UMA ARQUITETURA ABERTA PARA INTEGRAÇÃO DE DADOS DO AMBIENTE FABRIL COM OS NÍVEIS SUPERIORES DA EMPRESA

André Luiz Duarte Cavalcante

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

_____

Jorge Lopes de Souza Leão, Dr.Ing.

_____

Aloysio de Castro Pinto Pedroza, D.Sc.

_____

Luís Felipe Magalhães de Moraes, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

AGOSTO DE 2005

CAVALCANTE, ANDRÉ LUIZ DUARTE

Uma arquitetura aberta para a integração de dados do ambiente fabril com os níveis superiores da empresa [Rio de Janeiro] 2005

vi, 111 pp 29,7cm (COPPE/UFRJ, M. Sc., Engenharia Elétrica, 2005)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE.

1. Integração
2. Redes Industriais
3. Software

I. COPPE/UFRJ II. Título (série)

Agradeço a Deus a oportunidade da vida e a razão. À minha esposa e filha que souberam me dividir com o Mestrado. Ao Núcleo de Pesquisa e Desenvolvimento em Tecnologia Eletrônica e da Informação (NUTELI) pelo espaço e equipamentos cedidos e todo o apoio recebido durante o período de estudos. Igualmente ao meu orientador Prof. Dr. Jorge de Souza Leão pelos toques sobre o trabalho, correções e incentivo. Aos professores do NUTELI que ajudaram também com idéias e sua experiência. Também não poderia deixar de citar a UFAM e a COPPE pela bela empreitada de realizar em solo amazônico esse verdadeiro desbravamento de mentes que foi este Mestrado. Igualmente à SUFRAMA pelo financiamento do Mestrado e aberto o caminho para que nós outros pudéssemos realizar este trabalho. A todos os meus colegas de sala de aula que me ajudaram de alguma forma a concretizar os estudos durante as cadeiras. Igualmente aos funcionários do NUTELI que me apoiaram e incentivaram, até mesmo com idéias, a concretizar este trabalho. A todos

AGRADEÇO!

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UMA ARQUITETURA ABERTA PARA A INTEGRAÇÃO DE DADOS DO AMBIENTE FABRIL COM OS NÍVEIS SUPERIORES DA EMPRESA

André Luiz Duarte Cavalcante

Agosto/2005

Orientador: Jorge Lopes de Souza Leão

Programa: Engenharia Elétrica

Este trabalho propõe uma arquitetura de software aberta e flexível para a integração dos dados de uma planta fabril, gerados no chão-de-fábrica, com os níveis mais altos da empresa. Para tal, desenvolve um conjunto de componentes que formam uma camada de software fornecedora de serviços que possibilitam tal integração e, do ponto de vista das aplicações dos níveis superiores, os equipamentos reais da planta fabril passam a ser vistos como objetos distribuídos em uma rede.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M. Sc.)

AN OPEN ARCHITECTURE FOR DATA INTEGRATION BETWEEN THE SHOP FLOOR AND THE HIGHER LEVELS OF THE ENTERPRISE

André Luiz Duarte Cavalcante

August/2005

Advisor: Jorge Lopes de Souza Leão

Department: Electrical Engineering

This work considers a flexible and open architecture for integration of data generated in the shop floor, with the higher levels of an enterprise. For such, a set of components is proposed that constitutes a software layer that supplies services to make such an integration possible and, from the view point of the application, the shop floor equipments are seen as distributed objects in the intranet.

Sumário

Capítulo 1

INTRODUÇÃO

A Internet tem modificado as relações entre clientes e fornecedores, permitindo aos primeiros uma série de opções e facilidades que tornam a concorrência entre os últimos muito acirrada. Outrossim, os clientes se transformaram nos iniciadores dos fluxos de negócios, ordenando produtos e serviços. Os modelos de negócios entre empresas que passam a fazer parte da mesma cadeia de suprimentos também forçam uma mudança das relações entre as mesmas. Apesar de grande parte da infraestrutura de comunicações já estar pronta para a dsesignadsa “Era da Internet”, há uma outra questão, em si, binomial: - Os processos de manufatura estão aptos para a velocidade da Internet e seus novos paradigmas? E ainda: - No caso de não estarem, o que se poderia fazer para adequá-los?

A evidência empírica indica que a resposta para a questão poderia ser “ainda não”. Já no que se refere aos processos paradigmáticos de velocidade pode-se entender que as próprias tecnologias da Internet, em especial a Web que, como sistema distribuído de acesso a informações, permite proporcionar um novo ambiente de colaboração entre o chão-de-fábrica e a cadeia de suprimentos da empresa.

Entretanto, acredita-se que ainda faltam alguns elementos de integração entre a infraestrutura de manufatura e os elementos de comércio eletrônico e gestão da empresa. Estes elementos é que permitiriam preencher as lacunas que existem entre estes dois mundos.

Note-se que, nas últimas décadas, os investimentos na automação dos parques fabris geraram um legado muitas vezes incompatível com a Web ou com qualquer tecnologia de colaboração e integração. Some-se a isto o fato de que poucas empresas parecem estar preparadas ou abertas a novos investimentos sem os retornos esperados para os já feitos. Isto torna a tarefa de integração mais desafiadora, pois as novas propostas devem estar atentas também a questões de compatibilidade e provimento de mecanismos de compatibilização de tecnologias.

Para completar estas lacunas este trabalho propõe um estudo da integração fabril, das redes industriais e de uma infraestrutura de software para o desenvolvimento de aplicações de integração.

Fundamenta-se a partir de um modelo em camadas da automação industrial e seu objetivo é construir uma visão que agregue o clássico da automação com modernas técnicas de engenharia de software. Para isto, enumera algumas das questões essenciais para a nova era de integração que se iniciou e propõe soluções para os problemas de integração apresentados. Em conjunto, estes estudos mostram a necessidade do modelo em camadas apresentado, bem como a necessidade de elementos de integração intermediários.

Dentre as propostas de soluções estudadas, há a descrição da implementação de um sistema de integração que visa realizar a integração fabril através da construção de novas aplicações e o tratamento do legado conjuntamente, por meio de uma camada de

software intermediária, composta de um conjunto de componentes de software interligados. Estes componentes formam um arcabouço (*framework*) para a criação destas aplicações de integração sobre uma infraestrutura de rede existente.

Como exemplo optou-se pela descrição de um caso de implementação. Para isto são desenvolvidos componentes de software que obtém dados do chão-de-fábrica e criam, para os software dos níveis superiores, uma visão orientada a objetos das remotas da fábrica. As remotas são dispositivos que criam células de automação e são, em sua grande maioria, dispositivos mínimos; daí a necessidade de também caracterizar-se este tipo de equipamento.

Este trabalho está direcionado às indústrias de equipamentos eletroeletrônicos, particularmente aquelas alocadas no Pólo Industrial da Zona Franca de Manaus, PIM, onde se buscou grande parte dos cenários apresentados. Ainda assim, acredita-se que a solução aqui proposta possa vir a ser utilizada em outros pólos industriais.

1.1. Estrutura do Texto

A fim de realizar o exame dos pontos já descritos, decidiu-se por estruturar um texto baseado em uma seqüência capitular lógica.

O capítulo 1 contém esta introdução, o capítulo 2 expõe uma idéia geral sobre os níveis de automação e a integração entre eles, listando uma série de problemas encontrados no desenvolvimento de aplicações de integração entre as camadas. O capítulo 3 caracteriza o ambiente fabril quanto aos tipos de aplicações encontradas, os tipos de redes e alguns dos algoritmos que podem ser utilizados para a obtenção de dados e para acionamentos de recursos no chão-de-fábrica. O capítulo 4 propõe uma arquitetura de software firmada em técnicas de engenharia de software, que é, ao

mesmo tempo, uma base para o desenvolvimento de novas soluções de integração e tratamento do legado. Por fim, o capítulo 5 apresenta nossas conclusões sobre o tema e, após, vê-se uma relação das obras que foram referenciadas no texto.

