publish-Subscribe

O motivo pelo qual o padrão de projetos Publish-Subscribe é importante para o presente trabalho é a sua relação direta com os modelos de comunicação de mensagens que serão utilizados no modelo aqui proposto.

O padrão Publish-Subscribe é utilizado quando se deseja manter uma comunicação consistente de dados entre diversas classes sem que seja necessário manter um alto grau de acoplamento entre elas, o que poderia reduzir sua capacidade de reutilização [GAMA, 1995].

Por exemplo, em uma aplicação de chão-de-fábrica, têm-se normalmente terminais de supervisão que podem estar ligados de diversas formas a um mesmo sensor ou controlador. Nesses terminais de supervisão, as informações coletadas dos sensores são apresentadas ao usuário das mais variadas formas, como gráficos, listagens e mudanças de cores, sendo que, para o usuário, o sensor e o terminal comportam-se como se fossem uma única entidade. Entretanto, os diversos componentes que formam uma rede de supervisão e controle em um ambiente de chão-de-fábrica, podem ser de diversos fabricantes diferentes, da mesma maneira que o software que exibe as informações na tela para os operadores do sistema.

Deste modo, embora a coesão entre os diversos componentes do sistema seja um imperativo em uma rede industrial, é desejável que estes componentes mantenham sua independência funcional e seu isolamento operacional, a fim de que possam estar isolados dos problemas que possam vir a ocorrer com outros componentes e que possam ser eventualmente substituídos por componentes de função análoga em uma

situação de necessidade, sem que se precisem realizar modificações no restante do sistema.

Essa afirmação é válida não apenas para o hardware de controle dos processos industriais como também para o software de supervisão. Por exemplo, o componente de software que disponibiliza dados de um sensor aos terminais de supervisão deve ser independente do componente de interface do usuário que é executado no terminal, mas ambos os componentes precisam trocar informações e dados instantaneamente e devem comportar-se como se fosse um só.

Esse comportamento implica que o componente de exibição dos dados seja dependente do componente de envio de dados do sensor e que deva ser notificado de qualquer mudança no estado do sensor. Na verdade, não há qualquer razão para que se limite o número de componentes de exibição a um; pode ser necessário que diversos terminais e aplicações recebam os dados de um mesmo sensor.

Essa necessidade não é uma exclusividade das aplicações industriais. Muitos outros casos de aplicação de software encontram este mesmo problema, como, por exemplo, um software de planilha de cálculo em que se podem desenvolver gráficos com base em uma tabela de dados, ou um software CAD em que as dimensões de alguns sólidos estejam parametrizadas em função de uma tabela de características. Dessa forma, o padrão Publish-Subscribe vem sendo desenvolvido ao longo do tempo, com base na experiência em diversos projetos e hoje presta-se a solucionar esta questão independentemente do domínio da aplicação.

Os dois elementos principais do *pattern* Publish-Subscribe são o publicador e o subscritor. Um publicador pode estar associado a qualquer número de subscritores e todos os subscritores são notificados quando há uma mudança de estado no publicador. Por outro lado, todos os subscritores podem questionar o publicador para sincronizar seus estados.

Uma estrutura típica de implementação da comunicação de dados segundo o *pattern* Publish-Subscribe, e representada segundo o padrão UML, pode ser vista na figura 39.

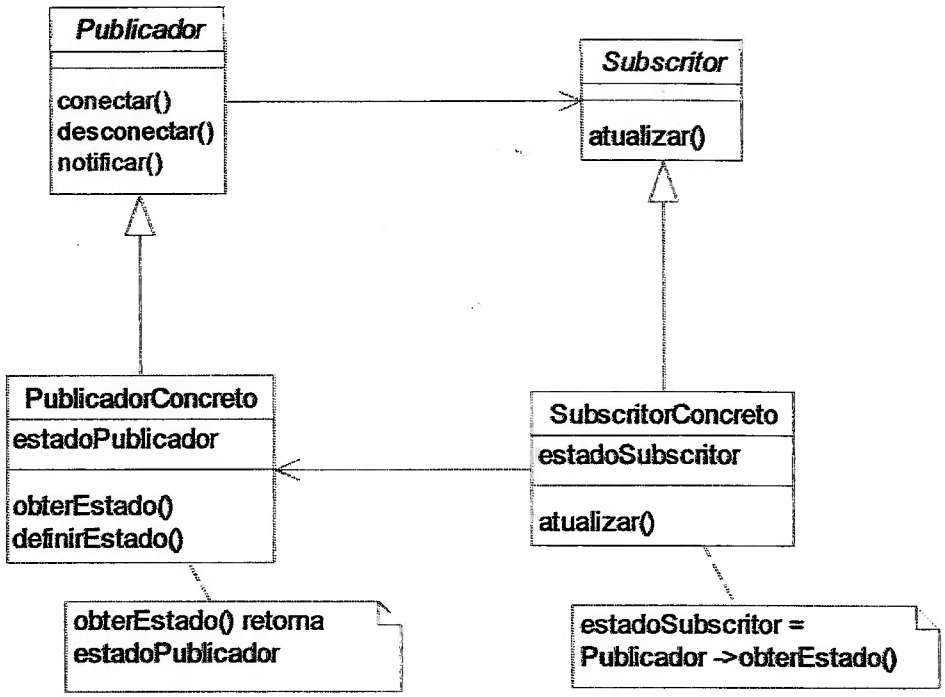

Figura 39 - Estrutura típica da implementação Publish-Subscribe

Nessa estrutura pode-se identificar a presença de quatro classes distintas: Publicador, PublicadorConcreto, Subscritor e SubscritorConcreto. A classe Publicador é responsável pelo controle dos objetos subscritores que se conectam a ela e fornece uma interface para a conexão e desconexão de subscritores. A classe Subscritor, por sua vez, define uma interface de atualização para objetos que devam ser avisados quando ocorrerem alterações nos publicadores (via um mecanismo de *callback*).

Já a classe PublicadorConcreto tem a função de armazenar os estados de interesse para os objetos subscritores concretos criados, e de enviar uma notificação para estes observadores quando seus estados sofrerem modificação.

A classe SubscritorConcreto, por sua vez, mantém uma referência para os objetos concretos publicadores e possui estados consistentes com os dos publicadores. Além disso, a classe SubscritorConcreto também implementa a interface de atualização definida pela classe Subscritor, e que é usada para a atualização dos estados dos publicadores.

O funcionamento da estrutura dá-se da seguinte forma: inicialmente, um subscritor que tenha interesse no estado de um publicador conecta-se a ele através do método

"conectar"; assim que o estado do objeto publicador concreto for alterado, o publicador em questão notifica aos subscritores interessados no estado em questão.

Entretanto, o subscritor pode assumir um papel ativo na comunicação dos eventos. Por exemplo, quando o subscritor deseja verificar o estado de um publicador, ele aciona o método "obterEstado", que retorna, a qualquer instante, o estado corrente do publicador. O subscritor pode utilizar ainda o método "definirEstado" para alterar o estado interno do publicador sendo que, neste caso, o próprio subscritor que solicitou a alteração deve aguardar por uma notificação do publicador para modificar seu estado interno, de modo que se tenha certeza de que a operação foi concluída com sucesso e de que os estados permanecerão consistentes.

Uma particularidade deste *pattern* está no uso das interfaces abstratas para que se realizem as conexões entre publicadores e subscritores. Essa técnica possibilita que tudo que um publicador precise saber seja apenas uma lista dos seus subscritores, cada qual em conformidade com a interface simples definida pela classe abstrata Subscritor, sendo que o publicador não possui referências para nenhuma das classes concretas de nenhum subscritor, ou seja, o acoplamento entre publicador e subscritor é mínimo.

Outra característica do esquema de comunicação Publish-Subscribe está em sua capacidade de transmitir informações de modo *broadcast*, ou seja, a notificação emitida pelo publicador não precisa ter um destinatário específico, sendo enviada automaticamente para todos os subscritores interessados na mesma, o que dá liberdade para se adicionarem ou removerem subscritores a qualquer momento e simplifica a comunicação de eventos para um grande número de objetos interessados.

A arquitetura CORBA prevê a implementação de uma forma de comunicação no esquema Publish-Subscribe através de seus serviços de Notificação e de Eventos. Em ambos os casos, todas as propriedades aqui assinaladas são respeitadas, embora os nomes das classes difiram dos apresentados nesta seção. Por exemplo, os subscritores são tratados por consumidores (ou *consumers*), e os publicadores são tratados por fornecedores (ou *suppliers*). Já a comunicação automática do publicador para o

subscritor é definida como uma transferência de dados do tipo *push* e a solicitação de atualização por parte do subscritor é definida como uma comunicação do tipo *pull*.

2.8.2 O padrão *Factory*

Os *Frameworks* utilizam classes abstratas para definir e manter relacionamentos entre objetos, como forma de manterem-se "genéricos". Por exemplo, um *Framework* para aplicações de escritório, que tenha por objetivo definir uma arquitetura genérica, utilizável tanto por um software de processamento de textos quanto por uma planilha de cálculo, precisa manter um grau de abstração suficiente para incorporar as funções comuns a ambos os tipos de software, mas não pode tornar-se específico a ponto de ter sua aplicação impossibilitada no caso de uma aplicação de criação de apresentações que venham a ser desenvolvidas posteriormente.

Ainda neste mesmo caso das aplicações de escritório, o *Framework* poderia definir, por exemplo, a abstração de pelo menos duas classes básicas: uma classe Aplicação e uma classe Documento. A classe Aplicação seria responsável por gerenciar instâncias da classe Documentos (que poderiam ser dos mais variados formatos) e, por criá-las conforme o necessário, através de um método "criarDocumento()".

Ambas as classes são abstratas e possuem classes filhas concretas que herdam seu comportamento e implementam códigos específicos, como AplicaçãoPlanilha, AplicaçãoTexto, DocumentoPlanilha e DocumentoTexto. Entretanto, cada subclasse de Documento (como DocumentoPlaniha), é específica para uma determinada subclasse correspondente de Aplicação e a classe Aplicação não consegue prever qual o tipo específico de Documento que deve gerar quando a criação de um novo Documento é solicitada pelo usuário do sistema.

O objetivo do padrão *Factory* é o de solucionar essa limitação, encapsulando a informação referente ao tipo de documento e removendo esta informação de dentro do *framework*. Isso é feito através da redefinição do método CriarDocumento() na classe filha, ou seja, na classe AplicaçãoPlanilha de nosso exemplo. Uma vez que a classe AplicaçãoPlanilha esteja instanciada, ela pode instanciar seus documentos específicos (DocumentoPlanilha), sem se preocupar com seu tipo.

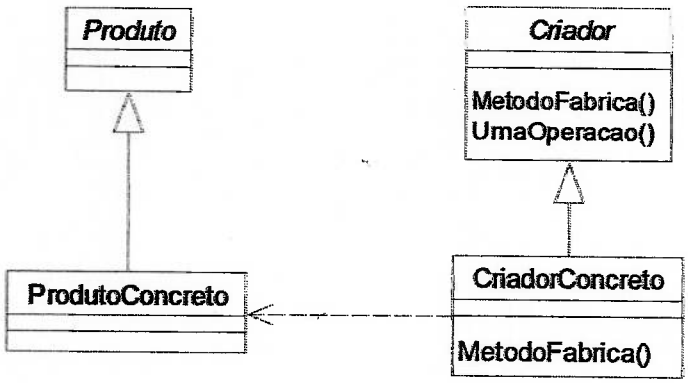

Figura 40 - Estrutura do *pattern Factory*

Na estrutura do padrão *Factory* (ver figura 40), a classe Produto define a interface dos objetos que o método fábrica cria, enquanto a classe ProdutoConcreto define a implementação dos diversos tipos de produtos que podem ser criados.

A classe Criador tem a função de declarar o método fábrica, que retorna, por sua vez, um objeto instanciado do tipo Produto. O Criador também pode definir uma implementação padrão do método fábrica que retorne um objeto Produto padrão.

Já a classe CriadorConcreto sobrepõe o método fábrica da classe Criador para retornar uma instância de ProdutoConcreto compatível.

2.8.3 O padrão *Decorator*

Um aspecto que deve ser observado pelo modelo sugerido no presente trabalho diz respeito à independência de plataforma, sistema operacional ou linguagem que possa vir a ser utilizada para sua implementação. Desta forma, surge a obrigatoriedade de levar-se em consideração algumas particularidades e limitações das linguagens de programação orientadas a objetos mais difundidas.

No caso da linguagem Java, há uma limitação específica que interfere diretamente no modelo a ser proposto, que é a impossibilidade da ocorrência de situações de herança múltipla entre as classes de um sistema [LEMAY, 1999].

O padrão *Decorator* tem a função de adicionar funcionalidades a um determinado objeto de maneira dinâmica, representando uma alternativa à herança [GAMA, 1995].

Por exemplo, em uma determinada aplicação de controle no ambiente de chão-de-fábrica, pode ser necessária a geração de diversos tipos de interfaces gráficas com os usuários. A fim de padronizar as diversas interfaces, o responsável pelo desenvolvimento do sistema pode desejar desenvolver uma tela padrão e adicionar ou remover funcionalidades a ela conforme o necessário.

Uma das maneiras de se adicionarem funcionalidades à interface é através da herança. Em uma situação em que seja interessante adicionar-se um controle do tipo barra de rolagem horizontal à interface, a classe "tela" pode herdar a barra de uma outra classe, de forma que todas as suas instâncias recebam a adição de um controle do tipo barra de rolagem. Entretanto, adotando-se esta solução, torna-se inviável controlar como e quando adicionar a barra à tela.

O padrão *Decorator* fornece uma forma mais flexível de se adicionarem funcionalidades a uma classe. A solução proposta por ele é a de se encapsular o componente original em um outro objeto (o decorador), que adicione a funcionalidade desejada. A interface do decorador deve estar em conformidade com a interface do objeto decorado, de modo que sua presença seja sempre transparente para os demais componentes do sistema, e essa transparência possibilita que se aninhem decoradores sucessivamente, adicionando-se uma quantidade ilimitada de novas funcionalidades ao objeto original.

Uma estrutura típica de implementação do *pattern Decorator* representada segundo o padrão UML, pode ser vista na figura 41 a seguir.

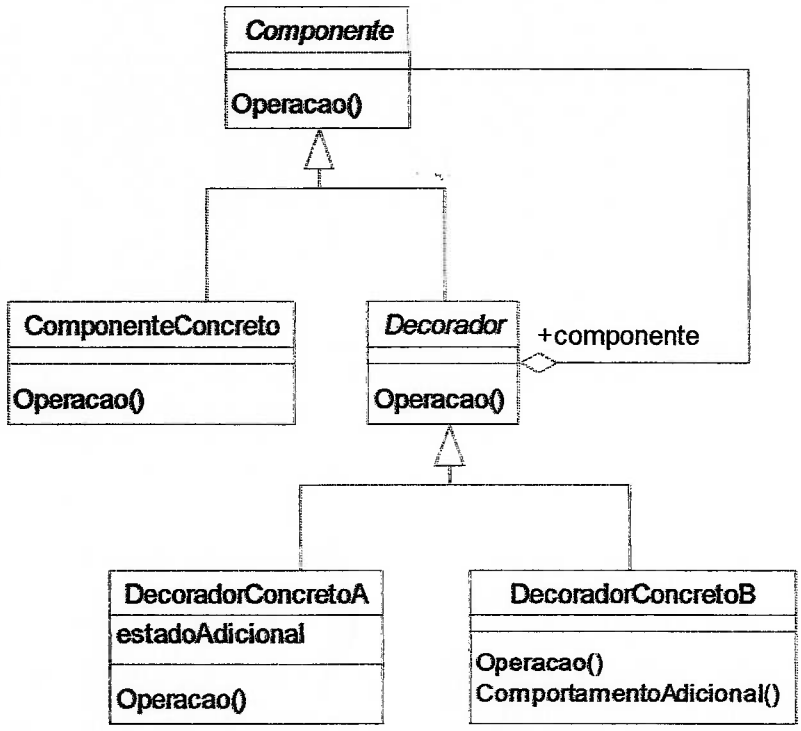

Figura 41 - Estrutura do *pattern Decorator*

Dentro desta estrutura, a classe Componente define a interface para os objetos que poderão receber funcionalidades adicionadas de forma dinâmica, enquanto a classe ComponenteConcreto define o objeto ao qual serão adicionadas as funcionalidades.

A classe Decorador mantém uma referência à classe Componente e define uma interface que esteja em conformidade com a interface de Componente. Já as classes DecoradorConcreto são as responsáveis por implementar efetivamente a adição das novas funcionalidades ao componente.

Durante a operação do sistema, quando o Decorador recebe uma solicitação de execução de um método, ele encaminha essa solicitação para o método correspondente do Componente, embora seja possível também ao Decorador executar operações adicionais antes ou após encaminhar o pedido de execução para o Componente.

3 Apresentação do modelo proposto e protótipo implementado

A fim de se avaliar a aplicabilidade da arquitetura CORBA a aplicações MES, desenvolveu-se um modelo básico, a ser usado como ponto de partida para eventuais implementações complexas que podem vir a abranger todas os diversos ramos de aplicação dos MES.

3.1 Foco do modelo proposto

A gama de funções dos MES, apresentada e detalhada no capítulo 2, engloba os seguintes domínios de aplicação [MESA, 1997b]:

- Alocação de Recursos e Status
- Agendamento de operações detalhado
- Alocação de unidades de produção
- Controle de documentação
- Coleta de dados
- Gestão de mão-de-obra
- Gestão da qualidade
- Gerenciamento de processos
- Gestão de atividades de manutenção
- Rastreabilidade e genealogia de produtos
- Análise de performance

Estas funções combinadas fornecem os subsídios necessários para que um sistema MES realmente atue como o elemento de ligação entre o ambiente de planejamento e o ambiente de execução, e forneça ganhos reais tanto para o escritório quanto para o chão-de-fábrica. Entretanto, uma vez que o objetivo do presente trabalho é o de avaliar a aplicabilidade da arquitetura CORBA aos MES, e o de propor uma

arquitetura básica funcional que comprove essa aplicabilidade, não se julga necessária a implementação de uma solução definitiva que abranja todas as funções supra citadas.

O modelo desenvolvido e apresentado neste trabalho tem um espectro de funções restrito, e resumido às seguintes atividades:

- Alocação de Recursos e Status
- Coleta de dados

A função de Alocação de Recursos e Status compreende o gerenciamento de recursos como máquinas, ferramentas, materiais e outras entidades que devem estar disponíveis para que a produção transcorra normalmente. Essa função proporciona a visão da alocação dos recursos e assegura que os equipamentos estejam prontos para o trabalho e forneçam informações de status em tempo real.

Já a atividade de Coleta de Dados implementa a interface de dados entre os equipamentos de produção e os registros e relatórios associados a cada unidade produtiva. Os dados podem ser coletados do chão-de-fábrica manual ou automaticamente de um equipamento, proporcionando, no segundo caso, informações atualizadas de forma instantânea.

3.2 Diagrama de classes proposto

Conforme o já afirmado ao longo desse texto, para a correta compreensão do modelo proposto para a implementação de um sistema MES, optou-se pela utilização da metodologia UML. A visão geral da arquitetura proposta pode ser obtida através da figura 42 abaixo, em que se apresenta o diagrama de classes simplificado do sistema proposto.