1.2. Termos utilizados

- Arquitetura: é um conjunto de padrões e interfaces de programação que são unidas para formar uma visão geral de um sistema e a base de desenvolvimento de aplicativos, tornando-se uma descrição abstrata de uma solução; ver [14] para maiores detalhes sobre o termo.

- Componente de software: é uma unidade de composição com interfaces especificadas em um contrato e com dependências explícitas que pode ser distribuído independentemente e está sujeito a composição de terceiros (ECOOP, 1996); um componente de software não pode ter nenhum estado interno observável externamente (Szyperski, 1997).

- Funções inteligentes: este termo refere-se ao conjunto de softwares que realizam uma tomada de decisão baseada na inferência a partir de um conjunto de regras, claramente definidas e formalizadas em termos sintáticos e semânticos; portanto é um termo que será utilizado em sentido mais estrito.

- *Framework*: é um "arcabouço" de software desenvolvido com o objetivo de resolver um problema em particular, sem contudo ser necessariamente de um domínio específico; conjunto de bibliotecas de funções, objetos e componentes de software, descritas sob um contrato rígido e acessadas por interfaces de programação (APIs);

elemento de reutilização de software, o qual mantém um certo número de classes abertas para implementação, herança etc., que permitem ao desenvolvedor que o utiliza codificar apenas o que em sua solução difere do comportamento geral do *framework*, diminuindo os esforços no desenvolvimento de software novo.

- Integração: o termo será utilizado com diversos sentidos que dependem do contexto e do aprofundamento do estudo; originalmente será tratado no sentido mais amplo, isto é, significando fazer um conjunto de sistemas operar de forma ordenada e harmônica, realizando uma ou mais funções especificadas, não restringindo o nível da integração, se de software ou hardware, entre camadas, entre pessoas etc.

- Intranet, internet e a Internet: o termo **intranet** se refere à rede interna de uma empresa; o termo **internet**, em minúsculo, se refere à rede global da empresa, possivelmente integrando muitas plantas fabris sobre a Internet; o termo **Internet** se refere a rede mundial de computadores que integra as redes corporativas de clientes e fornecedores, sendo a Web a sua parte mais visível.

- Máquinas inteligentes: este termo refere-se aos equipamentos que possuem processamento, memória e podem realizar operações lógicas necessárias para uma tomada de decisão; para os objetivos deste trabalho estes equipamentos também devem permitir comunicação em rede; portanto, este termo será utilizado no seu sentido mais amplo; o termo **dispositivo inteligente** será tomado como equivalente à máquina inteligente.

- Padrão de projeto: é uma micro-arquitetura que descreve uma interação abstrata entre objetos colaborando para resolver um problema particular, entretanto sem ser tão especializado quanto um *framework*, pois não possuem qualquer implementação.

- Remotas: termo genérico para máquinas-ferramentas, sensores e atuadores inteligentes etc., isto é, dispositivos de campo com capacidade de comunicação; são os nós de uma rede industrial que realizam alguma função no processo produtivo.

- Tecnologias de internet: este termo refere-se a todo o conjunto de protocolos e ferramentas de rede sobre os quais se constrói uma intranet, uma internet ou a própria Internet.

- Web: é uma das aplicações possíveis sobre as intranets, internets e a própria Internet.

1.3. Direitos Autorais

Durante este texto serão citadas marcas de diversos fabricantes. Por se tratar de um trabalho acadêmico e não comercial, foram aqui empregadas somente para referência às tecnologias mais proeminentes do mercado ou por causa do seu uso em algum cenário ou caso. Acredita-se que estas citações não interfiram nos direitos do autor ou quaisquer licenças de uso dos produtos referidos.

Todos os desenvolvimentos efetuados durante a execução deste projeto ou foram executados em softwares devidamente licenciados ou são softwares livres e também acredita-se que, da mesma forma, não interfiram nos direitos do autor ou licenças de uso dos produtos utilizados.

Grande parte deste trabalho foi construído fazendo-se uso de softwares livres[1], disponíveis sob a *GNU Public License (GPL)*, *Library GNU Public License (LGPL)*, *Sun Community Source License (SCSL)* e *Apache License.* Veja-se Referências Bibliográficas para uma relação dos sítios onde se pode encontrar as definições das referidas licenças.

1 Também são chamados de softwares de código aberto ou simplesmente código aberto ou ainda *open source* no jargão em inglês. No decorrer do trabalho, estes termos podem aparecer e são tratados como equivalentes.

Capítulo 2

AUTOMAÇÃO E INTEGRAÇÃO

Este capítulo mostra uma idéia geral sobre os níveis de automação e a integração entre eles, bem como lista alguns problemas encontrados nesta integração. Expõe igualmente determinados conceitos-chave utilizados no decorrer do estudo.

2.1. Os Níveis da Automação

Por ser uma atividade complexa o desenvolvimento de aplicações para Automação Industrial presta-se especialmente ao paradigma da divisão e conquista. Costuma-se dividí-la de acordo com os equipamentos utilizados e com o fluxo de informações da planta fabril, criando uma hierarquia em níveis, em que cada nível apresenta peculiaridades de hardware e software, como se pode observar na Figura 1.

O nível mais básico é o nível do processo ou nível 0, que é o nível da aquisição e ação direta no processo produtivo. Nele, encontram-se estão elementos sensores e atuadores, discretos ou contínuos.

Após, temos o nível operacional ou nível 1, que é o de controle do processo. Neste nível, situam-se os Controladores Programáveis (CLPs ou sistemas mínimos) monitorando e atuando no nível 0.

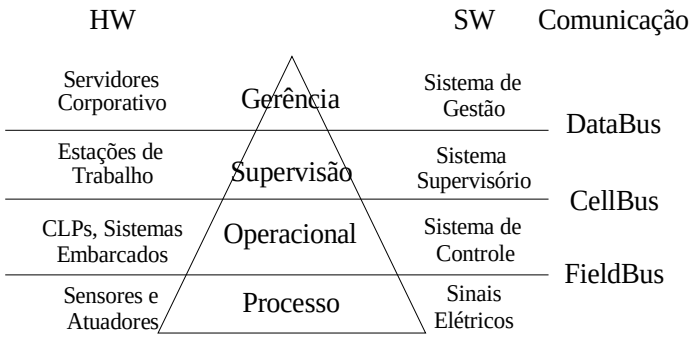


Figura 1 - Visão em Camadas da Automação

Mais acima observa-se o nível de supervisão ou nível 2 que é o responsável pela divisão da planta fabril em células de automação, cada uma controlada por um sistema de supervisão, tal como os SCADA. Neste nível estão computadores que fazem a supervisão e o controle dos equipamentos do nível 1 que, por sua vez, obtêm e formatam os dados da produção de uma forma que possa ser utilizada pelos softwares de gestão.

Por fim tem-se o nível de gerência ou gestão, ou ainda, o nível 3, aquele incumbido da tomada de decisão da planta como um todo. Também podem ser incluídos nele todos os sistemas estratégicos da empresa. Neste nível dispõem-se todos os computadores que fazem parte da rede corporativa e aqueles onde rodam os bancos de dados e softwares de gestão.

Do ponto de vista da comunicação temos três tipos de interfaces, também chamadas de barramentos, a saber: barramento de campo ou *FieldBus*, barramento de célula ou *CellBus* e barramento de dados ou *DataBus*. Primariamente, eles recebem dados da camada inferior, que são de alguma forma processados e transformados em informação para o nível superior. Fazem, ainda, o inverso, obtendo algum tipo de informação da camada superior que atua na camada inferior, modificando o estado do software ou do hardware subjacente.