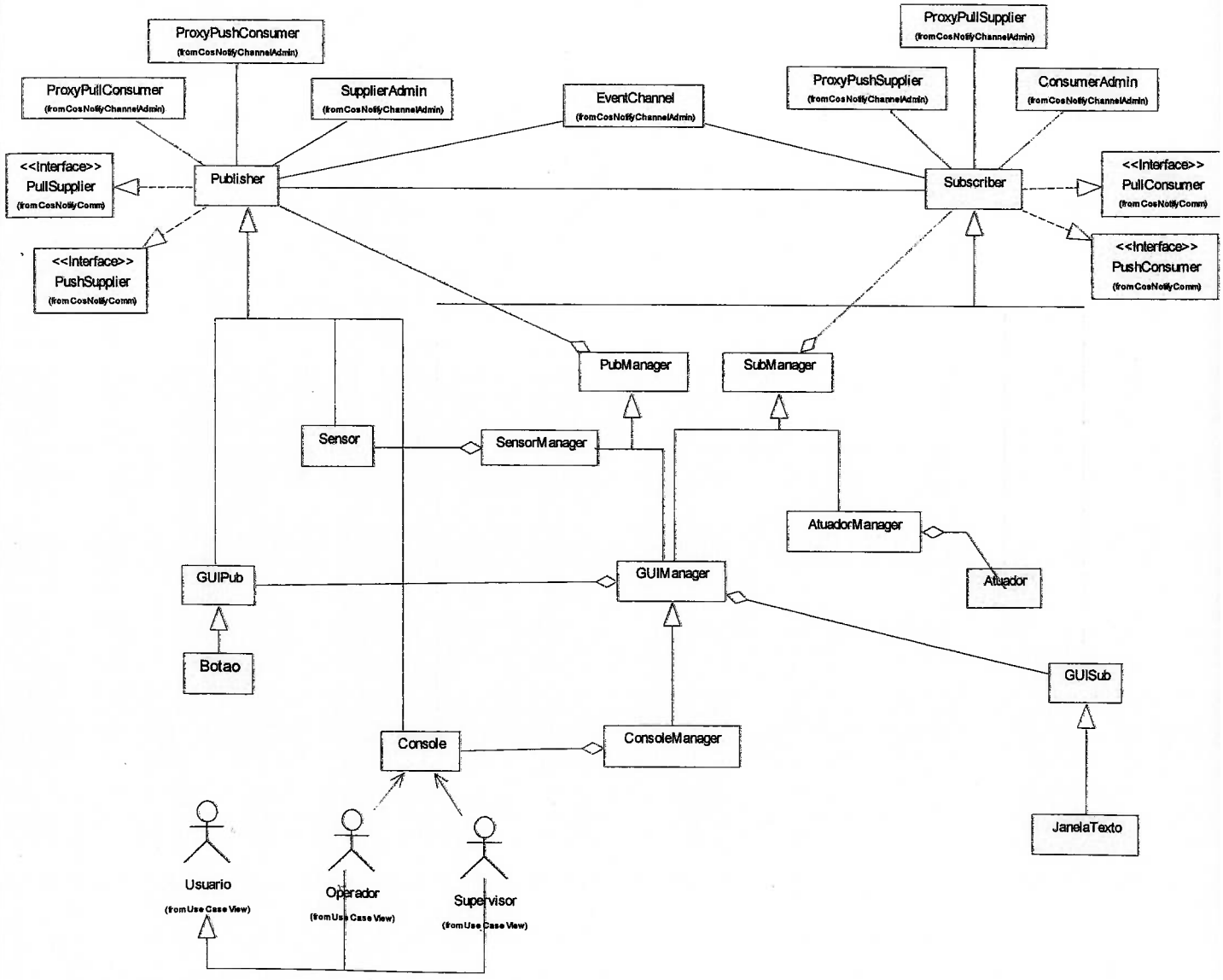

Figura 42 - Diagrama de classes proposto

No diagrama apresentado na figura 42, foram excluídas as descrições de métodos e atributos, a fim de melhorar sua clareza. Entretanto, segue uma apresentação detalhada de cada uma das classes introduzidas no diagrama, com a sua respectiva funcionalidade.

3.2.1 Classe Publisher

A classe Publisher (figura 43) implementa as funcionalidades de uma classe tipo Publicador, ou seja, gerencia a conexão de Subscritores a um objeto com capacidade de publicação de informações. A classe Publisher, juntamente com a classe Subscriber, forma o núcleo do modelo proposto, sendo as classes que possuem controle sobre todos os elementos presentes no sistema em um dado instante.

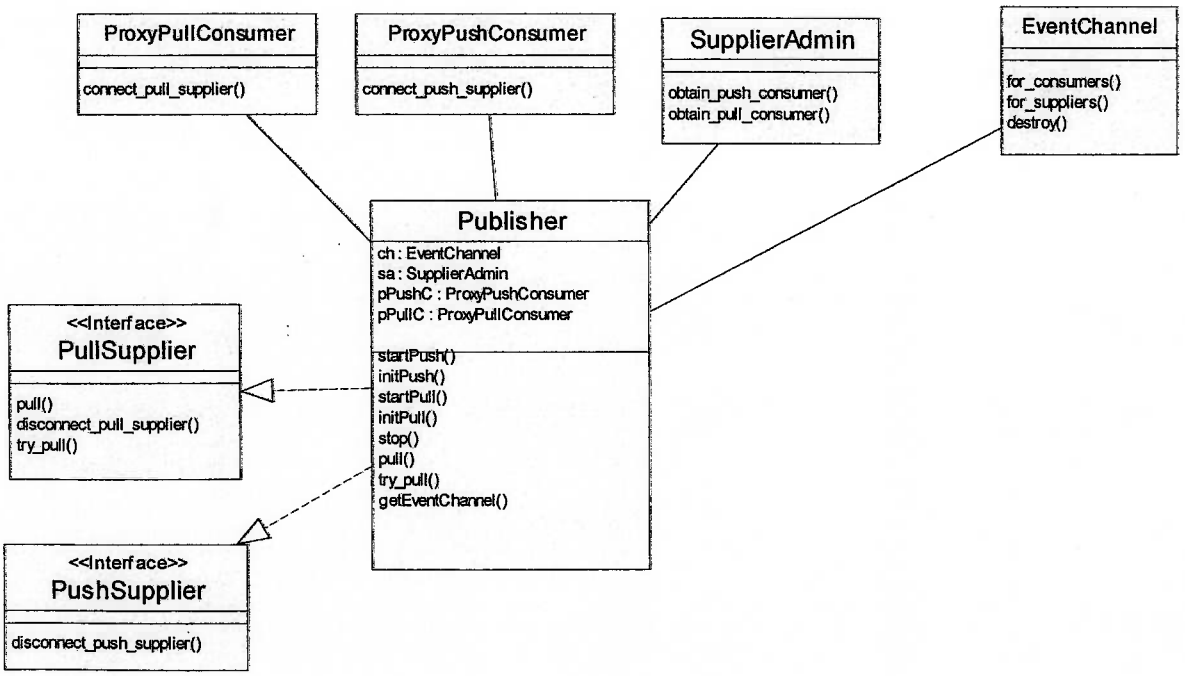

Figura 43 - A classe Publisher

Os métodos da classe Publisher implementam as seguintes funções básicas para o estabelecimento de comunicação através do serviço de notificação:

- Criação do canal de eventos;
- Criação de um proxy consumidor, para o envio dos eventos;
- Funções para o envio dos eventos pelo objeto que os gera.

Como pode ser notado na figura acima, a classe Publisher faz uso de diversas classes definidas e implementadas pelo próprio serviço de notificação. Essas classes, que são parte dos pacotes CosEventComm e CosEventChannelAdmin, possuem os métodos necessários para a criação dos elementos que formam a estrutura de comunicação através do serviço de notificação, ou seja, possuem os construtores para o canal de eventos, para os proxies e para os admins que se façam necessários para a transmissão de eventos e para a administração do canal.

Na verdade, a classe Publisher não chega a ser instanciada no software. Sua presença no modelo tem a finalidade de possibilitar aos elementos instanciáveis (como os sensores, por exemplo), que herdem os métodos definidos no serviço de notificação e os métodos necessários adicionais criados para auxiliar a comunicação dos eventos. Para facilitar a organização do modelo, bem como sua implementação, julgou-se conveniente que todos esses métodos fossem encapsulados nas classes Publisher e Subscriber.

Além dos métodos já definidos pelo serviço de notificação, alguns métodos adicionais foram inseridos na classe Publisher, a fim de tornar mais simples o estabelecimento da comunicação. São eles:

- Método getEventChannel – A função deste método é a de estabelecer um canal de eventos para a transmissão de dados por parte do objeto que o executa (por exemplo, um sensor). Este método aciona a fábrica de canais definida pelo serviço de notificação e cria, com sua ajuda, um canal para a transmissão dos eventos gerados pelo objeto em questão. A referência (IOR) para o canal criado é armazenada, então, em um arquivo qualquer que possa ser lido por um subscritor interessado, que saiba onde procurá-lo. Pode-se ainda armazenar a IOR com um nome adequado na listagem do serviço de nomes, de modo que possa ser obtida por um objeto subscritor interessado localizado em virtualmente qualquer outra máquina.
- Métodos initPush e initPull – são os métodos que inicializam os elementos necessários à comunicação dos dados, como, por exemplo, os proxies consumidores. Como os proxies são diferentes para os esquemas de comunicação

Push e Pull, optou-se por criar métodos específicos de inicialização para cada esquema.

- Métodos startPush e startPull – iniciam a comunicação dos dados através dos proxies criados pelos métodos initPush e initPull.
- Métodos pull e try_Pull – são os métodos padrão (cujos nomes são definidos pelo serviços de notificação), que podem ser chamados por um consumidor (através de um proxy fornecedor), quando esse consumidor desejar obter os dados de um fornecedor. Esses métodos são usados no esquema de comunicação Pull, e só precisam ser implementados em situações em que esse esquema seja usado.
- Método stop – interrompe a comunicação de dados do objeto publicador através do canal.
- Métodos connect e disconnect – são métodos padrão, definidos pelo serviço de notificação, e que realizam a conexão e desconexão dos proxies nos diversos esquemas de comunicação.

3.2.2 Classe Subscriber

A classe Subscriber possui uma função análoga à da Publisher, embora aplicada aos objetos subscritores. Da mesma forma que a classe Publisher, a Subscriber não é efetivamente instanciada no sistema, sendo sua função no modelo a de fornecer aos elementos subscritores o acesso aos métodos definidos pelo serviço de notificação.

Os métodos da classe Subscriber possuem, basicamente, as seguintes funções:

- Criação de um canal de eventos, ou conexão a um canal de eventos em que estejam publicados os dados de interesse ao objeto subscritor;
- Criação dos proxies fornecedores;
- Funções de comunicação dos dados, para o recebimento dos eventos dos elementos publicadores (esquema Push), ou para a solicitação dos dados destes elementos (esquema Pull).

O relacionamento da classe Subscriber com as diversas classes definidas pelo serviço de notificação pode ser verificado na figura 44.

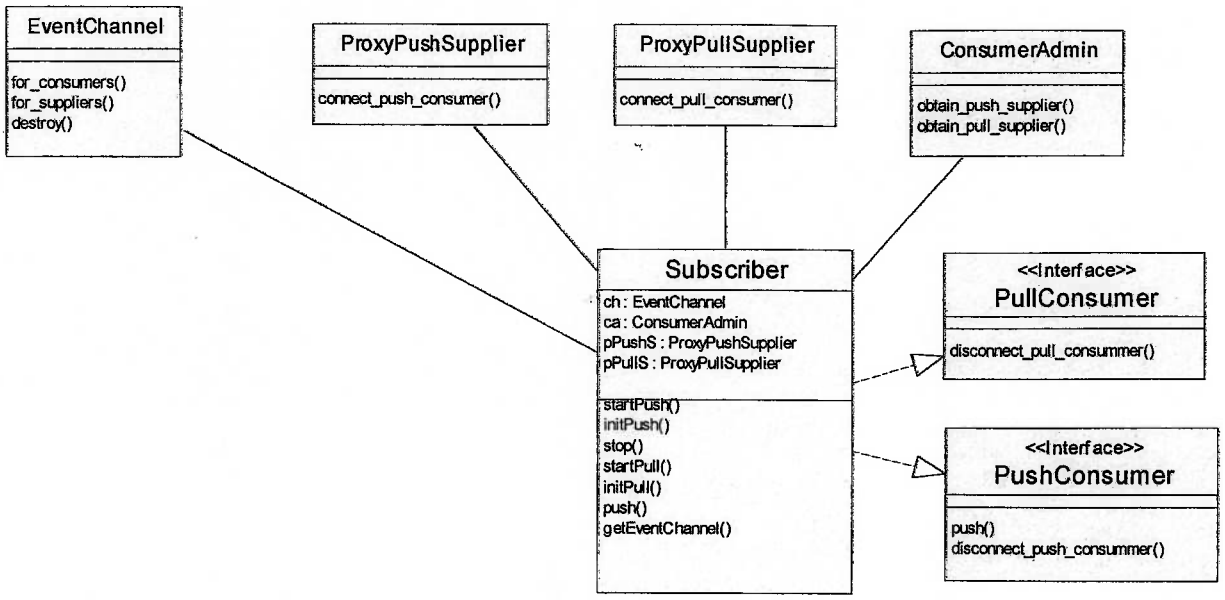

Figura 44 - A classe Subscriber

Do mesmo modo que a classe Publisher, a classe Subscriber utiliza diversas funções já definidas pelo serviço de notificação e que se localizam nos pacotes CosEventChannelAdmin e CosEventComm. Essas funções são implementações de diversos elementos básicos do serviço de notificação, como os canais de eventos e os proxies.

Entretanto, a fim de se facilitarem o estabelecimento dos canais e a conexão dos elementos subscritores aos proxies, foram inseridas na classe Subscriber diversas funções adicionais que prestam-se a simplificar o processo de inicialização do serviço de notificação e de transmissão dos dados. São elas:

- Método getEventChannel – verifica a existência de um canal de dados já estabelecido pelo elemento publicador de interesse sendo que, em caso afirmativo, obtém sua referência. O canal pode ser procurado através da leitura de um arquivo de referência que contenha sua IOR, ou mesmo através de uma lista definida pelo serviço de nomes. Caso não haja um canal já criado, esse método cria seu próprio canal e publica sua IOR.
- Métodos initPush e initPull – da mesma forma que os métodos homônimos da classe Publisher, são os métodos que inicializam os elementos de suporte à

comunicação, como os proxies publicadores. Criaram-se métodos específicos de inicialização para os esquemas Push e Pull.

- Métodos startPush e startPull – iniciam a comunicação dos dados através dos proxies criados pelos métodos initPush e initPull.
- Método push – é o método usado pelos fornecedores para o envio de eventos aos consumidores. Na verdade, este método é chamado pelo proxy consumidor (criado pelo fornecedor), e deve obedecer ao nome definido pelo serviço de notificação. Só está aqui apresentado o método push (adequado ao envio de dados do tipo Any), mas o serviço de notificação pressupõe a presença também de outros métodos como o push_structured_event, adequado ao envio de eventos estruturados.
- Método stop – interrompe a comunicação de dados do objeto publicador através do canal.
- Métodos connect e disconnect – são métodos padrão, definidos pelo serviço de notificação, e que realizam a conexão e desconexão dos proxies nos diversos esquemas de comunicação.

Os objetos com caraterísticas de subscritores a serem usados no sistema devem herdar os métodos definidos pela classe Subscriber, da mesma forma que os objetos com características de publicadores devem herdar os métodos definidos pela classe Publisher. Um exemplo de método subscritor está nas interfaces homem-máquina, definidas pelas classes GUISub.

3.2.3 Classes Manager

As classes Manager são as classes responsáveis pela instanciação de todos os objetos concretos utilizados no sistema. Além disso, em uma situação real de utilização do modelo proposto, as classes Manager podem ser utilizadas como uma maneira de se facilitar o gerenciamento e a monitoração dos equipamentos físicos, como proposto em [CHENG, 2000].

No modelo proposto, existem cinco classes principais do tipo Manager, a saber:

- Classe PubManager;
- Classe SubManager;
- Classe SensorManager;
- Classe AtuadorManager;
- Classe GUIManager.

Estas classes relacionam-se conforme o apresentado na figura 45.

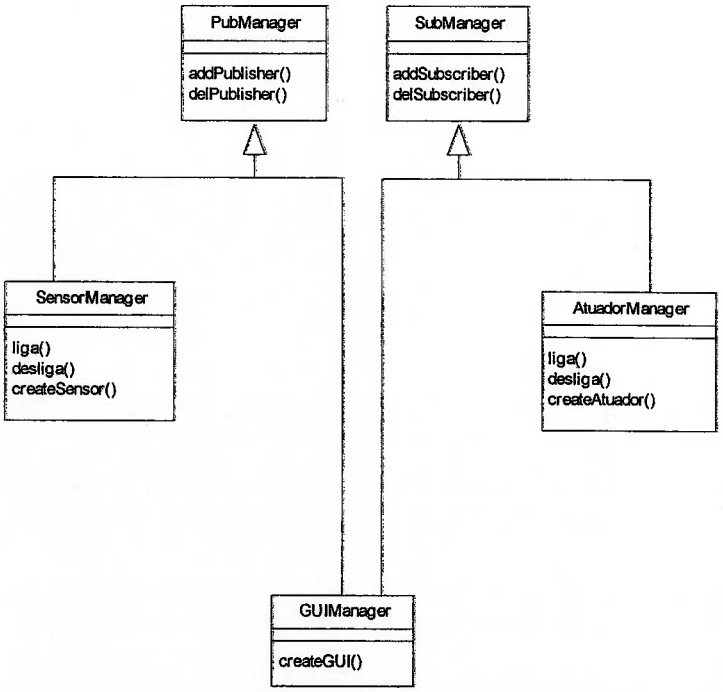

Figura 45 - Relacionamento das classes Manager

As classes PubManager e SubManager possuem uma relação de agregação com as classes Publisher e Subscriber respectivamente, e possuem os métodos responsáveis pela adição e remoção de elementos publicadores ou subscritores ao sistema. Na realidade, as classes PubManager e SubManager não chegam a ser instanciadas no sistema, mas servem como classes pai às outras classes Manager.

A classe SensorManager gerencia a criação e a operação de elementos do tipo sensor, tradicionalmente publicadores. Através desta classe, os sensores podem ser criados e destruídos, ligados e desligados, e terem seus parâmetros de operação alterados. A classe SensorManager define ainda o nome a ser dado a cada elemento sensor, e realiza a publicação desse nome junto ao serviço de nomes para que sua localização

realiza a publicação desse nome junto ao serviço de nomes para que sua localização posterior seja facilitada. De uma maneira geral, a classe SensorManager atua segundo como um "Criador Concreto" do *pattern Factory* apresentado anteriormente.

A classe AtuadorManager possui uma função análoga à SensorManager, aplicada aos atuadores do sistema, como válvulas, motores, cilindros, solenóides ou transportadores.

Já a classe GUIManager apresenta a função de gerenciar as interfaces com os operadores, sejam elas interfaces de consulta ou de operação do sistema. Essas interfaces podem estar relacionadas diretamente aos sensores e atuadores, ou seja, podem exibir a condição destes elementos diretamente, como também podem estar associadas a controladores (PLC's, por exemplo). Desse modo, a classe GUIManager gerencia tanto interfaces relacionadas a elementos publicadores como interfaces relacionadas a elementos subscritores.

Na implementação do sistema, foi definida uma interface padrão, que pode ser configurada pelo objeto que desejar utilizá-la. Por exemplo, um sensor pode operar associando uma interface padrão a si próprio, como também pode operar sem qualquer tipo de interface. A janela básica idealizada para a interface dos elementos do sistema com os usuários está apresentada na figura 46.

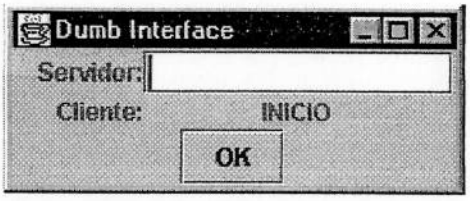

Figura 46 - Interface genérica dos elementos do sistema

Em sua versão básica (apresentada na figura 46), essa interface apresenta apenas o nome do elemento com que está relacionada (no caso, um elemento chamado Dumb); uma linha para a entrada de dados, uma linha para a apresentação do status corrente do elemento e um botão de ação. Os textos indicativos de cada um desses elementos também podem ser alterados pelo objeto que aciona a interface e as funções de cada campo podem ser modificadas.

A fim de se ilustrar melhor o potencial de personalização da classe que implementa a interface gráfica, seguem imagens das versões mais complexas de interface usadas para a implementação de componente sensor (figura 47) e um componente GUI (figura 48) na implementação do protótipo.

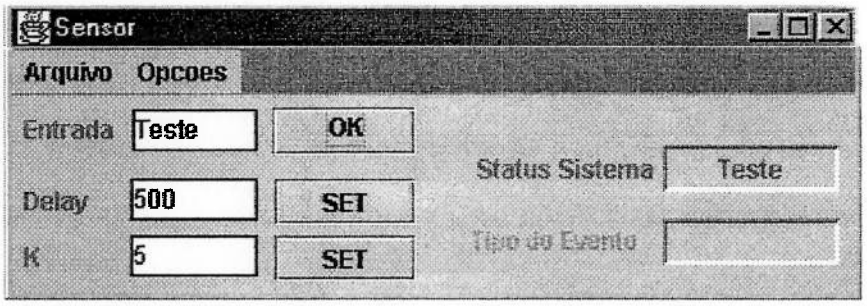

Figura 47 - Interface adaptada para um elemento Sensor

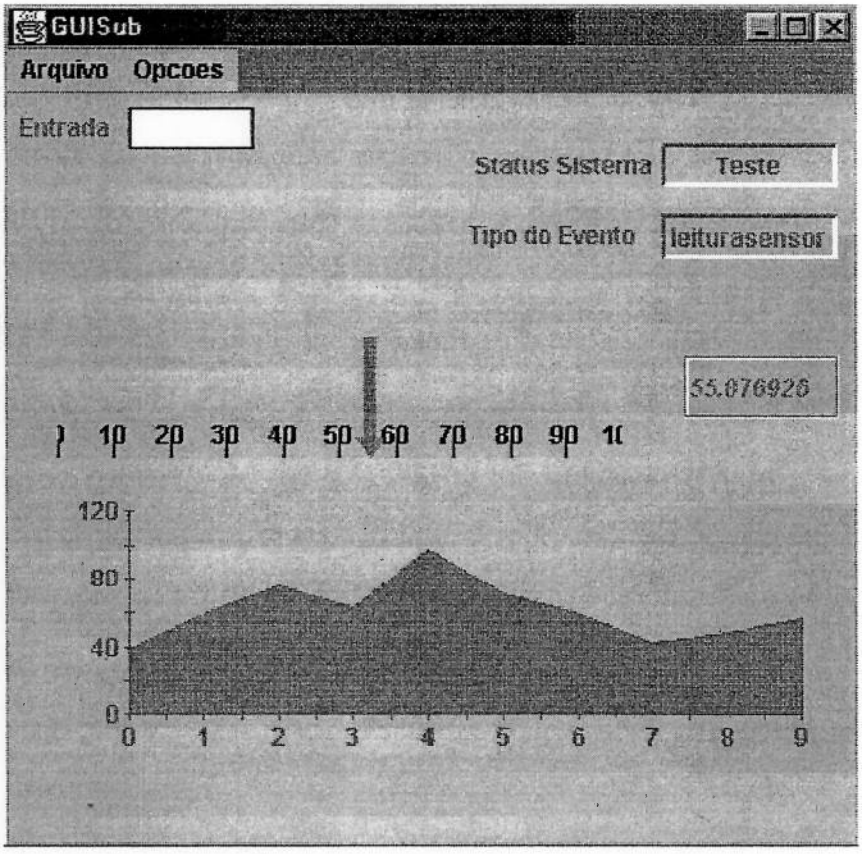

Figura 48 - Interface adaptada para um elemento de interface homem-máquina (IHM)

A personalização da interface padrão é normalmente realizada pelo objeto que a aciona no momento de sua instanciação. Entretanto, é possível também adicionar-se e

remover-se componentes da interface em tempo de execução, como é o caso do gráfico e da barra indicativa do valor medido apresentados na figura 48.

Devido às características da programação orientada a objetos e à arquitetura utilizada, podem-se incluir nas interfaces quaisquer tipos de componentes gráficos, sejam eles componentes desenvolvidos especificamente para uma determinada implementação de sistema MES, ou componentes padrão disponíveis no mercado, como é o caso do componente responsável pela criação do gráfico da figura 48.

Além dessas cinco classes Manager básicas, existem ainda mais duas propostas no modelo: a classe ControladorManager e a classe ConsoleManager. A classe ControladorManager gerencia os PLC's e demais equipamentos de controle que possam estar associados ao sistema, com a particularidade de que esses controladores podem atuar tanto como publicadores (ao divulgar, por exemplo, informações referentes às leituras de sensores sem capacidade de publicação de dados), quanto como subscritores, ao receberem dados de sensores para análise e realização de ajustes no processo produtivo. Já a classe ConsoleManager gerencia os painéis de operação que são usados pelos operadores humanos do sistema para as atividades de acionamento e ajustes de equipamentos.

3.2.4 As classes instanciáveis do sistema

No modelo apresentado, existe ainda uma outra categoria de classes. São as classes que representam os objetos concretos inseridos no contexto do sistema de controle, como os sensores, os atuadores, os controladores, os consoles e as interfaces homem-máquina.

É importante ressaltar que esses elementos podem participar do sistema de duas maneiras distintas:

1 – Podem ser elementos virtuais, ou seja, componentes de software que emulam um sensor ou um atuador, como por exemplo no caso de um computador com uma placa de aquisição de dados, cujo software pode interagir diretamente com o sistema de controle e supervisão proposto, ou seja, pode ser uma instância direta da classe sensor proposta;

2 – Podem ser elementos concretos, com ou sem capacidade de transmissão de dados, mas que são interfaceados a uma instância de uma das classes propostas e que são percebidos de forma transparente pelo sistema. Por exemplo, um atuador pode estar conectado a um pequeno controlador baseado em PC, e executando uma instância da classe atuador proposta. Nesse caso, o sistema vê o controlador como sendo o próprio atuador físico, e o controlador se encarrega de realizar o acionamento do dispositivo físico.

Embora essas duas condições de participação no sistema sejam possíveis, não há necessidade de representá-las de formas distintas no modelo. As particularidades de cada interface com os dispositivos físicos ficam encapsuladas dentro da implementação das classes representativas de cada elemento concreto do sistema.

Existem as seguintes classes básicas, representativas de elementos concretos:

- Classe Sensor;
- Classe Atuador;
- Classe Controlador;
- Classe Console;
- Classe GUIPub;
- Classe GUISub.

A classe Sensor representa todos os elementos que realizam a medição de grandezas contínuas ou discretas no sistema, como termopares, sensores de nível, detetores de fim de curso, leitores de códigos de barra, transdutores de pressão ou encoders. Quando cada um desses sensores é instanciado, a classe SensorManager se encarrega de registrar seu nome e sua categoria no serviço de nomes, de modo a possibilitar sua localização no sistema de forma rápida.

A classe Atuador representa todos os elementos físicos que exercem algum efeito sobre o processo ou sobre as máquinas do sistema, como os pistões, solenóides, válvulas, motores e transportadores. São gerenciados pela classe AtuadorManager e

também são classificados de acordo com seu tipo e função junto ao serviço de nomes. Tipicamente, os Atuadores são elementos subscritores no sistema de controle.

Tem-se também a classe Controlador. Essa classe representa todas as CPU's de atuação direta no processo, como os controladores (PLC's) das máquinas de produção, os comandos das máquinas CNC e os inversores de freqüência dos motores de médio e grande porte. Esses elementos apresentam uma particularidade importante, que é a de comportarem-se em determinados momentos como publicadores e em outros como subscritores, ou seja, os controladores podem necessitar receber informações de sensores ou também publicar os dados referentes a cálculos ou leituras de dispositivos a eles conectados.

Já a classe Console tem por objetivo representar os painéis de comando simples dos equipamentos que não dispõem de controladores para seu acionamento. Esses consoles são comandados diretamente por operadores humanos, e dispõe apenas de lógicas simples implementadas com chaves físicas como relês para comandar máquinas. Desse modo, a função da classe Console é a de possibilitar que seja criada uma interface com esses painéis, a fim de trazer seus comandos e seus status para dentro do sistema integrado de controle proposto, ou seja, possibilitar que o sistema tenha acesso às informações relevantes desses consoles de modo transparente.

As classes GUISub e GUIPub, por sua vez, são elementos responsáveis por implementar ferramentas básicas de exibição de dados e status de elementos subscritores e publicadores, respectivamente. Essas classes fazem uso de uma interface básica e genérica, e podem ser associadas a qualquer tipo de elemento do sistema, como sensores, atuadores, controladores e consoles. Sempre que se deseje que um usuário humano do sistema de supervisão e controle tenha acesso ao status de um componente, pode-se exibir sua interface básica, independentemente da função do componente em questão.

As classes GUISub e GUIPub podem também ser usadas para a criação de telas de supervisão mais complexas, como, por exemplo, a tela de um terminal IHM.

3.3 Implementação do protótipo

A fim de se demonstrar a validade da arquitetura proposta, desenvolveu-se um protótipo funcional que implementa o modelo apresentado e possibilita a realização de análises de itens como performance e confiabilidade na transmissão de dados. Esse protótipo foi desenvolvido na linguagem Java, e faz uso da infra-estrutura CORBA implementada pelo produto ORBACUS, da empresa americana OOC (Object Oriented Concepts).

Optou-se por esta combinação de linguagem e ferramenta CORBA principalmente em função das características de independência de plataforma da linguagem Java, pelo fato de Java tratar-se de uma linguagem orientada a objetos e pela a compatibilidade da linguagem com a ferramenta ORBACUS. Além disso, a ferramenta ORBACUS oferece implementações de todos os serviços necessários à construção do protótipo, como o Serviço de Nomes, o Serviço de Eventos e o Serviço de Notificação, bem como interfaces para administração desses serviços.

Para a implementação do protótipo, optou-se por desenvolver um sistema parcial, que demonstrasse o funcionamento da comunicação através do serviço de notificação, com um componente de software que emulasse o comportamento de um sensor genérico, e um segundo componente que simulasse uma interface interessada em receber notificações de leitura desse sensor.

Para tanto, implementou-se uma classe SensorManager, responsável por gerenciar os objetos sensores; uma classe Sensor, que implementa um emulador de sensor genérico; uma classe GUIManager, responsável por gerenciar as interfaces interessadas nos dados do sensor; e uma classe GUISub, responsável por receber e exibir os dados enviados pelo sensor.

O esquema de comunicação escolhido para a implementação do primeiro protótipo foi o esquema Push, e o tipo de dados escolhido para ser transferido entre o sensor e a interface foi o tipo estruturado.

Foi necessária, além das classes já mencionadas, a implementação de duas classes adicionais de suporte ao sistema protótipo: as classes GeneralGUI e MyObjectRef. A

classe GeneralGUI tem a atribuição de formar na tela a interface gráfica necessária à interação do usuário com os elementos do sistema e exibir os resultados obtidos pelo elemento subscritor. Já a classe MyObjectRef tem a função de possibilitar a gravação e a leitura de referências (IOR's) nos canais, uma vez que esta funcionalidade não é oferecida pronta pela ferramenta ORBACUS.

3.3.1 Descrição do processo de funcionamento do Sensor

O protótipo atual baseia-se diretamente nas implementações das classes SensorManager e GUIManager. São essas as classes principais e executáveis, que são chamadas pelo usuário para a inicialização do sistema. Pode-se optar por executar as classes em qualquer ordem, mas, para efeito dessa descrição, será apresentado um caso em que se executa primeiro a classe SensorManager, e depois a classe GUIManager.

Quando a classe SensorManager é executada, as seguintes tarefas são realizadas:

1 – Inicializam-se o ORB e o BOA, criando-se a infra-estrutura básica para a execução de uma aplicação CORBA;

2 – Inicializa-se o Serviço de Nomes;

3 – Cria-se o contexto de nomes SCADA;

4 – Instancia-se a classe Sensor, define-se um nome à mesma, e publica-se a referência a essa instância no Serviço de Nomes;

5 – Cria-se uma interface local para o objeto Sensor recém-criado e exibe-se a mesma;

6 – Aguarda-se que o usuário clique no botão "OK" da interface para que um evento seja gerado e enviado através do canal;

Após a execução dessas atividades, um elemento Sensor encontra-se criado, e pronto para gerar e enviar eventos através do serviço de notificação. A geração desses eventos é motivada pelo acionamento do botão "OK" da interface com o usuário, e os eventos gerados são compostos no mínimo por um valor de leitura aleatório, e a data e hora do evento em questão. Quando o botão OK é acionado, as seguintes atividades acontecem no elemento sensor:

1 – Verifica-se se já existe um canal de eventos criado. Caso exista, obtém-se a referência ao mesmo; caso contrário, cria-se um canal e grava-se a referência em um arquivo, chamado de "channel.ref";

2 – Obtém-se um proxy subscritor para o envio dos eventos. A função do proxy é a de se comportar da mesma maneira que o elemento subscritor, oferecendo o método Push que será chamado para a transmissão do evento. A grande vantagem do uso do proxy, é que ele colabora no isolamento dos objetos publicadores e subscritores, a ponto de não ser necessário que haja uma instância de um subscritor no ar para que o sensor consiga enviar um evento ao canal;

3 – Gera-se o evento a ser enviado pelo sensor, composto de um valor de leitura aleatório e o valor da data e hora locais no instante da criação;

4 – O evento é transmitido para o canal através da chamada do método push_strutctured_event presente no proxy.

Nesse instante, os dados gerados pelo sensor já estão disponíveis no canal de eventos gerado, e podem ser acessados por qualquer quantidade de subscritores que tenham interesse nos mesmos.

3.3.2 Descrição do processo de funcionamento da GUISub

Considerando-se que a classe executável GUIManager tenha sido acionada após a execução da classe SensorManager, a seqüência de tarefas que seriam executadas é conforme segue:

1 – O ORB e o BOA são inicializados, da mesma maneira que o acontecido quando se executa a classe SensorManager;

2 – É gerada uma interface gráfica para o objeto GUISub a ser instanciado;

3 – Verifica-se a existência de um canal de eventos através do nome do arquivo "channel.ref". Caso o mesmo não exista, ele é criado e sua referência armazenada nesse mesmo arquivo;

4 – É criado um proxy fornecedor, de modo que é então conectado ao canal de eventos;

5 – Instancia-se a classe GUISub, obtendo-se um objeto subscritor. É importante observar que esse objeto subscritor estende uma interface básica definida pelo Serviço de Notificação, e chamada de _StructuredPushConsumerImplBase. É nessa interface que é declarado o método push_strutctured_event também presente no proxy consumidor utilizado pelo objeto sensor para transmitir os eventos gerados;

6 – Conecta-se o objeto GUISub criado ao proxy fornecedor criado, sendo que o mesmo passa a receber os eventos enviados ao canal pelo objeto sensor.

Assim que recebe um novo evento, o elemento GUISub realiza sua decomposição, identificando o cabeçalho do evento, o valor da leitura e a data e hora de sua geração. Esses dados são, então, exibidos na tela através da interface genérica para ele criada.

4 Resultados Obtidos e Discussão

4.1 Aspectos gerais da implementação

Todo o código do protótipo implementado para esta finalidade baseia-se na linguagem Java, tendo sido utilizado o kit de desenvolvimento versão 1.3. Já a implementação CORBA selecionada foi a ferramenta ORBACUS, da OOC (*Object Oriented Concepts*) em sua versão 3.

Da maneira que está implementado, o protótipo pode ser utilizado normalmente em um ambiente de rede, com os elementos publicadores e subscritores rodando em máquinas independentes. Um dos pontos chave para a obtenção dessa característica de independência da localização física dos componentes do sistema deve-se principalmente à utilização do Serviço de Nomes para a publicação da localização dos componentes do sistema e dos canais de dados que precisam ser acessados tanto do lado servidor quanto do lado cliente.

Os dois serviços utilizados no protótipo: o Serviço de Nomes e o Serviço de Notificação podem ser executados normalmente em máquinas distintas na rede, não precisando ser atrelados nem à máquina que executa o componente publicador e nem à máquina que executa o componente subscritor. A localização destes serviços por parte dos elementos do sistema dá-se através de um arquivo de configuração que é chamado no momento da execução do ORB nas classes SensorManager e GUIManager.

Esse arquivo de configuração é passado como parâmetro quando se executam as classes Manager, e inclui informações sobre a localização dos servidores de nomes e de notificação, através do endereço IP da máquina em que estão sendo executados e da porta lógica que estão usando para a conexão de clientes. O serviço de notificação demanda ainda a definição do local onde é executado um banco de dados, que utiliza para manter armazenados quaisquer elementos que requeiram algum tipo de persistência. Pode-se ver um exemplo do conteúdo de um arquivo de configuração na figura 49.

```
ooc.service.NameService=corbaloc::192.192.7.8:27000/NameService
ooc.service.EventService=corbaloc::192.192.7.8:25000/DefaultEventChannel
ooc.service.TypedEventService=corbaloc::192.192.7.8:25000/DefaultTypedEventChannel
ooc.service.EventChannelFactory=corbaloc::192.192.7.8:25000/DefaultEventChannelFactory
ooc.service.TypedEventChannelFactory=corbaloc::192.192.7.8:25000/DefaultTypedEventChannelFactory
ooc.service.PropertyService=corbaloc::192.192.7.8:28000/DefaultPropertySetDefFactory
```

Figura 49 - Exemplo de arquivo de configuração

O Serviço de Notificação foi totalmente implementado em C++, ao contrário de todos os outros serviços e componentes utilizados, incluindo-se o console de gerenciamento do próprio Serviço, que foram implementados em Java. Apesar dessa diferença, há uma perfeita compatibilidade com o restante do software. A justificativa apresentada pelos responsáveis pelo produto é de que utilizou-se C++ a fim de se privilegiar o desempenho do software mas, como não há uma implementação disponível em Java até o momento, não foi possível realizarem-se comparações para se quantificar essa diferença de desempenho.

Uma das características do protótipo implementado até o momento está no alto grau de desacoplamento entre os componentes publicadores e subscritores. Uma vez que todos os métodos de transmissão de eventos ou dados entre publicadores e subscritores são derivados das interfaces já definidas pela especificação do Serviço de Notificação, os componentes podem ser executados de forma independente, sem perigo de acessarem um método de um objeto que não esteja no ar. Isto ocorre porque os métodos em questão não são acessados diretamente no objeto alvo, mas sim no proxy que o representa, isolando, dessa forma, os dois tipos de componentes.

Por exemplo, no caso de um componente publicador que usa o esquema de comunicação Push para enviar um evento a um componente subscritor, o método push_strutctured_event chamado pelo publicador está localizado no proxy subscritor, e não na instância real do subscritor. Na prática, o método localizado no proxy apenas

transfere a chamada para o método real implementado no subscritor, mas o fato de o publicador realizar a chamada da função pelo proxy o protege de um erro potencial, caso o subscritor não esteja no ar.

Entretanto, a contrapartida dessa vantagem está em que o método subscritor, nesse esquema de comunicação, deve estender uma classe já definida e implementada no Serviço de Notificação, limitando a liberdade do projetista de sistemas que queira usar o serviço.

A implementação do protótipo obteve sucesso na comunicação de um evento estruturado simples entre um elemento publicador (representado por um sensor típico, instância da classe Sensor) e um ou mais elementos subscritores (representados por uma interface homem-máquina típica, instância da classe GUISub). Os eventos foram corretamente inseridos no canal de eventos pelo elemento publicador, recebidos pelo elemento subscritor e devidamente interpretados e exibidos na tela.

4.1.1 Estrutura dos Eventos Gerados

Na implementação do protótipo utilizada para a análise de desempenho a ser descrita a seguir, optou-se pelo esquema de comunicação do tipo "*Push*" tradicional, ou "*Push*" canônico, conforme definido em [OOC, 2000a].

Nesse esquema de comunicação, o elemento subscritor conecta-se a um canal de eventos, através do qual elemento publicador efetua a inserção de dados. Os dados inseridos pelo elemento publicador são então enviados ao elemento subscritor pelo canal de eventos (ou "*event channel*"). Uma visão geral deste processo de comunicação pode ser obtida a partir da figura 50, abaixo.