Cada barramento tem o seu tempo de ciclo, isto é, a ordem de grandeza temporal em que os dispositivos geram ou coletam informações. Exemplo: para se verificar o estado de um conjunto de chaves, um CLP terá que fazê-lo em um tempo da ordem de milésimos de segundos. Esses tempos, bem como as suas tolerâncias, estão relacionados com a ocorrência de um evento do mundo real. Isto posto, um dispositivo de controle somente poderá monitorar tais eventos ou atuar sobre eles dentro dessas características temporais, conhecidas como “janelas de eventos”. Daí chamarmos este tipo de sistema de “sistemas de tempo real”. Observar que não há uma especificidade de qual valor de tempo é esse, pois este é dependente da aplicação. Veja-se outro exemplo: para acompanhar o aquecimento em um forno, um sistema poderá fazê-lo em décimos de segundo ou mais.

Perceba-se na Figura 1 que os diversos níveis formam trapézios. Na base do triângulo, o trapézio que representa o nível de processo é o de maior área, isto é, a figura apresenta uma visão centrada em dados. O nível de processo é o gerador dos dados para automação. Uma visão centrada em conhecimento seria um triângulo invertido, onde o nível de processo é o que teria a menor parcela, apesar de ser o originador dos dados. No tópico seguinte será conceituado mais claramente dado, informação e conhecimento para um melhor entendimento desta outra visão.

Outros autores, como em [4] , utilizam uma divisão em mais camadas, dando ênfase a certos agrupamentos de equipamentos com funções específicas ou colocando o que chamamos de “barramentos” também como camadas. Outros ainda quebram o terceiro nível em dois: um de gestão e outro de estratégia. Acreditamos que para os fins objetivados no presente estudo a divisão aqui apresentada é mais didática, pois difere claramente o software e o hardware de aplicação do software e do hardware de rede, dando ênfase igualmente nas camadas inferiores.

2.2. Dado, Informação e Conhecimento

O dado é tudo o que trafega nas redes industriais. Eles são os valores, unidades, caracteres, que são gerados por um processo e armazenados em algum lugar no próprio processo ou fora dele.

Um sistema centrado em dados visa exclusivamente capturar estes elementos de um lugar e os enviar a outro. De [2] , “o dado é um elemento puro, quantificável sobre um determinado evento.”

Além do conceito de dado há também o de meta-dado, ou seja, dados sobre dados. Esse conceito transforma os dados simples em algo de mais valor. Exemplo: um dado: R$ 100.000,00; o meta-dado: relacionado à venda 101203. Um meta-dado confere um contexto aos dados brutos. Vale observar que um meta-dado é um dado em si mesmo e, então, pode haver igualmente um meta-dado para contextualizá-lo.

Dados contextualizados são informação. Também de [2] , “A informação é o dado analisado e contextualizado. Envolve a interpretação de um conjunto de dados.” Isto posto, sistemas de informação são aqueles que podem contextualizar os dados obtidos, não bastando armazená-los ou enviá-los para outros lugares. Em algum ponto do processo ou fora dele os dados devem ser cruzados com o seu contexto a fim de que se possa ter informação. Informação é a base para uma tomada de decisão, em qualquer nível, seja ele estratégico ou na célula de automação. Exemplo: um programa de computador deve armazenar os níveis de falhas de um sensor em um processo e contextualizá-los, no tempo e com o tipo do sensor, a fim de poder decidir que já está na hora de indicar uma manutenção preventiva do sensor antes da próxima falha.

Ainda de acordo com [2] , “já o conhecimento refere-se à habilidade de criar um modelo mental que descreva o objeto e indique o que implementar, as ações a tomar.” Fala-se então de um conjunto de informações, isto é, cruzamento de dados

referentes a um objeto. É este conjunto que torna possível contextualizar os dados brutos. O conhecimento, como modelo, transcende a informação em si. O conhecimento pode gerar informação, confirmando ou negando uma informação anterior. Também pode gerar ação quando, num processo de tomada de decisão, direcionar as ações.

Não raro o conhecimento, dentro do conceito de conjunto de informações sobre um determinado objeto, estar implícito no agente humano do processo. Para a criação de sistemas com base em conhecimento é necessário, portanto, que este conhecimento se torne tácito, isto é, possa ser descrito de maneira clara e precisa e posteriormente implementado.

Ainda segundo [2] o desafio para as empresas da década de 80 era construir Sistemas de Informação, migrando dados para informação, a fim de melhorar o processo de tomada de decisão empresarial. A partir da década de 90, o desafio é construir Sistemas Baseados em Conhecimento, indo ao encontro às novas necessidades dos indivíduos, grupos e clientes em uma organização. Acreditamos que o desafio para a década corrente é o de popularizar o uso de ferramentas e técnicas baseadas em conhecimento para que alcancem os níveis mais inferiores da empresa. Este trabalho de certa forma abre espaço para que isso aconteça, pois permite uma certa contextualização dos dados mais próximo do chão-de-fábrica. Por exemplo, o uso de dispositivos mínimos para automação pode permitir que sistemas de gestão venham a possuir elementos de coleta e processamento de informações em uma célula fabril.

O triângulo da Figura 1 seria invertido se o processo de automação fosse encarado como um processo de geração de conhecimento na empresa, isto porque a cada novo nível em que os dados sobem na hierarquia eles seriam recontextualizados, gerando novas informações. É importante salientar, contudo, que a visão clássica mostrada na citada figura é ainda a de maior popularidade na literatura e na prática.

A infraestrutura de comunicação na indústria atual pode se ajustar a todos estes paradigmas e visões, possibilitando o seu estudo pelo pesquisador da computação, da engenharia ou mesmo das humanidades, dentro do espaço fabril. De fato os sistemas de comunicação instalados permitem a troca rápida de informações dentro da fábrica, notadamente dentro da mesma camada. Entretanto, há grandes dificuldades no que concerne em levar dados e informações de um nível para outro, principalmente por fatores históricos da evolução do hardware e do software nas diversas camadas.

Então, para um melhor entendimento, devemos fazer um pequeno histórico da evolução do hardware, do software e dos sistemas de comunicação associados à indústria.

2.3. Evolução do Hardware de Automação

Quando se observa a história do hardware para automação nota-se uma evolução natural dos sistemas puramente mecânicos para os mecatrônicos, com a adoção de dispositivos cada vez mais potentes. Aqui o termo potente refere-se a maior capacidade de processamento e memória do dispositivo. Também se pode observar uma integração crescente de serviços em poucos ou mesmo em um único dispositivo.

De diagramas Ladder, comuns na década de 80 e muito parecidos com diagramas pneumáticos, hoje os sistemas de automação contam com ferramentas de alto nível que permitem não somente o projeto e a simulação mas também o acompanhamento do sistema implantado, dentro de algumas limitações.

Por outro lado, nota-se igualmente uma mudança no profissional da área, cada vez mais afeito à engenharia elétrica e eletrônica, pois estes sistemas cada vez mais complexos exigem profissionais mais aptos a operá-los.

Mais recentemente ainda, vemos a entrada de profissionais de computação no meio fabril realizando projetos de automação.

Os pontos abaixo mostram alguns hardwares comuns para a criação de células de automação e dão uma idéia da evolução histórica do hardware de automação:

2.3.1. Sistema Centralizado

Comum na década de 60, grandes computadores realizavam o controle de vários periféricos, os quais desenpenhavam o papel de sensores ou atuadores no processo. Dado o seu custo elevado, apenas grandes corporações em países ricos possuíam tais soluções. A maior parte das plantas era monitorada por seres humanos e os atuadores eram quase todos mecânicos ou eletromecânicos. Essa visão clássica e centralizada pode ser vista nos filmes da época, como em “2001 Uma Odisséia no Espaço”[2], onde um grande computador era o responsável por “tudo e todos” em uma nave espacial a caminho de Júpiter. Na atualidade os sistemas tendem a ser uma grande rede que possibilita o processamento e o controle distribuídos.

2.3.2. Sistema Distribuído

Com o advento dos controladores na década de 70 e posteriormente do microprocessador houve uma disseminação do controle distribuído sobre uma planta fabril. Nasce, nesta época, o conceito de sistemas mínimos os quais possibilitariam processamento e controle distribuídos.

Os sistemas com controle distribuído eram construídas sob medida para a planta, o que a situa em um patamar de custos altos. Desta forma, indústrias siderúrgicas, petroquímicas e afins se beneficiaram deste tipo de sistema, uma vez que

2 De Artur C. Clark, com direção de Stanley Kubrick, 1968.

mantém uma planta com ciclos produtivos longos e produzem lucros suficentes para bancarem o desenvolvimento destas tecnologias. Somente se tornaram mais comuns na década seguinte, de 80, graças ao advento dos CLPs.

2.3.3. Controladores Lógico Programáveis (CLPs)

Uma evolução natural dos sistemas puramente mecânicos para os eletrônicos, seria a construção de um sistema modular em que os equivalentes mecânicos do controle são substituídos por sistemas eletrônicos. Daí o nascimento do CLP, na década de 80 e bastante popular na década de 90.

É com o advento dos CLPs, que as empresas tenham dado o primeiro passo para uma automação mais completa da planta fabril, criando-se as verdadeiras células de automação.

O desenvolvimento de ferramentas gráficas que permitiam a criação de sistemas a partir de esquemas que muito se assemelhavam aos antigos sistemas eletro-mecânicos são uma das chaves para o sucesso desta tecnologia. Entretanto, grande parte dessas células eram igualmente ilhas, isto é, o CLP operava sozinho e não estava interligado em rede, daí o surgimento do termo “ilhas de automação” em referência às células fabris.

2.3.4. Sistemas Mínimos

Com a popularização de microcontroladores, isto é, sistemas completos em um único chip ou placa de circuito impresso, houve a disseminação de sistemas pequenos, eficientes e baratos, os sistemas mínimos.

Essa popularização só foi possível porque as ferramentas de desenvolvimento passaram a utilizar linguagens de programação de alto nível, tais como C/C++, o que atraiu profissionais da área da computação para dentro do processo de automação. Fica claro que um programa feito em C pode possuir uma lógica muito mais complexa para a monitoração ou controle de uma célula do que os programas Ladder, sem as atuais extensões.

Entretanto, sistema mínimos não substituiram de fato os CLPs; ao contrário, estes ficaram cada vez mais ricos em funcionalidades. Um CLP hoje nada mais é que uma CPU, criada como um sistema mínimo, e mais um conjunto de módulos sensores ou atuadores que são acoplados àquela CPU conforme a necessidade para o controle da célula.

Uma das novas necessidades é a colocação de tais sistemas em rede, o que indica a criação de um módulo hardware de acesso a uma rede e seus softwares adjacentes. A programação, entretanto, ficou mais complexa: os diagramas Ladder, que eram um padrão ontem, hoje são quase exclusivos da família do CLP utilizado, uma vez que as extensões, necessárias para estas novas necessidades, tornam impossível a utilização do programa por dispositivos diferentes, às vezes dentro da própria família.

2.3.5. PCs Industriais

Na outra ponta, vários esforços foram feitos para a popularização do PC também no meio industrial. Entretanto a sua vulnerabilidade ao ambiente fabril, que é bastante hostil para diversos tipos de equipamentos incluindo aí os PCs comerciais, inviabilizava o seu uso, até que se conseguiu construir um dispositivo que utiliza em grande parte as placas disponíveis para *desktops*, mas condicionadas em um bastidor (*rack*) especial e montadas utilizando-se técnicas que permitem maior robustez.

Isso reduziu enormemente os custos da automação porque permitiu o uso dos padrões comerciais para interconexão de placas e periféricos. Mas, o principal benefício tenha sido o de também permitir o uso de soluções de software, como os sistemas operacionais, ambientes de desenvolvimento e aplicativos comerciais, o que simplifica e barateia o desenvolvimento de novas soluções.

De fato, a oferta de profissionais qualificados é maior no mundo do *desktop*, e então também temos aqui um fator importante para o barateamento dos custos no desenvolvimento soluções de automação, a partir de um maior número de profissionais e custos menores.

Quando o número de pontos de controle não for muito grande, que justifique um CLP, ou a quantidade de processamento for intensiva, ou mesmo quando se quer valorizar a interface com o operador, este tipo de solução é muito comum. Há inúmeras empresas que desenvolvem soluções completas

2.3.6. Sistemas Sem-Fio

O uso de equipamentos portáteis e protocolos de comunicação sem-fio são agora a “estrela da vez” para novos desenvolvimentos. Isto se deve principalmente por causa da propaganda massiva que se faz para a adoção de uma nova tecnologia, como fora o uso de redes *Process FieldBus* sobre RS485 em CLPs.

Entretanto, em algumas aplicações em que o custo com a infra-estrutura é alto e a necessidade de mudanças rápidas devido ao mercado é muito grande, justifica-se o investimento em soluções sem-fio e mesmo portáteis, num ambiente fabril.

Os tempos para desenvolvimento, colocação em produção até a chegada ao mercado estão cada vez menores, o que exige que a preparação (*setups*) de linhas de produção seja muito mais rápida e se tenha flexibilidade suficiente no parque fabril para

instalar células de produção eficientes, que garantam qualidade, nos menores espaços de tempo e com menos investimentos. Neste cenário é que temos a possibilidade do uso de soluções sem-fio; e provavelmente este deva ser o cenário da maioria das fábricas hoje.

Entretanto, há certos tipos de indústrias em que nem mesmo seria possível o uso do sem-fio. Um exemplo: em montagem de celulares as irradiações eletromagnéticas devem ser contidas a níveis mínimos, a fim de não interferir no processo fabril e, principalmente, nos testes do produto, em especial se tais irradiações estiverem em faixas próximas às dos equipamentos sendo montados e/ou testados. Outro exemplo: também na montagem de fornos microondas, nenhuma comunicação Bluetooth® é possível a partir de equipamentos com este tipo de sistema de comunicação nas imediações (alguns poucos metros) de um microondas ligado, dado que ambos irradiam praticamente na mesma faixa de freqüência e a potência irradiada por um forno microondas é relativa e infinitamente superior a dos dispositivos comunicantes; imagine-se a pequena possibilidade de tais equipamentos operarem em uma célula de testes.

Segundo os objetivos e as restrições deste trabalho, precisamos entrar em mais detalhes sobre dispositivos mínimos, os quais são a base para a proposta apresentada mais adiante.

2.4. Sistemas Mínimos

Os chamados sistemas mínimos[3] são equipamentos que possuem capacidade de processamento, capacidade de armazenamento de dados (memória) e capacidade de comunicação; a capacidade de processamento sugere a capacidade de tomada de

3 Neste trabalho será utilizado o termo sistema mínimo para se referir aos sistemas industriais, enquanto o termo sistema embarcado ou sistema embutido para os sistemas portáteis. Isto é uma convenção a fim de que o leitor entenda o objetivo e as limitações do texto.

decisão; a capacidade de armazenamento sugere a possibilidade de programação e retenção de dados; a capacidade de comunicação sugere a possibilidade de ligar um tal equipamento em rede. Essas capacidades são limitadas por questões de custo ou utilidade.