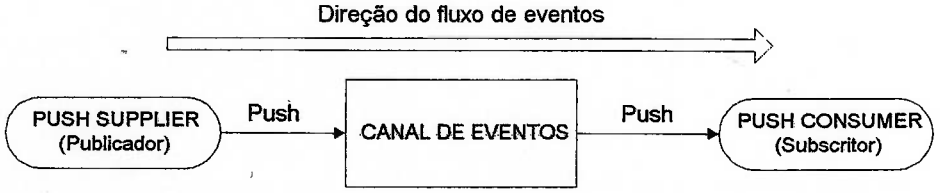

Figura 50 - Comunicação no esquema "*Push*" tradicional

Os eventos gerados pelos elementos publicadores do protótipo implementado fazem uso da capacidade do serviço de Notificação de trabalhar com eventos estruturados, possibilitando o envio de mensagens através de uma estrutura bem definida, em que diversos tipos de eventos podem ser mapeados.

A forma geral idealizada pela OMG para os eventos estruturados transmitidos através do serviço de notificação pode ser vista na figura 51.

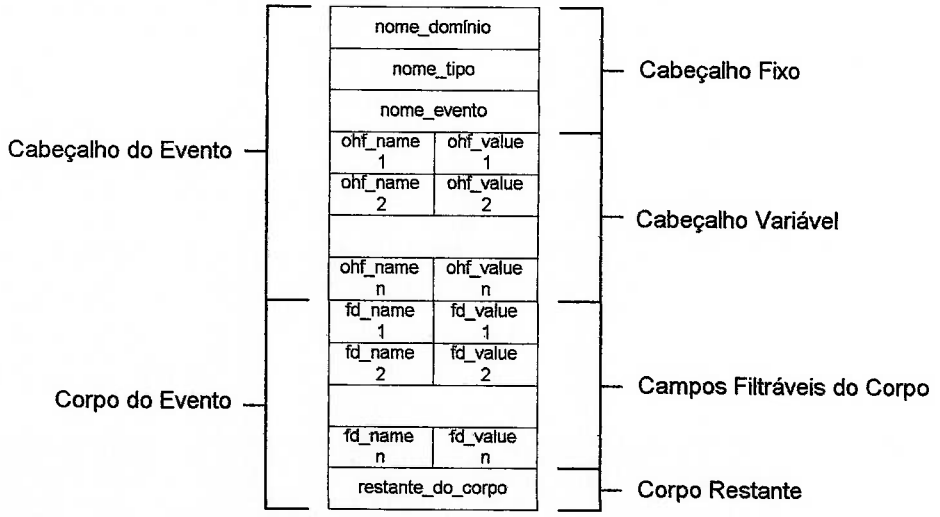

Figura 51 - Estrutura geral de um Evento Estruturado[26]

No protótipo implementado, os eventos gerados pelos elementos publicadores possuem de um cabeçalho fixo em que se informa o tipo do evento publicado (como por exemplo "medição"), uma informação mais específica sobre sua origem (como por exemplo "leiturasensor") e ainda um campo que pode ser definido pelo usuário quando o elemento sensor é instanciado.

Além do cabeçalho fixo, os eventos também possuem três campos de dados filtráveis, em que são publicados um valor numérico de leitura do sensor (no protótipo um número aleatório variando entre 0 e 100), um "*string*" apresentando a data e horário da geração do evento pelo publicador, e finalmente um campo numérico em que é lançado o horário da geração do evento em formato numérico, de modo que seja

[26] Adaptado de [OMG, 2000a]

possível, no elemento subscritor, a realização do cálculo do tempo utilizado na comunicação do evento. Um exemplo de evento pode ser visto na figura 52.

Cabecalho Fixo: medicao.leiturasensor.as

Dados Filtraveis :

Leitura : 86.71173

Horario : Sun Sep 22 11:13:16 GMT-03:00 2002

Figura 52 - Exemplo de evento gerado por um elemento publicador no protótipo

No elemento subscritor, os eventos recebidos são decompostos e apresentados em uma interface gráfica (*GUI – Graphical User Interface*) e em uma janela de texto; já o tempo de atraso verificado entre a transmissão e a recepção do evento (*delay*) é calculado, apresentado em milissegundos na janela de texto e também armazenado em um arquivo para que seja possível efetuar-se a análise de desempenho do sistema. Uma saída típica em formato texto dos elementos subscritores pode ser vista na figura 53, abaixo.

ID do evento: org.omg.CosNotification.StructuredEvent@4aa2db

Cabecalho Fixo: medicao.leiturasensor.as

Dados Filtraveis:

Leitura : 86.71173

Horario : Sun Sep 22 11:13:16 GMT-03:00 2002

Time St : 1032703996650

Horario Local : Sun Sep 22 11:13:16 GMT-03:00 2002

Figura 53 - Exemplo de evento recebido por um elemento subscritor no protótipo

4.2 Análise de Desempenho do Sistema

Testou-se a comunicação dos eventos estruturados em situações com até 5 subscritores simultâneos ao mesmo canal e todas as instâncias receberam e exibiram na tela os eventos de maneira adequada. Entretanto, observou-se que a diferença de tempo entre a geração dos eventos e sua exibição na tela pelos elementos subscritores apresenta uma variação significativa em função das condições do ambiente de execução do protótipo.

Essa diferença de tempo observada, doravante chamada de *delay*, apresenta variações de comportamento em função de basicamente três aspectos principais:

- Ambiente (Hardware e Sistema Operacional) em que está sendo executado cada um dos componentes do sistema;
- Tempo decorrido entre duas publicações de eventos pelo elemento publicador, ou seja, freqüência do sinal de leitura;
- Grau de complexidade da interface de recepção de eventos pelo elemento subscritor, ou seja, carga computacional demandada pelo elemento subscritor.

A fim de isolarem-se esses três aspectos principais, os testes realizados com o protótipo foram executados variando-se a localização dos componentes publicador, subscritor, do Serviço de Nomes e do Serviço de Notificação entre quatro configurações de máquinas diferentes, identificadas conforme a tabela 3.

Tabela 3 – Equipamentos usados nos testes

Identificação	*Configuração da máquina*
Máquina 1	Processador Pentium II com *clock* de 300 MHz e 256 MB de memória RAM, rodando Windows NT versão 4 com *Service Pack* 6
Máquina 1B	Máquina 1 com sistema operacional Windows 98
Máquina 2	Processador AMD K6-II com *clock* de 500 MHz e 128 MB de memória RAM, rodando o sistema operacional Windows 98
Máquina 3	Processador Pentium II com *clock* de 300 MHz e 64 MB de memória RAM, rodando Windows XP Professional

Todas as máquinas rodavam o *kit* de desenvolvimento Java (JDK) versão 1.3 e encontravam-se ligadas em rede, através de uma conexão de 100 MBPS, conforme pode ser visto na figura abaixo.

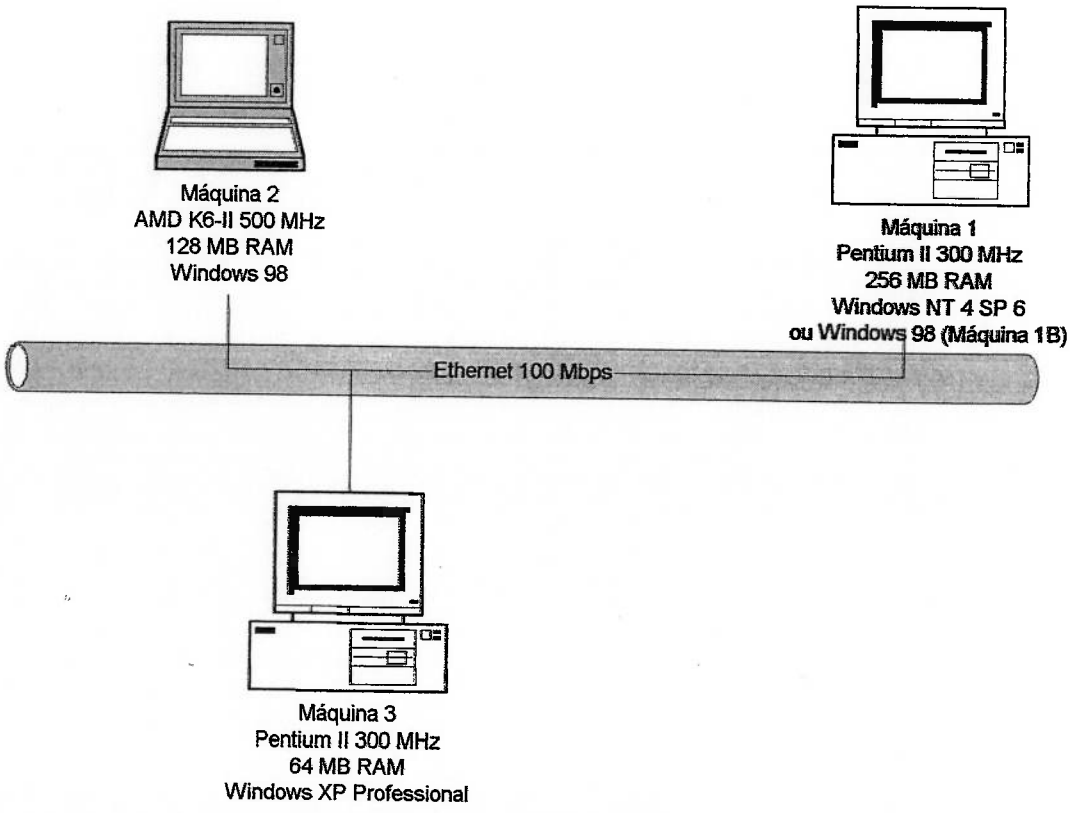

Figura 54 - Ambiente de execução do protótipo

Além da variação nas máquinas em que os componentes foram executados, variou-se ainda um parâmetro responsável por controlar a freqüência de envio dos eventos,

ajustado através do tempo de espera entre duas publicações de eventos consecutivas pelo elemento publicador (doravante chamado parâmetro *Sleep*), um parâmetro responsável por controlar o número de eventos gerados em cada teste de execução (doravante chamado parâmetro K) e o aspecto da interface GUI usada para exibição dos dados recebidos pelo elemento subscritor.

A motivação da variação do parâmetro *Sleep* está na necessidade de verificação do comportamento geral do sistema em situações em que a taxa de transmissão dos sinais enviados pelo sensor seja alta (até 50 ms entre cada leitura) e em situações de taxa mais baixa (da ordem de 1 s entre leituras).

Já a variação no aspecto da interface GUI foi realizada por dois motivos principais: o primeiro pelo fato de as diversas interfaces homem-máquina presentes em um sistema de controle variarem de maneira significativa em função da necessidade de cada operador [PENNER, 2002], de modo que um estudo do comportamento do protótipo deve levar em conta esta variação; e o segundo pelo fato de ser uma maneira relativamente simples de se variar o grau de complexidade no tratamento dos eventos pelo elemento subscritor e avaliar-se a influência desta complexidade no desempenho global do sistema.

A fim de se identificarem os dois diferentes aspectos da GUI do elemento subscritor utilizados nos testes, convencionou-se chamar a interface GUI completa, com exibição de um gráfico de histórico de leituras recebidas e uma barra de indicação do valor de leitura corrente como "GUI Tipo 1", e a interface GUI mais simples, com a apresentação apenas do valor numérico da leitura recebida e a decomposição do evento como "GUI Tipo 2". O aspecto de ambas as interfaces pode ser visto na figura 55 a seguir.

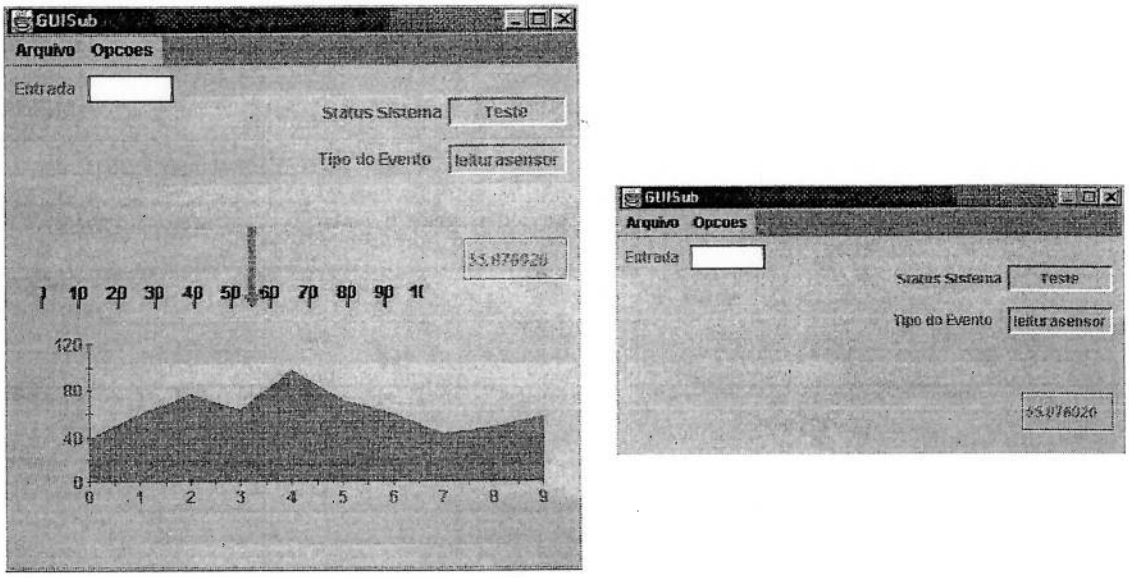

Figura 55 - Aspectos das GUI Tipos 1 (esquerda) e 2 (direita)

O índice escolhido para a avaliação do desempenho global do sistema foi o atraso, ou *delay*, observado entre a publicação de um evento pelo elemento publicador (no caso do protótipo, um elemento do tipo Sensor) e o recebimento desse mesmo evento pelo elemento subscritor (no caso do protótipo, um elemento do tipo GUISub).

4.2.1 Testes iniciais na máquina 1, com *Sleep* = 500 ms (Teste # 1)

Em uma primeira bateria de testes, o sistema todo foi executado na máquina número 1 (Windows NT), com um intervalo de tempo entre as transmissões de eventos de 500 ms e uma quantidade total de 1000 repetições. Os serviços de Nomes e Notificação também foram executados nessa mesma máquina. As condições do teste estão relacionadas na tabela 4.

Tabela 4 – Condições do Teste # 1

Variável	*Valor*
Local de execução do Publicador	Máquina 1
Local de execução do Subscritor	Máquina 1
Local de execução dos Serviços	Máquina 1
Parâmetro *Sleep*	500 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

O teste foi repetido por 10 vezes, e o aspecto geral da curva que representa o atraso observado em função da quantidade de eventos enviadas está apresentado na fig. 56.

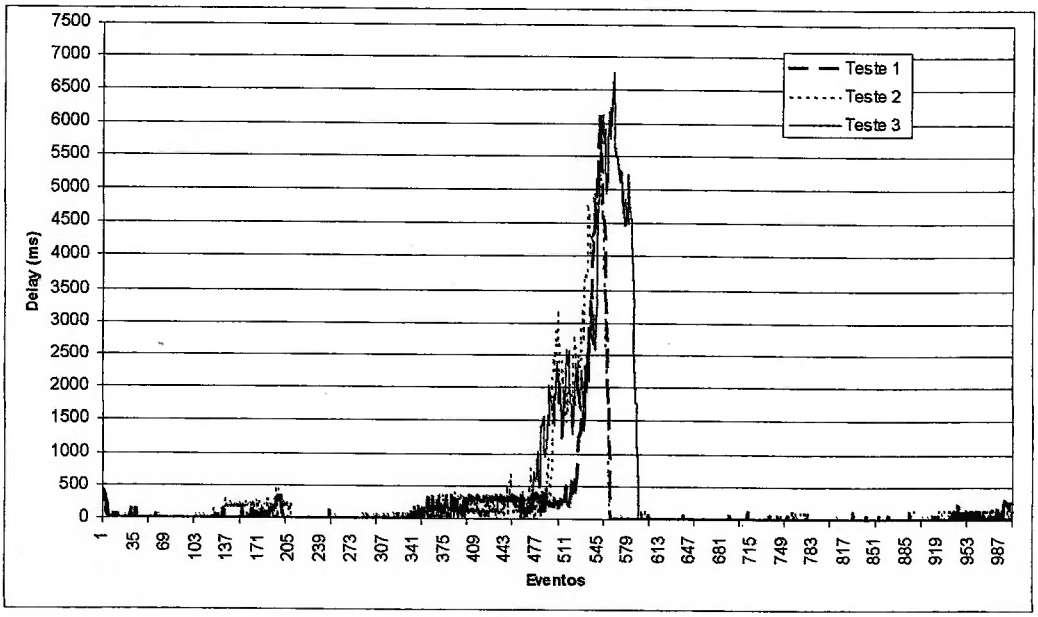

Figura 56 - Curva de atraso para a máquina 1, com Sleep = 500, K = 1000 e GUI tipo 1.

Para efeito de maior clareza no gráfico da figura acima, foram apresentados apenas os resultados de três repetições do teste, mas uma curva gerada a partir da média das leituras do *delay* verificada nas diversas repetições realizadas pode ser observada na figura 57 abaixo.

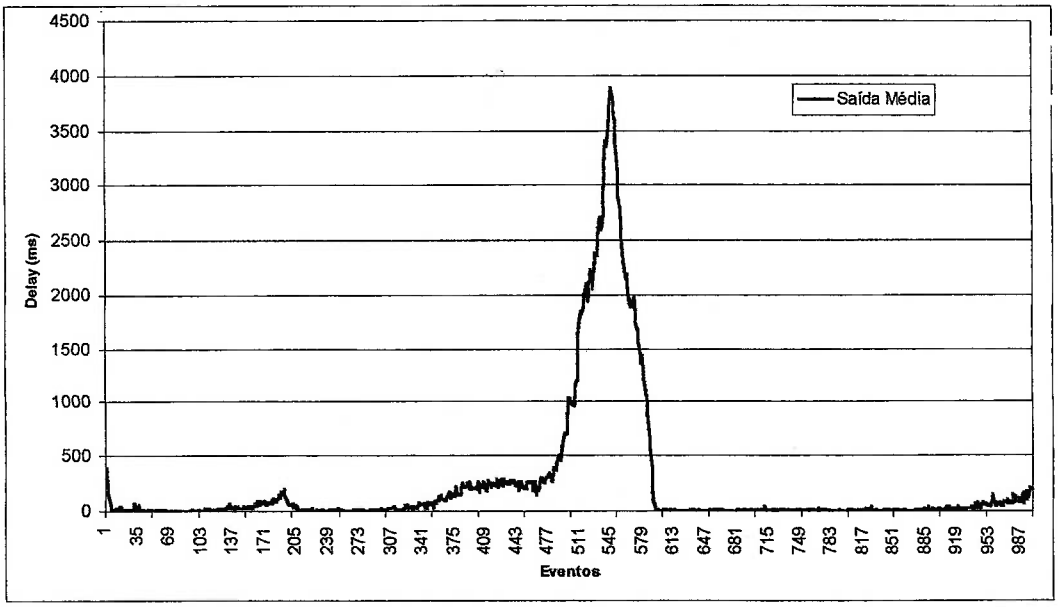

Figura 57 - Curva média de atraso para a máquina 1, com Sleep = 500, K = 1000 e GUI tipo 1.

Conforme pode-se observar nos gráficos acima, o tempo de *delay* nessa situação inicia na casa dos 300 ms para os primeiros 4 ou 5 eventos enviados, mas cai rapidamente e permanece na casa dos 10 ms com pequenos picos em torno dos 100 ms até por volta do evento de número 500, quando há um aumento rápido dos tempos de *delay* até um pico da ordem de 4 s (ou 4000 ms), após o qual cai novamente para a faixa de 10 ms. A média global encontrada foi de aproximadamente 270 ms.