A limitação por custo é devida ao fato que estes equipamentos são, em sua maioria, criados para fins específicos. Não tendo uma produção em grande quantidade para amortizar os custos de sua produção e do seu desenvolvimento, este tipo de dispositivo deve primar pela simplicidade e respeitar certos limites de processamento e memória.

A limitação por utilidade é devida ao fato que, para um equipamento cuja finalidade é única, não há necessidade de grandes capacidades como em um *desktop* comum, onde há uma infinidade de possibilidades de uso. Esses sistemas são construídos para um uso único ou, no máximo, para alguns poucos usos bem restritos no ambiente fabril. Por exemplo, um equipamento de testes não necessita de uma interface homem-máquina muito rebuscada, eventualmente gráfica, desde que a maior parte do seu trabalho será interfaceando máquinas; fica claro que não há necessidade de grande processamento para manipular textos em um *display* não-gráfico, isto é, em modo texto.

Os sistemas mínimos não são necessariamente pequenos, do ponto de vista mecânico, nem tampouco simples, do ponto de vista da aplicação. São sistemas dedicados que são formados por muitos módulos elétricos e mecânicos específicos. O nome deriva essencialmente das limitações das capacidades acima mencionadas, se bem que, na maioria deles, ao menos a CPU possui tamanhos reduzidos. Contudo, mesmo sem muito processamento, é possível a criação de sistemas mínimos que monitorem ou controlem dezenas ou centenas de pontos analógicos e/ou digitais.

O termo sistema mínimo é facilmente confundido com outro: sistema embarcado. O termo em inglês, *embeded system*, é utilizado com uma significação bem ampla, que inclui o de sistema embarcado (ou embutido) e sistemas mínimos.

O termo sistema embarcado nasceu a partir de sistemas criados para automóveis, mas passou a ganhar fama com os celulares e PDAs, isto é, sistemas portáteis, com baixa capacidade de processamento e memória (se comparados a um *desktop* comum), com boa capacidade de comunicação e associados em um único invólucro com alguns periféricos de interface homem-máquina (IHM). PDAs e celulares originalmente tinham por alvo, serem feitos para automóveis.

Neste sentido, um sistema mínimo quando em tamanhos pequenos pode ser encarado como uma variante de um sistema embarcado. Os sistemas embarcados, entretanto, tendem a se tornar cada vez mais populares, o que favorece a sua produção em larga escala, vencendo o primeiro item acima citado dos motivos das limitações dos sistema mínimos, pois eles têm a possibilidade de aumentar cada vez mais as suas capacidades de processamento e memória. Eventualmente um PDA irá possibilitar até mesmo a execução de certos aplicativos criados para PCs. Portanto, estes sistemas deixarão de ser considerados mínimos em pouco tempo.

O termo sistema embutido é tomado aqui como sinônimo de sistema embarcado, sendo que aquele termo (sistema embutido) é mais comum para se referenciar os softwares de aplicação que rodam num sistema embarcado.

Notar que distinção entre sistema embarcado e sistema mínimo é convencional, pois diversos profissionais da área de automação tomam os significados exatamente ao contrário do que aqui colocado, sendo os sistemas embarcados ligados ao ambiente fabril e sistemas mínimos a PDA's.

Um sistema mínimo sempre existirá, mesmo com toda a evolução que a tecnologia possa alcançar. Isso porque haverá sempre aplicações específicas que requerem um hardware específico e, por questões de custo e/ou utilidade, não há sentido em se desenvolver tal solução a partir de sistemas mais complexos. Mas fica claro que a evolução do hardware leva a possibilidades de maior processamento e memória. Portanto o termo sistema mínimo deve ser entendido em sentido relativo. Estas possibilidades de maior processamento e memória traz para o mundo dos sistemas mínimos maiores funcionalidades embutidas, como por exemplo:

- um sistema de testes pode passar também a permitir a atualização do seu software via rede, ou a monitoração do seu uso pela rede;
- melhores IHMs possibilitam o uso de códigos de barras bidimensionais ou mesmo reconhecimento de imagens, o que ajudariam sobremaneira o operador na execução de suas funções;
- funções inteligentes de manutenção podem ser inseridas no sistema de tal forma que o próprio dispositivo de testes busque, em uma base de conhecimento, as informações necessárias para um auto diagnóstico; também pode realizar automaticamente uma chamada à manutenção, seja ela preventiva ou corretiva; após a manutenção, o próprio sistema realimenta a base de conhecimentos com novos tipos de defeitos, processos de correção etc.;
- funções inteligentes também podem permitir uma flexibilidade maior do sistema, adequando-o a diversas células de automação, permitindo, por exemplo, o uso do mesmo equipamento para a fabricação de diversos modelos de um mesmo produto.

Entretanto, sistemas com estas características são bastante complexos tanto do ponto de vista do hardware quanto do software que eles processam. Assim eles necessitam de uma plataforma de desenvolvimento adequada, provavelmente com um sistema operacional de tempo real como núcleo, as pilhas dos protocolos de comunicação também de tempo real, linguagens de programação de última geração e conjuntos de bibliotecas de funções e componentes de software (os *frameworks*). Isto permite multitarefa, composição de componentes e um desenvolvimento baseado em componentes. A utilização destas técnicas tem com fim a maximização da reutilização de software. Também há a necessidade de todo um conjunto de ferramentas de desenvolvimento que agilizem tanto a concepção quanto a implementação das soluções.

Como se pode observar pelo exposto, as gerações de sistemas de controle digitais têm sido combinações de práticas existentes na automação e avanços em tecnologias eletrônicas e da informação. Com a emergência dos microprocessadores nos anos 70, a visualização de controladores pneumáticos foi transferida para as telas dos computadores dos sistemas de controle distribuídos (DSC). Desde então a tecnologia PC tem encontrado o seu caminho nas aplicações industriais. Juntando-se aos sistemas mínimos feitos sob medida para aplicações industriais, os sistemas de controle correntes são tipicamente uma mistura de muitas técnicas[4]. Isto, é claro, gera um problema de fragmentação tecnológica.

4 Tradução livre de [15]

2.5. Fragmentação Tecnológica

A visão técnica do processo em camadas é muito útil e serve de base para as nossas apreciações. Este modelo facilita a automação gradativa de uma planta industrial, o que dilui os custos no tempo e os riscos por projeto. Entretanto ela pode inserir alguns inconvenientes:

- grande fragmentação das tecnologias utilizadas na automação e na comunicação entre as camadas;
- grande número de fabricantes e "padrões" de hardware, software e comunicações, que não são compatíveis entre si;
- os sistemas que são interrelacionados não estão interoperando entre si, porque foram desenvolvidos independentemente e sem a preocupação da interoperabilidade;
- há uma diminuição da visão integrada do processo fabril;
- é difícil tratar o legado;

Sobre a fragmentação das tecnologias, veja-se o seguinte cenário, baseado em um caso real de desenvolvimento, feito pelo Núcleo de Pesquisa e Desenvolvimento de Tecnologia Eletrônica e de Informação da Faculdade de Tecnologia da Universidade do Amazonas (NUTELI):

Uma fábrica possui várias linhas de produção para diferentes modelos de displays de telefones celulares. Ela deseja unificar o seu processo de testes através de um único tipo de jig, bem como disponibilizar todas as informações possíveis destes jigs para fins de análise e estatística do processo produtivo. Para minimizar os custos e os riscos do desenvolvimento, contratou três equipes diferentes: uma equipe irá realizar o projeto do hardware e do software do jig de testes, outra irá realizar a rede de jigs e os

programas dessa rede, a fim de prover comunicação entre os jigs e o banco de dados corporativo, e uma outra irá realizar o sistema de levantamento estatístico da produção, o qual poderá ser alimentado manualmente se o programa da rede falhar. Cada equipe ficará apenas com uma parte do problema para ser enfrentado, conforme o modelo hierárquico apresentado acima.