Optou-se então, pela realização de algumas seqüências de testes com uma quantidade maior de iterações, da ordem de 5000 eventos (K = 5000) para que fosse possível a análise do comportamento da curva com um horizonte maior de iterações.

O resultado obtido pode ser verificado no gráfico da figura 58, que representa a média das curvas levantadas:

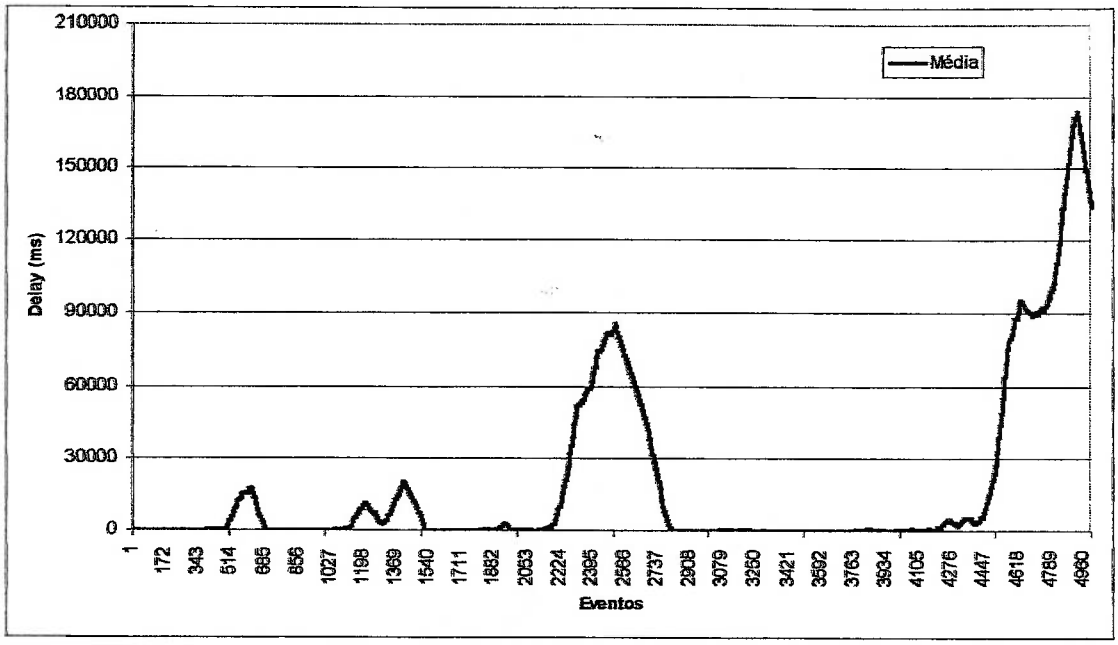

Figura 58 - Curva média de atraso para a máquina 1, com Sleep = 500, K = 5000 e GUI tipo 1.

Como pode ser percebido através do gráfico, surgem diversos picos no valor do atraso, que tendem a ser crescentes. Fora das regiões de pico, os valores observados para o atraso permanecem na faixa dos 10 ms.

A amplitude dos picos sofreu uma certa variação ao longo das várias execuções do primeiro teste; entretanto, o aspecto das curvas obtidas foi sempre o mesmo.

4.2.2 Testes com ambiente de execução otimizado (Testes 2, 3 e 4)

A fim de isolarem-se ao máximo os efeitos do ambiente em que o protótipo estava sendo executado, e esclarecerem-se as causas do aspecto da curva de atrasos obtida nos primeiros testes com a máquina 1, optou-se por refazerem-se os mesmos testes, com a mesma máquina e versão de sistema operacional (S.O.), mas em uma instalação nova do sistema, em que foram instalados e executados apenas seus serviços mínimos e removidos todos os programas residentes em memória que pudessem estar afetando o resultado. A essa nova configuração otimizada, deu-se o nome de "configuração mínima para a máquina 1".

Tabela 5 – Condições do Teste # 2 (com uma instalação mínima do S.O.)

Variável	*Valor*
Local de execução do Publicador	Máquina 1
Local de execução do Subscritor	Máquina 1
Local de execução dos Serviços	Máquina 1
Parâmetro *Sleep*	500 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

O gráfico com a curva de atraso obtida, com os mesmos parâmetros do Teste # 1, mas com uma configuração otimizada do sistema operacional, pode ser visto na figura 59 a seguir.

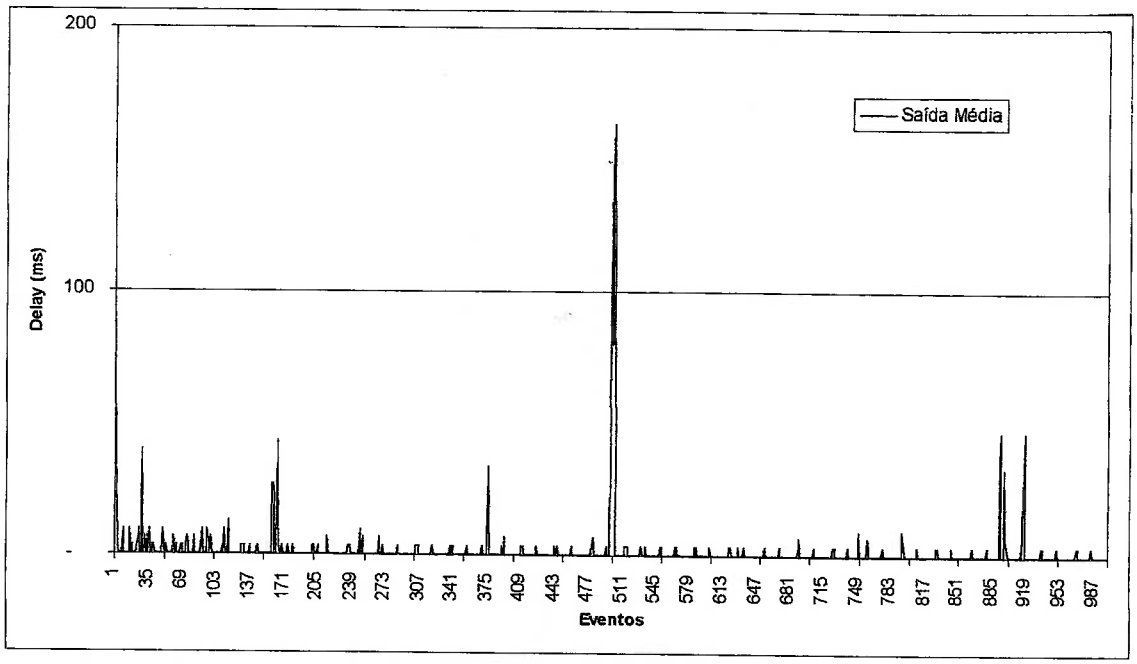

Figura 59 - Curva média de atraso para a máquina 1, na configuração mínima (Teste # 2).

Como pode-se observar, os índices de *delay* para a nova configuração do ambiente de execução do protótipo são muito menores do que os anteriores, apesar do protótipo estar sendo executado no mesmo *hardware* e com os mesmos parâmetros. Para efeito de comparação, segue abaixo uma tabela de dados comparativos dos resultados obtidos nas duas situações de teste.

Tabela 6 - Comparação dos atrasos na máquina 1 para Sleep = 500 ms

Ambiente	Atraso Máximo	Atraso Médio
Teste # 1	7961 ms	184 ms
Teste # 2	421 ms	1,7 ms

Observou-se que, em ambos os casos, os picos de atraso máximo ocorrem entre o evento de número 500 e o de número 550, ou seja, aproximadamente após 4,2 minutos após o início do teste, embora no caso do Teste #1 a amplitude do pico tenha sido cerca de 20 vezes maior do que no Teste # 2 e com duração consideravelmente mais longa.

Desse modo, a fim de se possibilitarem comparações realmente coerentes entre os diversos ambientes de teste usados na execução do protótipo, optou-se por realizar

todos os demais testes com o mínimo possível de serviços do sistema operacional em operação e eliminando-se todos os softwares residentes em memória que pudessem interferir nos resultados medidos. Para efeitos de comparação com os resultados obtidos nas outras plataformas de teste, desconsideraram-se os resultados do Teste # 1 e utilizaram-se apenas os resultados do Teste # 2.

Além das execuções realizadas com o intervalo entre publicações de eventos em 500 ms (taxa de transmissão em 2 Hz), foram realizados também outras execuções, sempre com o mínimo ambiente em termos de sistema operacional e com o intervalo entre eventos em 50 ms e 1s.

O resultado obtido para a máquina 1 no Teste # 3, com o intervalo de 200 ms (ou 5 Hz), é conforme segue:

Tabela 7 – Condições do Teste # 3

Variável	*Valor*
Local de execução do Publicador	Máquina 1
Local de execução do Subscritor	Máquina 1
Local de execução dos Serviços	Máquina 1
Parâmetro *Sleep*	200 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

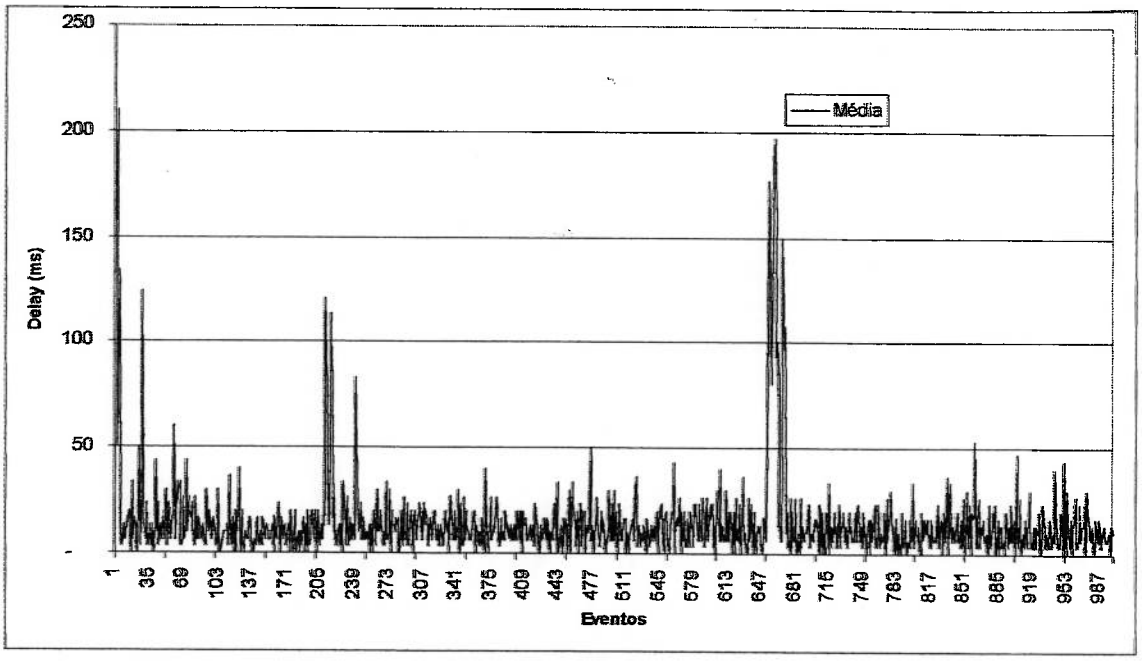

Figura 60 - Curva média de atraso para maq. 1, Sleep = 200, K = 1000, GUI tipo 1 (Teste # 3)

Já o resultado obtido para o intervalo de 1s entre as transmissões de eventos (Teste # 4), foi conforme o apresentado no gráfico da figura 61.

Tabela 8 – Condições do Teste # 4

Variável	*Valor*
Local de execução do Publicador	Máquina 1
Local de execução do Subscritor	Máquina 1
Local de execução dos Serviços	Máquina 1
Parâmetro *Sleep*	1000 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

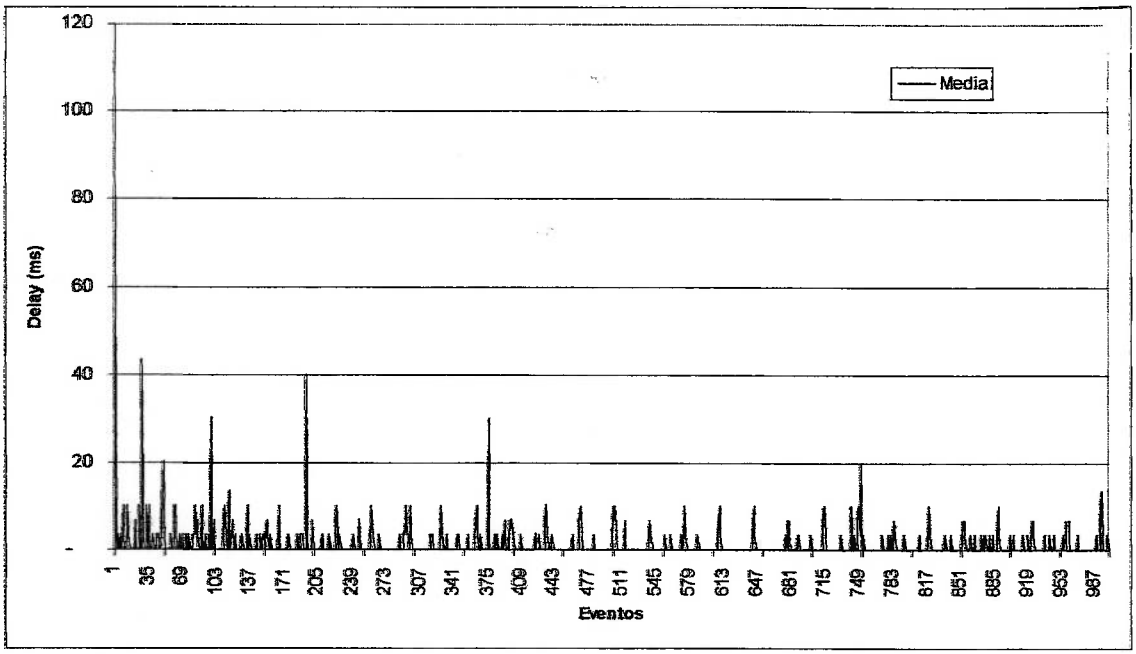

Figura 61 - Curva média de atraso para maq. 1, Sleep = 1000, K = 1000, GUI tipo 1 (Teste # 4)

Pode-se observar que o valor do atraso na recepção dos eventos pelo elemento subscritor está diretamente relacionado com o intervalo de tempo definido entre as publicações de dados pelo elemento publicador, no caso do protótipo, o elemento sensor.

Conforme pode ser visto na tabela abaixo, quanto maior o tempo entre as publicações de eventos, menor o atraso observado, tanto em termos de valor de pico quanto em termos de valor médio.

Tabela 9 - Comparação dos atrasos na máquina 1, no sistema operacional Windows NT

Teste	*Sleep*	Atraso Máximo	Atraso Médio
Teste # 3	200 ms	531 ms	15,2 ms
Teste # 2	500 ms	421 ms	1,7 ms
Teste # 4	1000 ms	110 ms	1,3 ms

Observando-se os dados individuais coletados nos diversos testes efetuados, verifica-se que um dos pontos de pico no *delay* sempre está nos primeiros eventos transmitidos, em função do estabelecimento da conexão entre o elemento subscritor e o canal de eventos, sendo que em todos os casos o atraso para o primeiro evento sempre oscila entre 90 e 160 ms.

Desconsiderando-se essa região, que será identificada no texto como região de transiente, obtém-se que o atraso máximo verificado para a condição de intervalo de 1s entre transmissões de eventos é de 60 ms e não de 110 ms como obtido quando tomados todos os dados. Já o valor médio não sofre uma diferença significativa.

4.2.3 Testes com variação do intervalo de transmissão e ambiente de execução (Testes de 5 a 11)

O mesmo teste, com o valor do intervalo entre eventos em 500 ms foi realizado na máquina 2, estando também os serviços de Notificação e de Nomes sendo executados nessa máquina (Teste # 5).

Tabela 10 – Condições do Teste # 5

Variável	*Valor*
Local de execução do Publicador	Máquina 2
Local de execução do Subscritor	Máquina 2
Local de execução dos Serviços	Máquina 2
Parâmetro *Sleep*	500 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

O resultado obtido, neste segundo caso, foi diferente do obtido no Teste # 2, conforme pode ser verificado no gráfico a seguir.

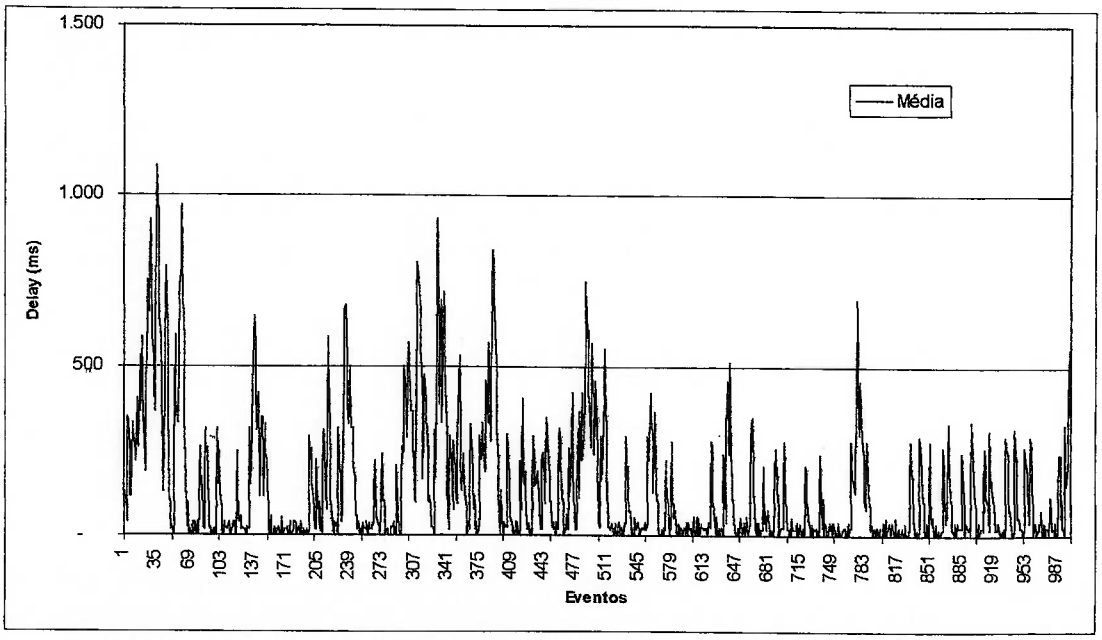

Figura 62 - Curva média de atraso para a maq. 2, Sleep = 500, K = 1000, GUI tipo 1 (Teste # 5)

Como pode-se perceber, a mesma condição de execução na máquina 2 não ofereceu os mesmos resultados observados na máquina 1, tendo apresentado mais oscilação nos valores de atraso e uma média do tempo de *delay* de aproximadamente 140 ms, contra uma média de delay de aproximadamente 1,7 ms encontrada no teste análogo na máquina 1. Embora a máquina 1 disponha de mais recursos de memória RAM, a velocidade do processador da máquina 2 é cerca de 67% maior, o que torna interessante a diferença de 80 vezes entre os atrasos médios obtidos nas duas máquinas.

A fim de se avaliar a eventual influência do sistema operacional Windows NT no comportamento apresentado no teste realizado na máquina 1, a mesma avaliação foi refeita nessa máquina, mas agora com o sistema operacional Windows 98 (Teste # 6). Nos testes em que a máquina 1 operou com o sistema Windows 98, ela está identificada como máquina 1B, para maior clareza na interpretação dos dados.