Fica claro que as equipes devem estar afinadas quanto ao cronograma de execução macroscópico e às respectivas interfaces, entretanto não necessariamente devam concordar em seus cronogramas específicos, de forma que o hardware pode eventualmente ser entregue antes da rede a qual, por sua vez, pode ser entregue antes do sistema de gestão, e o sistema de gestão, por sua vez, poderia operar no manual, caso o projeto da rede atrase.

Ao final do processo de desenvolvimento, têm-se alguns elementos de hardware com seus vários aplicativos, e vários dispositivos de comunicação e seus respectivos softwares, sendo que um novo protocolo proprietário foi proposto para realizar os requisitos nos prazos e custos que a empresa esperava.

Após o término deste projeto, reconheceu-se a necessidade da realização de automação em outro setor da fábrica. Mais uma vez o modelo se impôs e, ao final deste novo desenvolvimento, vêem-se mais alguns elementos de hardware e outros de software. A fábrica tem, ao final de um certo período, inúmeros padrões de comunicação e vários sistemas de software muito pouco acoplados, apesar de executarem tarefas muito semelhantes, isto é, eles estão interrelacionados mas não interoperam. A integração e a visão global da empresa ficam muito prejudicadas, como bem enfatiza [15].

O maior problema para esta integração está em se criar uma infraestrutura de hardware e software que permita integrar os diversos níveis, em especial o sistema de controle com os sistemas de supervisão e estes com os de gestão. No caso acima também não houve preocupação em como estes sistemas se comportarão ao longo prazo, quando se tornarem legados. A manutenção de um tal sistema, a fim de se inserir uma função inteligente, por exemplo, passa forçosamente pelo domínio das tecnologias envolvidas, ou seja, deverá ser feita pela mesma equipe que fez o desenvolvimento primário sob pena de nunca mais o sistema evoluir.

O processo mostrado necessitaria incluir alguns conceitos de engenharia de software para minimizar problemas com reutilização de recursos.

Apesar disto, atualmente observa-se uma preocupação maior, no processo de automação industrial, com a integração fabril, dado que seus benefícios são, em última análise, o que as empresas buscam em termos de qualidade no processo.

Como pode se perceber ao longo do texto, o termo *integração* foi até aqui utilizado em seu sentido mais amplo, conforme o item 1.2., merecendo agora uma abordagem mais específica.

2.6. O que é Integração?

Utilizando a definição de [8] para o caso particular do termo para o ambiente industrial, “integração é o processo pelo qual um conjunto de componentes tais como máquinas-ferramentas, robôs, sensores, alimentadores, transportadores, programas de aplicação, pessoas, etc., também designados por recursos, se reúnem para formar recursos ou sistemas/sub-sistemas de ordem mais elevada”. É um conceito recursivo em que a integração gera um único elemento ou novo recurso único visto de uma ordem mais elevada, isto é, quando galgamos as camadas expostos em 2.1. Daí a visão em

triângulo com uma ponta para cima: no nível mais alto, a empresa é encarada como um único ponto, o todo, e as aplicações são desenvolvidas do ponto de vista de um recurso único, portanto integrado. As demais camadas também encaram as suas imediatamente inferiores como diversos pontos que são integrados em um único, o qual é disponibilizado para a camada imediatamente superior.

Entretanto [8] continua: “implícita nessa reunião estará uma noção de complementaridade entre componentes e uma ‘fluida’ interoperação entre eles. Isto é, a criação de condições para que os vários componentes, independentemente do seu nível de autonomia, possam dialogar e cooperar com vista a atingir os objetivos do sistema de manufatura.”

2.6.1. Visões para Integração

A integração se mostra, portanto, como um trabalho de união de componentes que envolve cooperação, interoperabilidade, interfaces de comunicação, conjugação de objetivos e funções. Esses itens criam também visões distintas que a integração pode apresentar. Então, podemos encarar a integração fabril a partir de vários prismas (veja-se ainda [8]) ou visões, as quais serão abordadas abaixo.

2.6.1.1. Integração funcional

A reunião dos recursos pode ser de ordem funcional, isto é, os recursos são reunidos para que uma certa função seja realizada no todo integrado. Exemplos: a empresa, como um todo, tem a função de obter matéria-prima, gerar um produto acabado e comercializá-lo; no nível micro, uma célula de automação pode ser montada

integrando-se a função de diversos componentes: atuadores, sensores, CPU e operador humano para realizar uma tarefa específica, como por exemplo, um teste sistêmico em uma PCI já montada antes da embalagem.

2.6.1.2. Integração da informação

Cada elemento a ser integrado gera e consome informação, sendo então possível se pensar em uma integração das informações sendo geradas e consumidas, em níveis de abstração cada vez maiores. Então, a integração se faz removendo-se redundâncias e inconsistências, definindo-se meios e linguagens de comunicação comuns à todos os elementos, incluindo humanos.

A integração da informação visa, no nível mais alto, a construção de um modelo da empresa, o qual passa pela análise de processos da empresa. No nível da célula é a análise de onde a informação vem e para onde ela vai. Por exemplo, dentro de um sistema de monitoramento ambiental, cada sensor fica responsável por uma área a ser monitorada e a informação da temperatura daquela área será transmitida para um gerenciador, o qual integra todas as medidas utilizando um algoritmo definido para saber se já está na hora de ligar o condicionador de ar.

2.6.1.3. Integração por interfaces e comunicação

Aqui a visão da integração é centrada nas interfaces, isto é, em como os recursos se comunicam, incluindo aí o operador humano. Realizar uma definição integrada das interfaces e dos meios de comunicação também é uma forma de se encarar a questão da integração em um ambiente fabril.

Exemplo: um sistema de teste pode necessitar da intervenção do operador para uma verificação visual, então ele deverá ter uma interface clara e objetiva para questionar o operador quando da verificação e este, por sua vez, deve entrar a informação correta e não ambígua de como está o estado visual do equipamento sob teste.

A partir do exemplo, do ponto de vista da interface na integração, o sistema de teste como um todo é um sistema de inspeção visual, obtido pela interface entre o operador humano e uma máquina.

No nível gerencial a interface poderia se preocupar com os protocolos de colaboração entre diversas aplicações distintas.

2.6.1.4. Integração por coordenação

A reunião dos elementos deve ser “orquestrada” em suas participações no processo. É uma visão sistêmica da empresa muito útil do ponto de vista humano. Encarar a integração fabril como uma questão organizacional ou de coordenação é também uma das visões da integração.

Sob este ponto de vista, uma célula na fábrica seria a coordenação de vários recursos, por exemplo: os sensores de temperatura que fornecem os dados climáticos da sala de produção são coordenados por um “gerenciador” que tomará as decisões. Este gerenciador pode ser um operador humano lendo as indicações ou um sistema mínimo comunicando-se via rede com os dispositivos de campo.

Como se pode notar, pelos exemplos colocados, as diversas visões são apenas maneiras de se encarar o processo de integração, onde cada uma possui a sua utilidade.

2.6.2. Níveis de Integração

Dado que os requisitos atuais para sobrevivência das empresas, tais como flexibilidade, agilidade, eficiência e qualidade, demandam esforços em padronização e mudanças tecnológicas constantes, então eles somente podem ser atacados com um processo de integração fabril. Fica claro que o resultado deste processo de integração leva à preservação do conhecimento da empresa, à maior robustez da planta e à gestão mais otimizada[5].