Tabela 11 – Condições do Teste # 6

Variável	*Valor*
Local de execução do Publicador	Máquina 1B
Local de execução do Subscritor	Máquina 1B
Local de execução dos Serviços	Máquina 1B
Parâmetro *Sleep*	500 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

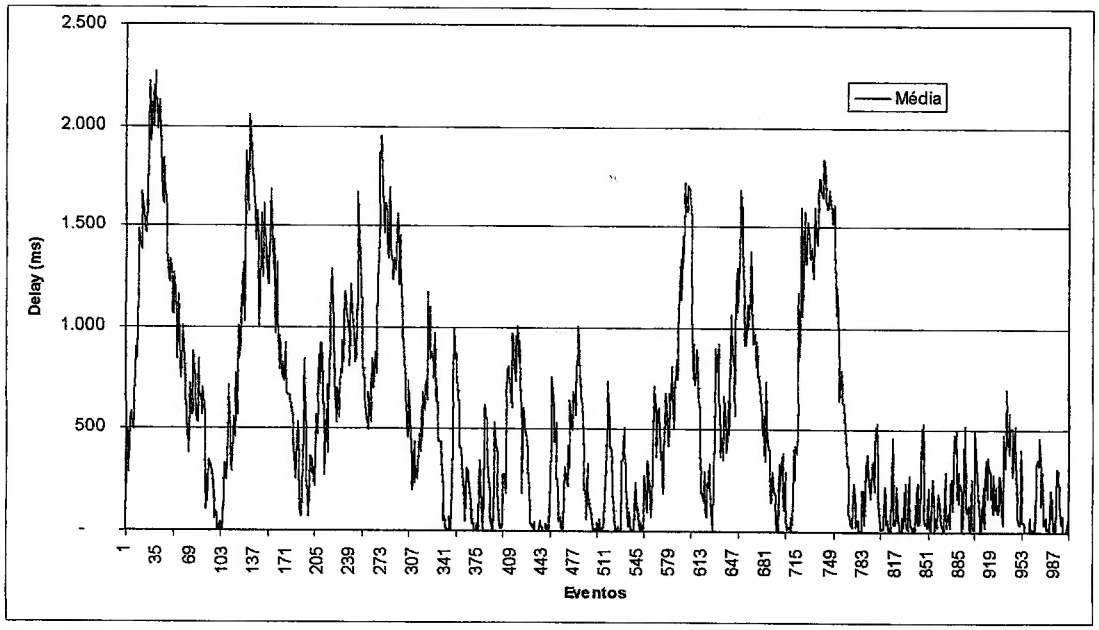

Figura 63 - Curva média de atraso para maq. 1B, Sleep = 500, K = 1000, GUI tipo 1 (Teste # 6)

Conforme pode ser visto no gráfico acima, o perfil da curva de atrasos obtido na máquina 1 com o sistema operacional Windows 98 não apresenta o comportamento observado na curva gerada pela mesma máquina quando trabalhando com o sistema operacional Windows NT. Entretanto, mostrou-se próximo do perfil obtido na máquina 2. Desse modo, fica clara a influência do sistema operacional na eficiência da transmissão e no processamento dos eventos.

O valor médio obtido para o atraso no Teste # 6 foi de aproximadamente 565 ms. O pico de atraso obtido durante o teste foi de 5,16 s.

Comparando-se, através da tabela abaixo, os resultados obtidos nos testes realizados com a máquina 1, a máquina 2 e a máquina 1B,confirma-se a influência do sistema operacional utilizado no desempenho da transmissão de eventos pelo serviço de notificação. O mesmo sistema, executado no mesmo *hardware*, mas em sistemas operacionais diferentes pode apresentar diferenças de desempenho da ordem de 10 vezes, quando se leva em consideração a atraso máximo. Quando se compara o atraso médio, a diferença pode chegar a mais de 300 vezes.

Tabela 12 - Comparação dos atrasos nas máquinas 1, 1B e 2 para Sleep = 500 ms

Teste	Ambiente	Atraso Máximo	Atraso Médio
Teste # 2	Máquina 1	421 ms	1,7 ms
Teste # 5	Máquina 2	2.200 ms	140 ms
Teste # 6	Máquina 1B	5.160 ms	565 ms

Desse modo, pode-se afirmar que a seleção e a correta configuração do ambiente de sistema operacional em que se executa a solução apresenta uma influência tão grande, ou até mesmo maior que a influência do *hardware*.

A fim de se validar essa constatação, foram realizados ainda os mesmos testes com intervalos entre eventos em 200 ms e 1000 ms nas máquinas 1B e 2.

Para a máquina 2, os resultados obtidos foram conforme segue:

Tabela 13 – Condições dos Teste # 7 e # 8

Variável	*Valor*
Local de execução do Publicador	Máquina 2
Local de execução do Subscritor	Máquina 2
Local de execução dos Serviços	Máquina 2
Parâmetro *Sleep*	200 ms (Teste # 7) 1000 ms (Teste # 8)
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

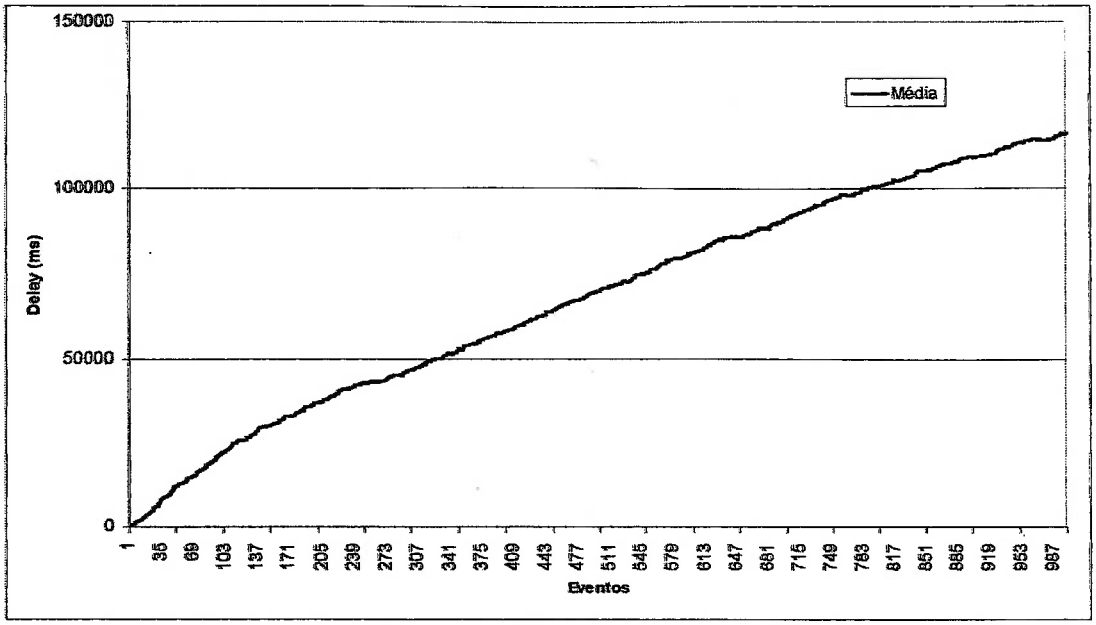

Figura 64 - Curva média de atraso para maq. 2, Sleep = 200, K = 1000, GUI tipo 1 (Teste # 7)

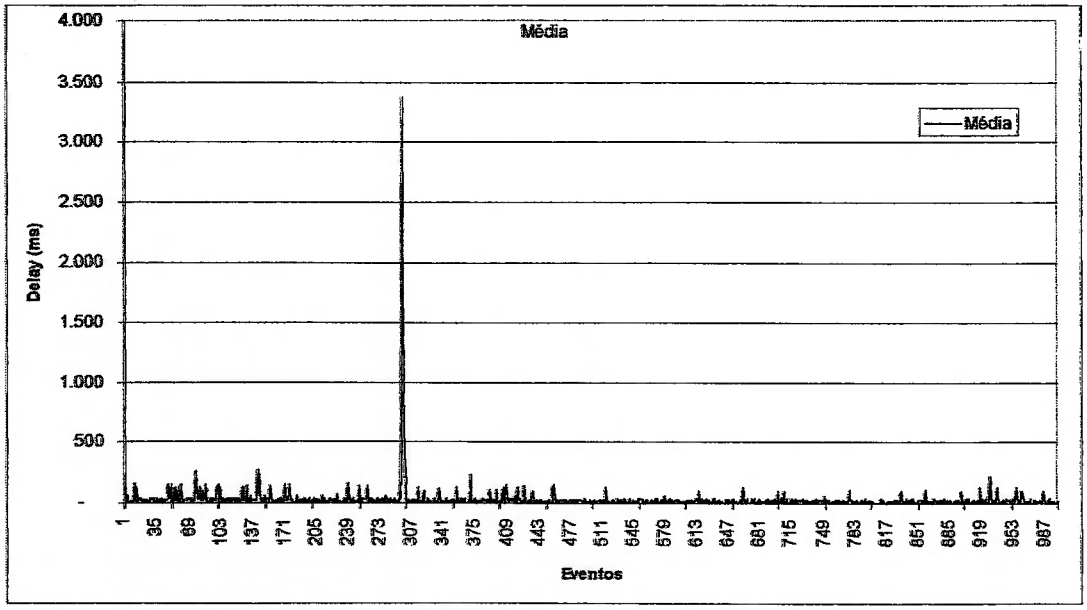

Figura 65 - Curva média de atraso para maq. 2, Sleep = 1000, K = 1000, GUI tipo 1 (Teste # 8)

Já para a máquina 1B, os resultados obtidos foram os apresentados nos dois gráficos a seguir:

Tabela 14 – Condições dos Teste # 9 e # 10

Variável	*Valor*
Local de execução do Publicador	Máquina 1B
Local de execução do Subscritor	Máquina 1B
Local de execução dos Serviços	Máquina 1B
Parâmetro *Sleep*	200 ms (Teste # 9) 1000 ms (Teste # 10)
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

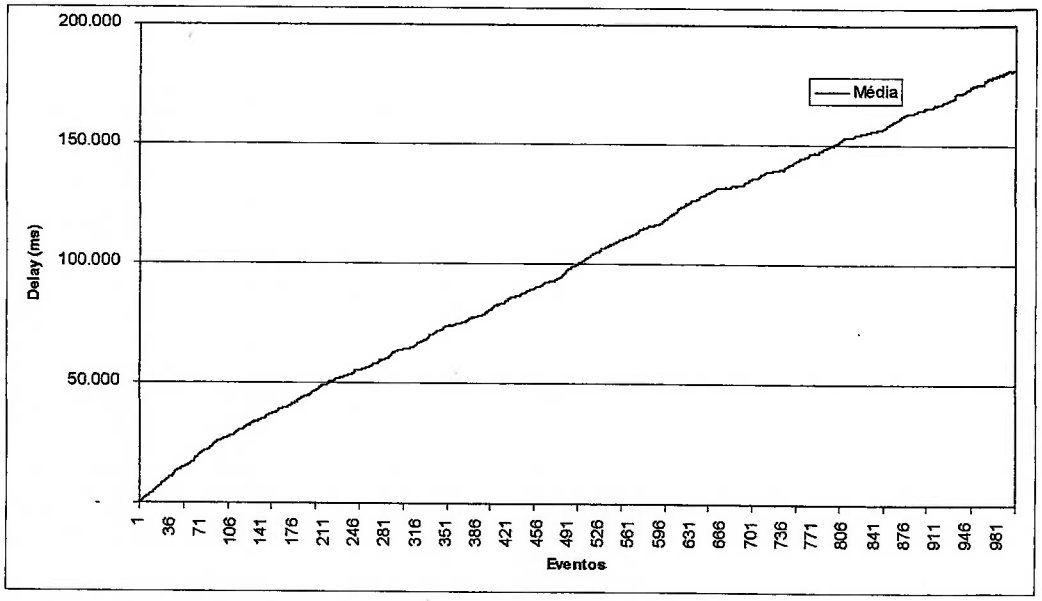

Figura 66 - Curva média de atraso máquina 1B, Sleep = 200, K = 1000, GUI tipo 1 (Teste # 9)

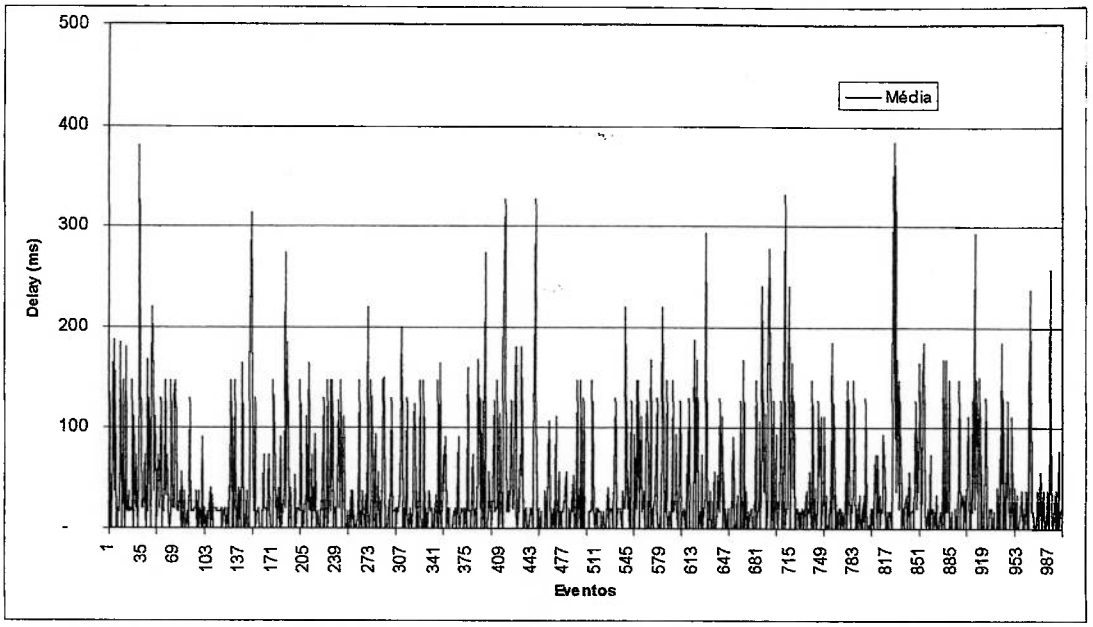

Figura 67 - Curva média de atraso maq. 1B, Sleep = 1000, K = 1000, GUI tipo 1 (Teste # 10)

Como se pode observar, com exceção do pico de atraso apresentado pela máquina 2 no teste com atraso em 1000 ms, o aspecto geral das curvas é semelhante entre si, mas difere do apresentado pelas curvas geradas pela máquina 1.

Um resumo geral dos índices observados nas três diferentes combinações de ambiente estudadas pode ser visto na tabela a seguir.

Tabela 15 - Comparativo dos atrasos nas máquinas 1, 1B e 2

	Sleep = 200 ms		Sleep = 500 ms		Sleep = 1000 ms	
Ambiente	**Atraso Máximo**	**Atraso Médio**	**Atraso Máximo**	**Atraso Médio**	**Atraso Máximo**	**Atraso Médio**
Máquina 1	Teste # 3		Teste # 2		Teste # 4	
	531 ms	15,2 ms	421 ms	1,7 ms	110 ms	1,3 ms
Máquina 1B	Teste # 9		Teste # 6		Teste # 10	
	220.970 ms	98.200 ms	5.160 ms	565 ms	383 ms	37,6 ms
Máquina 2	Teste # 7		Teste # 5		Teste # 8	
	127.100 ms	67.600 ms	2.200 ms	140 ms	6.750 ms	27 ms

Observa-se, nas configurações que utilizam o sistema operacional Windows 98, um comportamento particularmente interessante quando o intervalo entre as publicações de eventos é de 200 ms. Nesses casos, o valor do atraso tende a crescer continuamente, indicando que a taxa de recepção de eventos pelo elemento subscritor é menor que a taxa de introdução de eventos no canal pelo elemento publicador (que corresponde a 200 ms). Ou seja, há um acúmulo progressivo de atrasos, demonstrando claramente que o sistema como um todo não funcionaria de maneira adequada para taxas de leitura do elemento sensor em 200 ms ou abaixo deste valor.

Já para as taxas de leitura de 500 ms e acima, o atraso do sistema tende a se manter relativamente estável, oscilando entre patamares bem definidos e viabilizando sua utilização, desde que as especificações do sistema de controle que irá utilizá-lo aceitem as margens de atraso obtidas para cada caso.

O comportamento de acúmulo progressivo de atrasos também pode ser observado no ambiente da máquina 1, mas ele só se faz presente com valores de intervalo entre os eventos da ordem de 50 ms para baixo. Um exemplo deste comportamento pode ser observado no gráfico abaixo (Teste # 11).

Tabela 16 – Condições do Teste # 11

Variável	*Valor*
Local de execução do Publicador	Máquina 1
Local de execução do Subscritor	Máquina 1
Local de execução dos Serviços	Máquina 1
Parâmetro *Sleep*	50 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 1

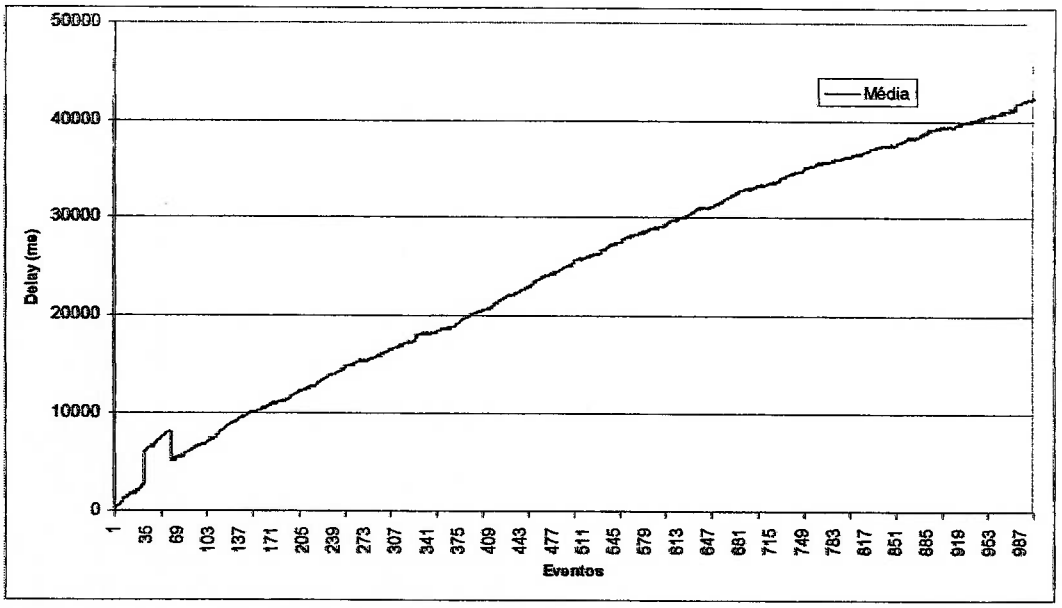

Figura 68 - Curva média de atraso máquina 1, Sleep = 50, K = 1000, GUI tipo 1 (Teste # 11)

Verifica-se, portanto, que uma análise criteriosa dos requisitos de taxa de transferência entre os elementos publicadores e subscritores faz-se necessária para que se possa selecionar e especificar o ambiente de execução (*hardware* e sistema operacional) dos componentes de maneira adequada.