Mas a integração por si também pode ser encarada como uma atividade em níveis, isto é, hierárquica. Isto é claro, pois ela surge a partir do modelo em camadas já exposto. Na atualidade os níveis de integração se relacionam mas extrapolam os níveis hierárquicos da automação. Abaixo, pode-se verificar os vários níveis de integração:

- Nível de célula, correspondendo à integração na inteface *FieldBus*.
- Nível de planta-fabril, correspondendo à integração na interface *CellBus*.
- Nível intra-empresa, correspondendo à integração na inteface *DataBus*
- Nível inter-empresa, correspondendo à integração da cadeia produtiva da empresa.

Para cada nível, há problemas específicos a serem contornados para a sua realização e há problemas em todos os níveis, muito embora a “moda” seja a integração global, isto é, uma integração inter-empresas, entre as empresas que participam de uma mesma cadeia de suprimentos, gerando negócios globais, ou eventualmente uma única empresa com vários sítios, em geral, em várias localidades geográficas distintas.

Estes níveis de integração serão abordados com maiores detalhes nos sub-tópicos abaixo.

5 Veja-se [8] para maiores detalhes.

2.6.2.1. Integração em níveis superiores

Este sub-tópico abordará os níveis de integração inter-empresa e intra-empresa de forma bastante ampla, por não ser o objetivo principal deste trabalho. Os sub-tópicos seguintes abordam os demais tipos de integração com mais detalhes.

Para se realizar a integração inter-empresa, entidades internacionais efetuam as atividades de padronização da comunicação, em termos de protocolos, estabelecimento e término de contratos etc. Além disso há ainda trabalho na definição, padronização e implementação destes protocolos e em como os sistemas devem interagir, sem gerar dependências exclusivas. A solução mais popular neste sentido está centrada no conjunto de protocolos que fazem parte das definições do *Word Wide Web Consortium* (W3C) para Serviços Web (*Web Services*), tais como: SOAP, WSDL, UDDI, WSFL. Com estes protocolos, sistemas *Enterprise Resource Planning* (ERP) e *Manufacturing Execution System* (MES) podem ser integrados entre empresas ou em empresas com vários sítios ou unidades produtivas.

Uma variante para este tipo é o uso de conectores em sistemas de gestão. Por exemplo, o ERP SAP/R3 possui conectores específicos para aplicações J2EE, normalmente uma outra aplicação ERP de um outro sítio. Mais uma vez, esta solução requer conhecimento específico dos sistemas. Esta solução somente pode ser utilizada quando um dos sistemas, provavelmente o sistema mais recente, é J2EE e um conector estiver disponível. Aqui o termo conector é o específico da plataforma Java® Enterprise Edition. O desenvolvimento de um conector é possível, porém é uma tarefa árdua e cara. Outras soluções de outras plataformas também estão disponíveis, mas todas sofrem mais ou menos desses mesmos problemas.

Outra forma para a integração, e talvez a única para sistemas que já estão em operação, é realizar a replicação de bancos de dados através de bancos de dados de interface.

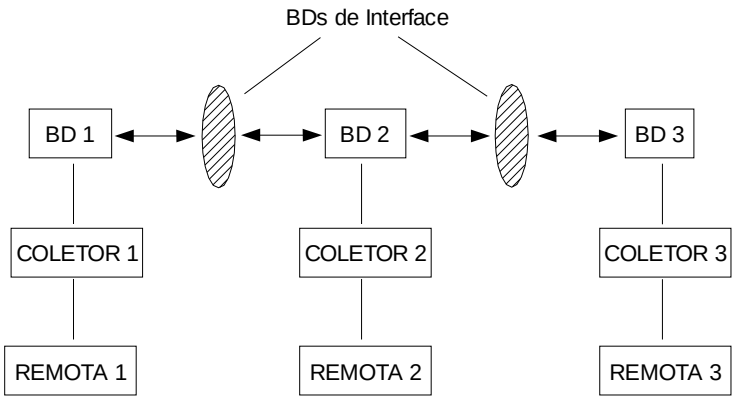


Figura 2 - Integração através de Bancos de Dados

Nesta solução, bancos de dados adequadamente produzidos conseguem atualizar dados de um para outro sistema. Esses bancos de dados atuam como interface entre os sistemas existente. Além do banco de dados propriamente falando, uma aplicação associada também deve ser desenvolvida para o tratamento adequado da replicação. Essa aplicação pode ser desenvolvida inteiramente dentro dos ambientes dos SGBDs envolvidos ou ser um serviço em algum lugar da rede corporativa. A produção de tais bancos requer conhecimentos específicos sobre os sistemas, bem como uma estrutura de hardware e software com elevados custos de suporte e manutenção. Esta solução é apresentada esquematicamente na Figura 2.

2.6.2.2. Integração da planta fabril: integração vertical

Em uma mesma planta fabril, não há sentido em se ter mais de uma unidade de ERP ou MES para se integrar e admite-se que a integração nos diversos níveis já estejam em andamento. Então o que se deseja é a integração vertical[6], isto é, aquela baseada na interface *CellBus*, integrando-se as células de automação com os sistemas de supervisão.

O objetivo da interface *CellBus* é disponibilizar meios para que as células de automação, isto é, cada conjunto de equipamentos do nível 1 (nível de controle) passe a ser encarado como um recurso único, passível de ser supervisionado pelo nível 2 (nível de supervisão). Essas células serão refletidas para os níveis de gestão dentro das aplicações de supervisão, sendo colocadas como um novo conceito: postos de produção. Em alguns casos, há uma verdadeira concentração de todos os postos de produção dentro de uma visão única: a linha de produção.

Para se realizar a integração em uma planta fabril, pode-se optar por soluções proprietárias ou por padrões internacionais.

Sobre o tema da padronização, caberia aqui uma interrogação: seria possível fazer-se integração a partir de protocolos proprietários? A resposta seria sim, para o conceito de integração adotado. Segundo este conceito, a integração ocorre quando se realiza a reunião de certos recursos que passam a operar como recurso único para níveis mais altos. Isso é importante, pois não há nada neste conceito que obrigue o engenheiro

6 O termo integração vertical também é utilizado na literatura (conforme [11]) para se referenciar o processo de uma planta fabril passar a produzir os elementos e insumos que seus fornecedores (integração vertical para traz) lhe demandavam ou os elementos e insumos que demandava a seus clientes (integração vertical para frente), dentro de uma mesma cadeia de suprimentos. Aqui o termo é utilizado apenas para enfatizar a integração entre as camadas inferiores para as superiores de uma planta fabril. Da mesma forma, o termo integração horizontal é entendido como a integração operada dentro do processo produtivo no mesmo nível hierárquico.

a realizar uma integração fabril a partir de sistemas abertos ou protocolos padrão. Uma vez reunidos os recursos dentro de um contexto comum que passe a ser encarado como um recurso único, a integração está acontecendo.

A propósito do uso de sistemas proprietários, o autor deste trabalho teve oportunidade de observar, em uma grande empresa do pólo industrial da Zona Franca de Manaus, um sistema de coleta de dados interessante, apesar de muito simples, que será agora descrito para fins de análise:

O sistema MES da empresa necessitava coletar dados diretamente dos equipamentos de teste da produção (jigs), contudo não houve entendimento entre a engenharia e o setor de Tecnologia da Informação (TI) sobre o que usar. A solução veio de um estagiário de TI que fez um pequeno programa, já ao final do cronograma, o qual roda como um serviço na plataforma Windows® NT e que a cada 3 minutos lê uma pasta onde o jig colocava arquivos com os resultados dos testes (em um formato proprietário), reformatava estes arquivos em arquivos no formato que o MES podia ler nativamente (portanto novamente proprietário) e os escrevia em uma outra pasta, a pasta de destino. Um segundo programa, que roda como serviço também a cada 3 minutos varre a pasta de destino e se encarrega de atualizar a base de dados do MES. Todos eles rodando no PC industrial que controla o jig.