4.2.4 Testes com interface gráfica mínima (Testes 12 e 13)

A fim de se avaliar o efeito do último dos três itens de influência no desempenho da recepção de eventos pelo elemento subscritor, ou seja, a complexidade da implementação do elemento subscritor, realizaram-se alguns testes com a GUI em sua forma 2 já descrita, com o mínimo possível de elementos gráficos. Para esses testes, foi escolhido o intervalo entre eventos de 200 ms, e os resultados obtidos estão apresentados nos gráficos abaixo:

Tabela 17 – Condições do Teste # 12

Variável	*Valor*
Local de execução do Publicador	Máquina 1
Local de execução do Subscritor	Máquina 1
Local de execução dos Serviços	Máquina 1
Parâmetro *Sleep*	200 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 2

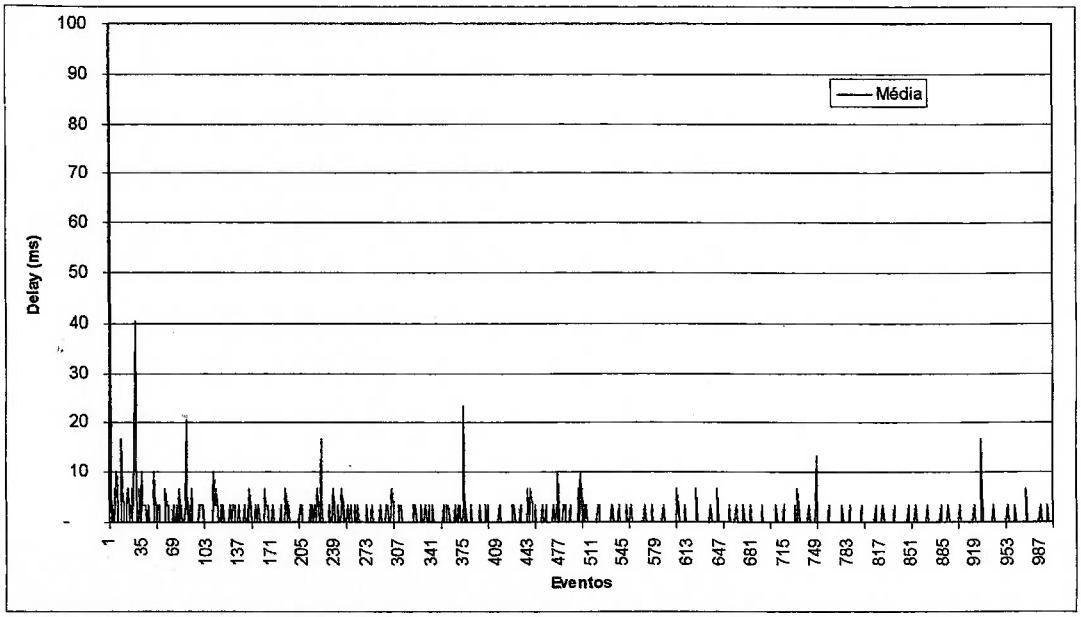

Figura 69 - Curva média de atraso maq. 1, Sleep = 200, K = 1000, GUI tipo 2 (Teste # 12)

Tabela 18 – Condições do Teste # 13

Variável	*Valor*
Local de execução do Publicador	Máquina 2
Local de execução do Subscritor	Máquina 2
Local de execução dos Serviços	Máquina 2
Parâmetro *Sleep*	200 ms
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 2

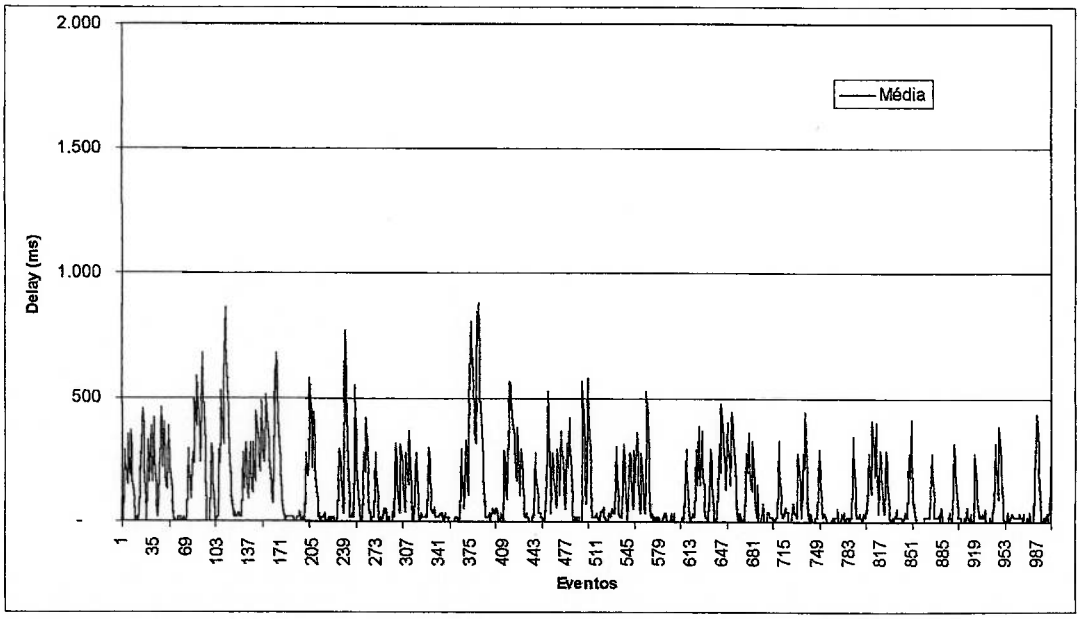

Figura 70 - Curva média de atraso maq. 2, Sleep = 200, K = 1000 e GUI tipo 2 (Teste # 13)

Um comparativo dos índices obtidos nesse teste pode ser verificado na tabela abaixo:

Tabela 19 - Comparação dos atrasos nos testes 3, 7, 12 e 13

Sleep	Ambiente	*Teste*	*GUI*	Atraso Máximo	Atraso Médio
200 ms	Máquina 1	Teste # 12	Tipo 2	170 ms	1 ms
		Teste # 3	Tipo 1	531 ms	15,2 ms
	Máquina 2	Teste # 13	Tipo 2	1760 ms	128 ms
		Teste # 7	Tipo 1	127.100 ms	67.600 ms

Deste modo, pode-se observar que a redução da complexidade da implementação do componente subscritor (GUISub) influi de maneira significativa nos valores de atraso, sendo que uma implementação que exija um mínimo de recursos da máquina pode resultar em valores de atraso médio iguais ou inferiores a 1 ms.

Fica demonstrado, desse modo, a influência dos 3 fatores principais de influência no desempenho da arquitetura de sistema proposta: o fator ambiente (*hardware* e sistema operacional), o fator intervalo entre a publicação de eventos (taxa de amostragem dos sensores) e o fator complexidade da implementação do elemento subscritor.

4.2.5 Testes com interface gráfica mínima e sem janelas DOS ativas (Testes de 14 a 22)

Em função da significativa melhora nos resultados obtidos com uma interface gráfica minimalista na janela do elemento subscritor (GUI), optou-se pela realização de um teste final, em que, além da remoção dos elementos gráficos mais sofisticados da interface, efetuou-se também a remoção das janelas de texto DOS utilizadas para a monitoração do status do protótipo. Essas janelas haviam sido usadas em todos os testes anteriores, como uma forma de se acompanhar continuamente a execução dos testes.

Os testes com as janelas desativadas foram realizados em todas as máquinas e configurações usadas nos testes anteriores, com intervalos variados entre as publicações de eventos, e os resultados obtidos são conforme segue.

A primeira bateria de testes foi realizada com a configuração Máquina 1, utilizando-se as variações apresentadas na tabela abaixo.

Tabela 20 – Condições dos Teste # 14, 15 e 16

Variável	*Valor*
Local de execução do Publicador	Máquina 1
Local de execução do Subscritor	Máquina 1
Local de execução dos Serviços	Máquina 1
Parâmetro *Sleep*	200 ms (Teste # 14) 500 ms (Teste # 15) 1000 ms (Teste # 16)
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 2
Janela DOS de monitoração	Desativada

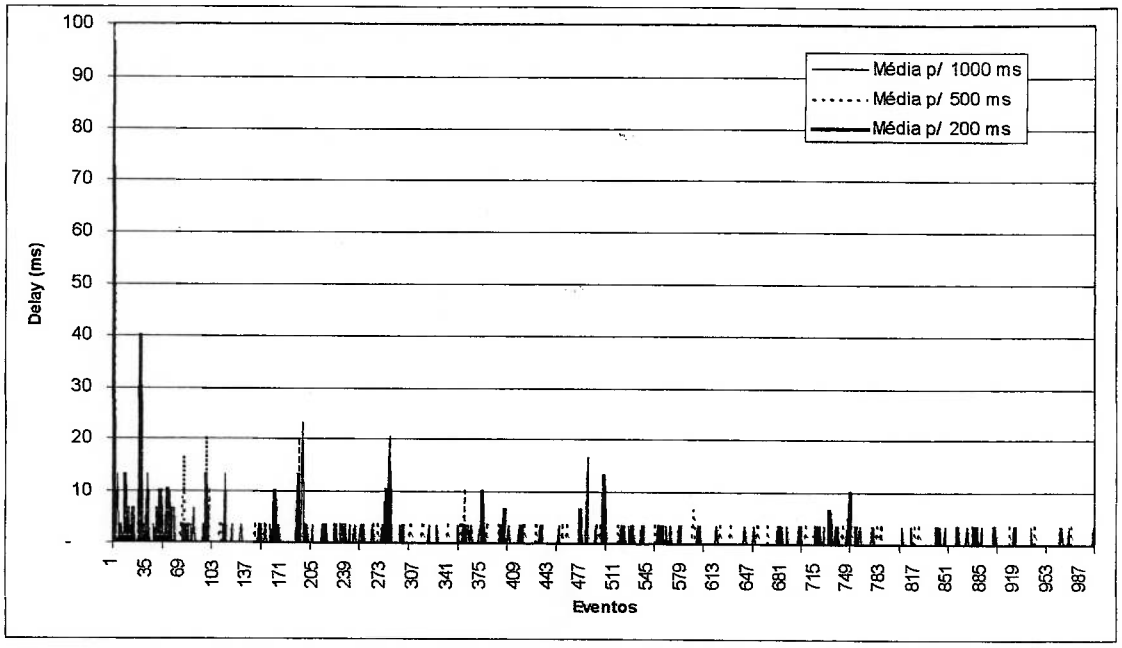

Figura 71 - Curvas médias de atrasos para a configuração máquina 1, com Sleep = 200, 500 e 1000 ms, com GUI tipo 2 e sem janela DOS para monitoração (Testes # 14, 15 e 16) .

Tabela 21 - Comparativo dos atrasos na máquina 1 (Testes # 14, 15 e 16).

	Teste # 14 Sleep = 200 ms		Teste # 15 Sleep = 500 ms		Teste # 16 Sleep = 1000 ms	
Ambiente	**Atraso Máximo**	**Atraso Médio**	**Atraso Máximo**	**Atraso Médio**	**Atraso Máximo**	**Atraso Médio**
Máquina 1	120 ms	0,48 ms	120 ms	0,50 ms	130 ms	0,55 ms

Após a conclusão dos testes com a configuração Máquina 1, realizou-se a mesma bateria de testes para a configuração Máquina 1B, ou seja, o mesmo hardware com o sistema operacional Windows 98, em vez de Windows NT.

A descrição das variações de parâmetros utilizadas e dos resultados obtidos está apresentada a seguir.

Tabela 22 – Condições dos Teste # 17, 18 e 19

Variável	*Valor*
Local de execução do Publicador	Máquina 1B
Local de execução do Subscritor	Máquina 1B
Local de execução dos Serviços	Máquina 1B
Parâmetro *Sleep*	200 ms (Teste # 17) 500 ms (Teste # 18) 1000 ms (Teste # 19)
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 2
Janela DOS de monitoração	Desativada

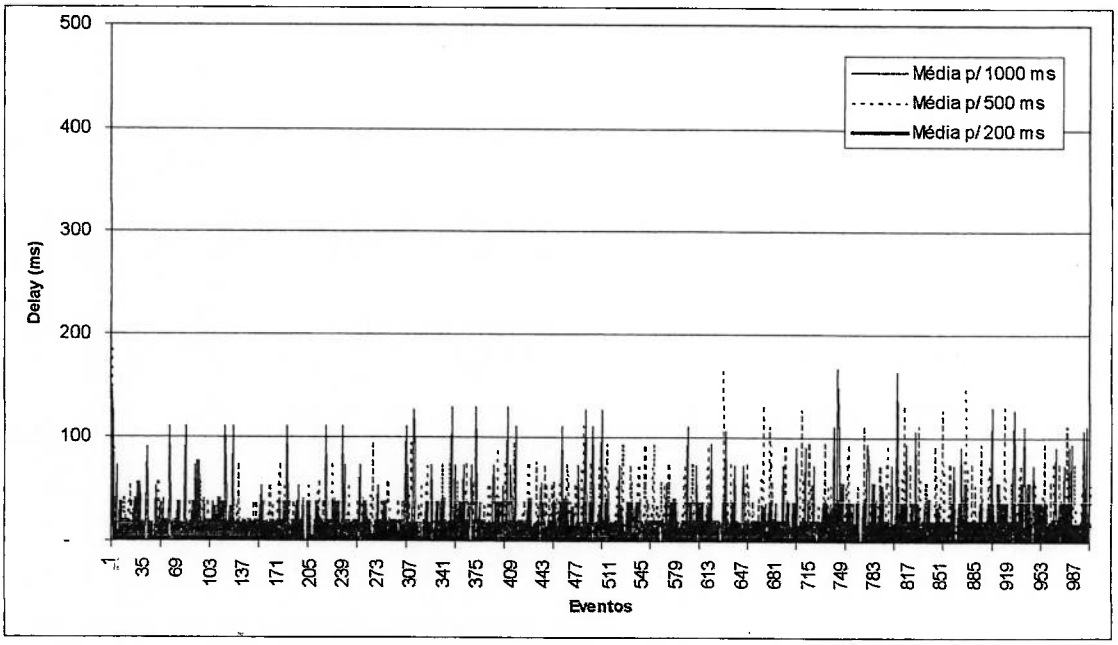

Figura 72 - Curvas médias de atrasos para a configuração máquina 1B, com Sleep = 200, 500 e 1000 ms, com GUI tipo 2 e sem janela DOS para monitoração (Testes 17, 18 e 19).

Tabela 23 - Comparativo dos atrasos na configuração máquina 1B

	Teste 17 Sleep = 200 ms		Teste 18 Sleep = 500 ms		Teste 19 Sleep = 1000 ms	
Ambiente	**Atraso Máximo**	**Atraso Médio**	**Atraso Máximo**	**Atraso Médio**	**Atraso Máximo**	**Atraso Médio**
Máquina 1B	220 ms	9,8 ms	280 ms	16,7 ms	390 ms	13,5 ms

Finalmente, realizaram-se também os mesmos teses com a configuração Máquina 2, conforme os parâmetros apresentados abaixo.

Tabela 24 – Condições dos Teste # 20, 21 e 22

Variável	*Valor*
Local de execução do Publicador	Máquina 2
Local de execução do Subscritor	Máquina 2
Local de execução dos Serviços	Máquina 2
Parâmetro *Sleep*	200 ms (Teste # 20) 500 ms (Teste # 21) 1000 ms (Teste # 22)
Parâmetro K	1000 eventos
Tipo de interface do Subscritor	GUI Tipo 2
Janela DOS de monitoração	Desativada

Os resultados obtidos estão apresentados no gráfico e na tabela a seguir.

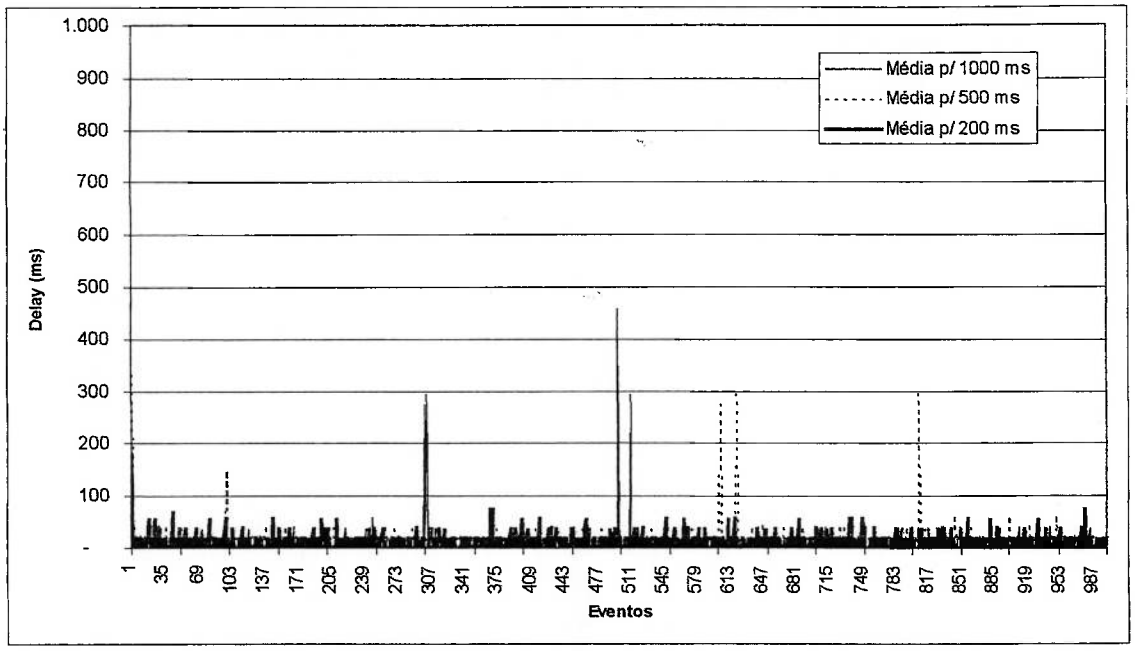

Figura 73 - Curvas médias de atrasos para a configuração máquina 2, com Sleep = 200, 500 e 1000 ms, com GUI tipo 2 e sem janela DOS para monitoração (Testes 20, 21 e 22).

Tabela 25 - Comparativo dos atrasos na máquina 2

	Teste 20 Sleep = 200 ms		Teste 21 Sleep = 500 ms		Teste 22 Sleep = 1000 ms	
Ambiente	**Atraso Máximo**	**Atraso Médio**	**Atraso Máximo**	**Atraso Médio**	**Atraso Máximo**	**Atraso Médio**
Máquina 2	220 ms	9,6 ms	880 ms	8,60 ms	1.370 ms	8,4 ms

A fim de se ter uma visão clara da melhoria adicional de desempenho obtida com a remoção da interface DOS de monitoramento podem-se observar os dados da tabela abaixo, em que se comparam os resultados obtidos nas execuções para as Máquinas 1 e 2, para um tempo de *Sleep* de 200 ms.

Tabela 26 - Comparação dos atrasos nos testes 12, 13, 14 e 20

Sleep	Ambiente	*Teste*	*Janela DOS*	Atraso Máximo	Atraso Médio
200 ms	Máquina 1	Teste # 12	Ativada	170 ms	1 ms
		Teste # 14	Desativada	120 ms	0,48 ms
	Máquina 2	Teste # 13	Ativada	1760 ms	128 ms
		Teste # 20	Desativada	220 ms	9,6 ms

Como se pode ver, há um ganho adicional de desempenho com a remoção da janela DOS de monitoramento do sistema, o que reforça a constatação de que uma redução da carga computacional requerida pelo elemento subscritor aumenta o desempenho do sistema.

4.2.6 Testes em ambiente de rede (Testes de 23 a 30)

A fim de se estudar o comportamento dos atrasos em configurações distribuídas, realizaram-se alguns testes com o sistema operando em rede, com o elemento publicador em uma máquina e o elemento subscritor em outra. Nesse tipo de teste os serviços de nomes e de eventos foram sempre executados na mesma máquina em que se encontrava o elemento publicador.

O primeiro desses testes foi realizado com os intervalos entre transmissões de dados variando de 50 ms a 1000 ms, com a máquina 1 rodando o componente publicador e os serviços de Notificação e Nomes e a máquina 2 apenas rodando o componente subscritor. Essa configuração será chamada de Máquina 1 / Máquina 2. Nesses testes em rede, que foram numerados de 23 a 26 a interface escolhida sempre foi a GUI tipo 2, e a janela DOS para monitoramento do sistema sempre esteve desligada, de modo que se pudesse obter o melhor desempenho possível no sistema.

Tendo-se concluído os testes na configuração Máquina 1 / Máquina 2, substitui-se a máquina 2 pela máquina 3, um microcomputador com processador Pentium II 300 MHz, com 64 MB RAM e rodando o sistema operacional Windows XP. Essa configuração, chamada de Máquina 1 / Máquina 3, também foi utilizada em testes com a freqüência de transmissão de dados entre 50 ms e 1000 ms e que foram numerados de 27 a 30.

Os resultados obtidos para o atraso nas duas configurações descritas acima estão tabulados a seguir:

Tabela 27 - Comparação dos atrasos nos testes de 23 a 30

Sleep	Ambiente	*Teste*	Atraso Máximo	Atraso Médio
50 ms	Máquina 1 / Máquina 2	Teste # 23	248 ms	36,4 ms
	Máquina 1 / Máquina 3	Teste # 27	112 ms	6,9 ms
200 ms	Máquina 1 / Máquina 2	Teste # 24	1.459 ms	33,7 ms
	Máquina 1 / Máquina 3	Teste # 28	109 ms	7,2 ms
500 ms	Máquina 1 / Máquina 2	Teste # 25	1.010 ms	35,1 ms
	Máquina 1 / Máquina 3	Teste # 29	145 ms	7,1 ms
1000 ms	Máquina 1 / Máquina 2	Teste # 26	551 ms	32,8 ms
	Máquina 1 / Máquina 3	Teste # 30	102 ms	6,5 ms

Um dos pontos que se pode verificar, quando se observam os resultados apresentados na tabela 27, é a confirmação de que o desempenho dos sistemas operacionais baseados em Windows NT, como é o caso do Windows XP Professional é claramente superior para a aplicação com sistemas MES do que a plataforma Windows 95. Mesmo a configuração de hardware da Máquina 2 (AMD K6-II, 500 MHz, com 128 MB RAM) sendo superior à da Máquina 3 (Pentium II, 300 MHz, com 64 MB RAM), o desempenho observado na máquina 3 foi da ordem de até 15 vezes superior em termos de atraso máximo.

A segunda observação que se pode fazer a respeito dos resultados da tabela 27, quando comparada aos resultados da tabela 25, é de que há um atraso adicional de aproximadamente 25 ms em média na configuração Máquina 1 / Máquina 2, quando comparada à configuração Máquina 2, em que todos os elementos do sistema estavam sendo executados em uma mesma máquina. Essa diferença deve-se ao tempo de transmissão do evento através da rede e indica que também a infra-estrutura disponível para a transmissão de dados entre os elementos publicadores e subscritores desempenha um papel importante no desempenho final do sistema.

5 Conclusões

Conforme o proposto na Introdução do presente trabalho, apresentou-se inicialmente uma série de conceitos necessários para a compreensão dos sistemas MES e de seu posicionamento dentro do estado atual dos sistemas corporativos de Tecnologia da Informação, para então, através do uso da arquitetura CORBA, propor-se um modelo básico de arquitetura a partir do qual as funcionalidades de coleta de dados e alocação de recursos dos MES podem ser implementadas.

A partir da arquitetura proposta, realizou-se o desenvolvimento de um protótipo funcional, que faz uso da linguagem de programação Java e da infra-estrutura CORBA implementada por um fornecedor do mercado, para validar-se a aplicabilidade dessa arquitetura à solução dos problemas envolvidos na aplicação de um sistema MES a um processo produtivo.

Com relação aos recursos necessários para a implementação do protótipo de sistema MES, a arquitetura CORBA mostrou-se suficiente, tendo-se feito uso especialmente dos serviços de Notificação e de Nomes para a criação do sistema. A aplicação dos serviços já implementados no pacote de mercado usado mostrou uma maneira eficiente de se economizar tempo no desenvolvimento da aplicação, uma vez que todo o código necessário para a localização dos componentes do sistema já se encontrava desenvolvido e pronto para ser aplicado de maneira transparente em uma instalação residente em uma única máquina ou em uma configuração distribuída. Uma vez que a procura e localização dos objetos publicadores pelos objetos subscritores através do Serviço de Nomes só é realizada no instante da carga dos objetos subscritores, o desempenho nessa etapa de localização não é um fator crítico para o desempenho global do sistema MES.

Por outro lado, o desempenho na transmissão de eventos através do Serviço de Notificação compõe um fator crítico para o desempenho global do sistema MES, definindo, em grande parte, a sua aplicabilidade a um determinado processo industrial. Sendo assim, realizou-se uma série de testes a fim de se avaliar o desempenho global

do protótipo implementado e identificarem-se os fatores de maior influência sobre esse desempenho.

Após a execução dos testes, os três pontos identificados como sendo os de principal influência no desempenho do sistema foram os seguintes:

- Ambiente de execução do sistema;
- Freqüência de transmissão de eventos;
- Complexidade do elemento subscritor.

Por ambiente de execução do sistema entenda-se o conjunto da configuração de *hardware* e sistema operacional utilizado tanto do lado do elemento publicador quanto do lado do elemento subscritor. Embora a influência do *hardware* (especialmente da quantidade de memória disponível na máquina e a velocidade de seu processador), seja direta sobre o desempenho de qualquer *software* que venha a ser executado na máquina, no caso do protótipo implementado o sistema operacional apresentou uma influência ainda mais significativa.

O primeiro ponto de influência do sistema operacional sobre o desempenho global do sistema foi verificado ao se realizarem medições do tempo decorrido entre a publicação e o recebimento de um evento em uma mesma máquina, em um momento com o sistema operacional Windows NT e em outro com o sistema Windows 98.

O mesmo *hardware*, com a mesma quantidade de memória física, chegou a apresentar uma diferença de desempenho da ordem de 300 vezes (no caso mais crítico, quando comparados os testes 2 e 6), em favor do sistema Windows NT.

Além disso, em algumas situações, o sistema operacional definiu a aplicabilidade ou não do protótipo a uma aplicação MES real. Por exemplo, no caso do teste 3, em que se usava o sistema operacional Windows 98, o resultado obtido invalidou a aplicabilidade do protótipo, já que o desempenho tornava-se progressivamente pior ao longo do tempo, em função de um acúmulo gradual dos atrasos entre a publicação e o recebimento dos eventos. Já no caso do teste 9, em que se usou a mesma máquina, com os mesmos parâmetros, mas com o sistema Windows NT, o resultado obtido foi

mais satisfatório, não se observando o acúmulo gradual de atrasos que inviabiliza o sistema.

O segundo ponto chave de influência sobre o desempenho global do sistema, ou seja, a freqüência de transmissão dos eventos pelo elemento publicador, representa um papel essencial na especificação técnica dos elementos do sistema final e na verificação da aplicabilidade de uma dada implementação MES a um processo produtivo.

Por exemplo, em uma aplicação em que a freqüência de medição é de 1 Hz (apenas um evento a cada segundo) ou inferior, praticamente qualquer uma das configurações testadas ao longo do presente trabalho seriam adequadas, desde que o tempo máximo de resposta do sistema estivesse compatível com os resultados verificados nos testes. Entretanto, para aplicações mais críticas, com freqüências de medição maiores, algumas configurações do sistema tornam-se inadequadas, já que a partir de uma certa freqüência limite passa-se a verificar o efeito de acúmulo de atrasos. Esse efeito pode ser verificado comparando-se os gráficos dos testes 2, 3, 4 e 11.

Desse modo, a freqüência de leitura de um determinado sensor torna-se um parâmetro essencial à correta especificação do ambiente e da configuração com que a parcela do MES responsável por sua supervisão será executada.

Finalmente, o terceiro ponto chave de influência no desempenho do protótipo foi identificado como sendo o grau de complexidade da implementação do elemento subscritor. De uma maneira geral, quanto maior a carga computacional demandada para o processamento do sinal recebido através do canal de notificação, piores foram os índices de atraso máximo e médio observados nos testes de avaliação de desempenho do protótipo implementado.

Apesar de este terceiro aspecto poder ser contornado através de estratégias como a de apenas exibirem-se na interface gráfica alguns dos eventos recebidos, como por exemplo apenas os alarmes verificados e uma média geral das últimas leituras, uma análise detalhada de todo o tipo de processamento que deverá ser realizado pelos elementos subscritores deve ser efetuada antes de se fecharem as especificações de

hardware e ambiente para um dado MES. Como já observado em [MISHRA, 2001], o projeto cuidadoso dos elementos do sistema colabora substancialmente para a melhora do desempenho.

Uma vez que esses três aspectos chaves sejam observados e o sistema MES seja configurado de modo a respeitar as restrições por eles impostas, pode-se afirmar que a aplicabilidade da arquitetura proposta a um processo industrial passa a ser uma realidade.

Como já havia sido apresentado ao longo do texto, não se tencionou no presente trabalho apresentar uma especificação completa para um sistema MES, mas sim validar-se a aplicabilidade da arquitetura CORBA à sua implementação satisfatória. Desse modo, através do estudo dos resultados de desempenho do protótipo implementado a partir do modelo básico proposto, chega-se à conclusão de que a aplicação da arquitetura CORBA é realmente viável em aplicações MES, desde que sejam observados os requisitos de desempenho definidos pelo processo industrial a cujo controle se destina o MES e se configure a aplicação de forma a atendê-los adequadamente.

Finalmente, cumpre observar que todos os resultados aqui apresentados e as análises realizadas foram obtidos com base na utilização de uma implementação CORBA de um único fornecedor. Uma vez que a OMG apenas especifica as interfaces para os diversos componentes e serviços que fazem parte da arquitetura CORBA, pode ocorrer que resultados melhores (ou piores) sejam verificados com uso de implementações de outros fornecedores, mas esse tipo de comparativo já extrapola o objetivo central deste trabalho.

O mesmo se aplica aos sistemas operacionais utilizados para os procedimentos de teste do protótipo implementado. Apesar de haverem sido realizados testes apenas em ambientes derivados da tecnologia Windows (Windows 98, NT e XP), a linguagem Java e a arquitetura CORBA são independentes de plataforma, de modo que o sistema apresentado pode ser executado também em outros sistemas operacionais ou mesmo em um ambiente híbrido, podendo haver variação nos resultados observados.

6 Bibliografia

[ATHERTON, 1998] - Atherton, Robert – Java Object Technology Can Be Next Process Control Wave - Control Engineering - Outubro - 1998

[BERGE, 2002] – Berge, Jonas – OPC DX is the Software Gateway – Worldbus Journal – Abril, 2002

[BOYER, 2002] – Boyer, Stuart – The long arm of SCADA – InTech – Março, 2002

[BROWN, 2000] – Brown, Laura - Integration Modeling Techniques for Enterprise Resource Planning (An ERP Case Study) – Sams Publishing – 2000

[CHENG, 2000] – Cheng, Fan-Tien et al. – Modeling and Analysis of Equipment Managers in Manufacturing Execution Systems for Semiconductor Packaging – IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics - Vol. 30 - Número 5 – Outubro, 2000

[CHUNG, 1998] - Chung, P. Emerald et al. - DCOM x CORBA Side by Side, Step by Step and Layer by Layer - Control Engineering website - www.controleng.com - 1998

[CONSIDINE, 1993] – Considine, Douglas M. – Process/Industrial Instruments & Control Handbook – 4ª Edição – McGraw Hill – 1993

[DOBRIVOJE, 1990] - Popovic, Dobrivoje; Bhatkar, Vijay – Distributed Computer Control for Industrial Automation - Marcel Dekker Inc. - EUA - 1990

[EXTON, 1997] – Exton, Chris et al. – Comparisons between CORBA IDL & COM/DCOM MIDL: Interfaces for Distributed Computing – IEEE Computer Society – Technology of Object-Oriented Language and Systems – Melbourne, Austrália – Novembro, 1997

[FLASH, 2002] – ... - The .NET Framework – Flashdadee – Abril, 2002 – Flashdadee website: http://www.flashdaddee.com/Books-Technical/InsideCsharp/32ch02c.htm

[GAMA, 1995] - Gamma, Erich et al. – Design Patterns – Elements of Reusable Object-Oriented Software – Addison Wesley Longman, Inc. – 1998

[HAWKER, 1997] - Hawker, Scott – SEMATECH CIM Framework - Sematech website: www.sematech.org - Julho, 1997

[HIGHLANDER, 1998] – Highlander Communications – CORBA in the Embedded Systems Context – Highlander Communications, L.C. – 1998

[HOSKE, 1998] - Hoske, Mark T. – Objects Make Software Behave Like Hardware - Control Engineering - Outubro - 1998

[KAMAL, 1998] - Kamal, S. Zafar - Integrated Enterprise proves key to flexible manufacturing - InTech - Vol. 45, Número 7 - Julho 1998

[KOSAKAYA, 2002] – Kosakaya, Juichi et al. – Distributed Supervisory System with Cooperative Multi-Agent FEP – 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02) – Viena, Áustria – Julho, 2002

[KOSHIKEN, 2000] – Koshike, Kari – Integration of Automation Systems to Business Process – New Technologies – Palestra ministrada na Helsinki University of Technology – Novembro, 2000

[KUO, 2000] – Kuo, Chung-Hsien e Huang, Han-Pang – Failure Modeling and Process Monitoring for Flexible Manufacturing Systems Using Colored Timed Petri Nets – IEEE Transactions on Robotics and Automation - Vol. 16 - Número 3 – Junho, 2000

[LANGE, 1998] - Lange, Stephen T. - What's all this about Object Technology? - Control Engineering - Maio 1998

[LEMAY, 1999] - Lemay, Laura et al. - Aprenda em 21 Dias Java 1.2 - Editora Campus – 1999

[LOCKE, 1999] – Locke, C. Douglas e North, Thomas A. – A Real-Time, Fault Tolerant CORBA Implementation: A Case Study – Fourth International Workshop on Object-Oriented Real-Time Dependable Systems – Janeiro, 1999

[LONG, 1999] – Long, Earl et al. – Application of Model-Integrated Computing in Manufacturing Execution Systems – IEEE Conference and Workshop on Engineering of Computer-Based Systems – Nashville, EUA - Março, 1999

[MELLO, 2002a] - Mello, Adrian - ERP Fundamentals - ZDNet - Fevereiro, 2002 - ZDNet website: http://zdnet.search.com/search?cat=279&tag=st.ne.sr.srch.zdnet&q=%22IDC%22

[MESA, 1997a] - ... - The Benefits of MES: A Report From The Field - MESA International White Paper # 1 - Maio, 1997

[MESA, 1997b] - ... - MES Funcionalities & MRP to MES Dataflow Possibilities - MESA International White Paper # 2 - Março, 1997

[MESA, 1997c] - ... - MES Explained: A High Level Vision - MESA International White Paper # 6 - Setembro, 1997

[MESA, 2000a] - ... – Control Definition & MES to Controls Data Flow Possibilities - MESA International White Paper # 3 – Fevereiro, 2000

[MESA, 2001a] - ... - Definition of MES - MESA Website - www.mesa.org/html/overview.html

[MICROSOFT, 2002] - ... - What's Different? Innovating a Third Generation of Computing – Microsoft – Abril, 2002 – Microsoft website: http://www.microsoft.com/net/defined/netchange.asp

[MISHRA, 2001] – Mishra, Shivakant et al. – On Group Communication Support in CORBA – IEEE Transactions on Parallel and Distributed Systems - Vol. 12 - Número 2 – Fevereiro, 2001

[MORLEY, 1998] – Morley, Richard – The Future of Factory Automation – Evolving Enterprise Newsletter – 1998

[OMG, 1995a] - Soley, Richard Mark et al. - Object Management Architecture Guide - 3ª edição - John Wiley & Sons, Inc. - Junho, 1995

[OMG, 1995b] - ... - Common Facilities Architecutre - OMG Document 98-07-10 - Revision 4.0: Novembro, 1995

[OMG, 1996a] – OMG Manufacturing Special Interest Group - Manufacturing Enterprise Systems, A White Paper - OMG Document Number mfg/96-01-02 – Fevereiro, 1996

[OMG, 1997a] - ... - Manufacturing Domain Task Force, RFI-3, Manufacturing Execution Systems (MES) - OMG Document Number mfg/97-11-01, - Novembro, 1997

[OMG, 1998a] - ... - The Common Object Request Broker: Architecture and Specification - OMG Document 98-07-01 - Revision 2.2: Fevereiro, 1998

[OMG, 1998b] - ... - CORBAservices: Common Object Services Specification - OMG Document 98-12-09 - Dezembro, 1998

[OMG, 2000a] - ... - Notification Service Specification - OMG Document 00-06-20 - Junho, 2000

[OMG, 2001a] - ... – DAIS - Data Acquisition for Industrial Systems Specification – Draft Adopted Specification – OMG Document 01-07-03 – Julho, 2001

[OOC, 2000a] - ... – Orbacus Notify – Versão 1.0.1 – Object Oriented Concepts Inc. - 2000

[OPC, 1998a] – ... – OPC Common Definitions and Interfaces – Versão 1.0 – OPC Foundation Document – Outubro, 1998

[OPC, 1998b] – ... – OPC Overview – Versão 1.0 – OPC Foundation Document – Outubro, 1998

[ORFALI, 1998] - Orfali, Robert et al. - Client/Server Programming with Java and CORBA - 2ª edição - John Wiley & Sons, Inc. - 1998

[PENNER, 2002] – Penner, Robin R. e Steinmetz, Erick S. – Model-Based Automation of the Design of User Interfaces to Digital Control Systems – IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans - Vol. 32 - Número 1 – Janeiro, 2002

[ROSEMBERG, 1998] - Rosemberg, Jeremy - Teach Yourself CORBA in 14 days - 1ª edição - Sams Publishing - 1998

[ROY, 2002] – Roy, Chinmoy e Johnson, Leonard – Positive Yield - InTech - Vol. 49, Número 4 – Abril, 2002

[RUMBAUGH, 1994] - Rumbaugh, James - Modelagem e Projetos Baseados em Objetos - Editora Campus - Rio de Janeiro - 1994

[SCHMIDT, 2000] – Schmidt, Douglas C. e Kuhns, Fred – An Overview of the Real-Time CORBA Specification – IEEE Computer Society Magazine – Vol. 33, Número 6 - Junho, 2000

[SCHMULLER, 1999] - Teach Yourself UML in 24 Hours - 1ª edição - Sams Publishing - 1999

[SEMATECH, 1995] - ... - Implementation Handbook for the Computer Integrated Manufacturing (CIM) Application Framework Specification 1.2 - SEMATECH - Technology Transfer # 95092971^{A}-ENG - Setembro, 1995

[SEMATECH, 1998] - ... - Computer Integrated Manufacturing (CIM) Framework Specification (version 2.0) - SEMATECH - Technology Transfer # 93061697 J-ENG - Janeiro, 1998

[STROTHMAN, 2002a] – Strothman, Jim – OPC Spreads Wings – Worldbus Journal – Junho, 2002

[TALLMAN, 1998] – Tallman, Owen e Kain, J. Bradford – COM versus CORBA: A Decision Framework – Distributed Computing - Setembro, 1998

[TOMOMICHI, 1997] – Seki, Tomomichi et al. – Decentralized Autonomous Object-Oriented EMS/SCADA System – 3rd International Symposium on Autonomous Decentralized Systems (ISADS'97) – Berlim, Alemanha – Abril, 1997

[VIRDHAGRISWARAN, 1995] – Virdhagriswaran, S. Et al. – Manufacturing Collaboration Resource Discovery System (McRDS) – Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE'95) – Berkeley Springs, EUA - Abril, 1995

[WARREN, 1990] - Shrensker, Warren L. – CIM - Computer Integrated Manufacturing - A Working Definition - SME Blue Book Series - CASA/SME – 1990

[WASKIEWICZ, 1997] - Waskiewicz, Fred – An Object-Oriented Framework for CIM Applications - Sematech website: www.sematech.org - Julho, 1997

[WOHLEVER, 1999] – Wohlever, Steven et al. – Building Adaptable, Real-Time Command and Control Systems Using CORBA – Fourth International Workshop on Object-Oriented Real-Time Dependable Systems – Janeiro, 1999

[XING, 1999c] – Xing, Gang e Lyu, Michael R.. – Testing. Reliability, and Interoperability Issues in the CORBA Programming Paradigm – Sixth Asia Pacific Software Engineering Conference – Takamatsu, Japão - Dezembro, 1999