Esta solução é interessante pela simplicidade, mostra uma tentativa de integração, mas possui diversos inconvenientes que serão apontados aqui para estudo:

- pouco flexível: se houver uma atualização do jig ou do MES novo desenvolvimento deverá ser realizado para coadunar eventuais mudanças, como de fato ocorreu. A motivação da entrada do autor na empresa foi justamente a de realizar um novo desenvolvimento para um novo jig. O melhor talvez fosse a colocação de um sistema

intermediário que fosse suficientemente flexível para permitir que, na entrada de um novo modelo de jig ou uma nova versão do MES, somente um único módulo fosse alterado conforme a mudança ambiental, enquanto as interfaces internas dos sistemas permanecessem as mesmas;

- pouco eficiente: quando se perguntou de onde veio a grandeza “3 minutos” ninguém, nem da engenharia, nem do TI soube explicar o seu porquê. Os jigs tem tempo de ciclo variável que, em média, dá 2 minutos e meio por placa produzida. Um sistema mais eficiente capturaria os dados a medida que fossem sendo gerados, ou melhor ainda, os próprios jigs poderiam emitir seus relatórios em um formato padrão quando necessário;

- é limitada: essa solução somente resolveu a questão da coleta de dados: não há nenhum tratamento para os dados, não há contextualização e, portanto, não há geração de informação sobre o estado do jig e dos produtos sendo testados no nível da célula. Isto vai ser realizado no MES fora do tempo em que os eventos acontecerem. Portanto, a integração aqui é apenas superficial. A contextualização que falta está implícita no MES dentro de algumas regras de negócio, tais como as relações de tempo entre os ciclos de produção. Uma solução melhor talvez fosse a de colocar um programa intermediário em algum lugar na rede, o qual faria essa contextualização e daria um retorno imediato para a engenharia, como se fosse um braço do MES mas diretamente relacionado à supervisão dos equipamentos e testadores da fábrica;

- não é robusta: em presença de falha do jig, ou somente da rede, o sistema MES não pode ficar sabendo deste fato, pois todos os aplicativos que poderiam fornecer esta informação estão em falha simultaneamente com o jig, ou o pior, o MES não consegue discernir entre uma falha do jig e uma falha da rede. Retirar o segundo programa, o de varredura da pasta de destino, para algum outro lugar na rede, não ajuda, pois isto insere uma brecha de segurança com a presença de compartilhamentos diretamente no sistema de arquivo do sistema operacional do jig, o que pode ser inaceitável. Uma solução melhor talvez seja a criação de uma aplicação intermediária, colocada em algum lugar da rede que possa monitorar os jigs e que o MES pudesse acessar de forma mais transparente. De fato, o ideal é que o MES não soubesse se está operando diretamente com um jig ou se está se comunicando com a aplicação intermediária. Isto somente apareceria em caso de falha, porque esta saberia exatamente se o jig está em falha ou se a rede entrou em falha, e poderia informar isso ao MES;

- mostrou uma falha maior: o problema da integração aqui não está no nível fabril, mas na coordenação de esforços. A falta de entendimento entre a Engenharia Industrial e o setor de Tecnologia da Informação tem sido um grave empecilho para a busca de soluções integradas para as empresas. Um *middleware*, desenvolvido sob medida para esta situação, que expusesse o vocabulário comum e os termos distintos entre os dois mundos, o da engenharia industrial e o de TI, poderia ter resolvido o impasse, onde cada qual implementaria a sua solução e o *middleware* se encarregaria de realizar o transporte das informações de um mundo para outro. Entretanto tal *middleware* ainda deve ser desenvolvido o que torna mais cara a solução.

Este cenário mostra um dos grandes problemas da integração: integrar sistemas que já estão em produção e, portanto, não permite nenhuma alteração nos contextos em que estas aplicações operam nem nas próprias aplicações.

De fato, podemos encarar o problema acima como o problema de integração entre dois elementos da planta fabril que são legados e incompatíveis entre si. O fato de haver sistemas legados e incompatíveis mostra algumas características de como a integração é encarada na empresa:

- não houve um estudo prévio das escolhas tecnológicas para a criação das células de automação, assim as interfaces não foram adequadamente definidas, faltado portanto uma visão de comunicação e interfaces para integração;
- não há uma visão integrada de como a informação deve trafegar, portanto não há uma visão ou algum modelo de empresa;
- não há uma coordenação dos esforços visando a integração, nem as interfaces foram planejadas; deste modo os ambientes operacionais da empresa, que são interdependentes, não apresentam a mesma "linguagem" para o entendimento e a solução dos problemas, nem tampouco há coordenação de esforços para se conseguir isso; e
- o mais comum prevalece: os profissional de produção, de automação, de tecnologia da informação (TI), gastam mais tempo e, portanto, recursos, tentando resolver os problemas cada um a seu modo do que pensando e agindo na empresa como um todo, é o que se chama "apagar incêndio" no jargão do chão-de-fábrica.

Um outro problema que o engenheiro responsável pela integração tem que lidar é com a necessidade de tratar sistemas de tempo real, isto é, sistemas que geram informações e que agem no processo em eventos que ocorrem sob condição de tempos específicos e com pequenas variações. O período de tempo para uma ocorrência desses eventos pode até ser grande, entretanto as possíveis variações para a sua ocorrência, o *jitter*, tendem a ser bem pequenas.

2.6.3. Integração de célula

Neste nível de integração, o engenheiro está preocupado em interligar diversos componentes em uma única unidade funcional, chamada célula de automação. Corresponde à integração na interface *FieldBus*. Há diversas soluções de interfaces, chamadas de interfaces de campo, que permitem a interconexão de máquinas-ferramentas, atuadores e sensores inteligentes, sistemas de medição etc. A escolha apropriada de qual tecnologia usar é fundamental para o sucesso do desenvolvimento. Há diversos fatores que influem na escolha, dentre os quais:

- escolha prévia da tecnologia pela empresa, quando esta dita as escolhas tecnológicas; neste cenário o desenvolvimento pode correr riscos porque a equipe de desenvolvimento pode não ter o domínio da tecnologia ditada pela empresa, ou esta escolha pode não ser a melhor do ponto de vista técnico;
- *expertise* da equipe de desenvolvimento, quando esta escolhe a tecnologia que melhor domina, a fim de minimizar os seus riscos no desenvolvimento; neste cenário a escolha também pode não ser a adequada do ponto de vista técnico;

- uso da tecnologia dominante, quando a escolha é feita pelo mercado, notadamente por questões de custo ou simplesmente por causa da propaganda realizada, a qual pode dar a ilusão de uma tecnologia ser melhor que outra;
- uso de uma solução proprietária personalizável, e que pode não atender às necessidades da fábrica;
- desenvolvimento da própria solução, “do zero”, e que pode não chegar ao término por questões de prazo curtos, especificações incompletas e capacitação da equipe de desenvolvimento.

Dentre as escolhas tecnológicas disponíveis devem ser citadas: *FieldBus*®, *ControlBus*®, *DeviceNet*® e *ModBus*®. Muitas dessas soluções são construídas sobre a camada física mais comum que também deve ser citada: RS485. Sistemas legados ponto-a-ponto com ModBus® sobre RS-232 ainda são bastante comuns.

Há interfaces de um nível mais inferior, utilizada para interconexão curta entre *chips*/dispositivos ou placas, tais como *I2C*® da Philips e *CanBus*® da Motorola, sendo o primeiro um padrão muito comum em dispositivos da indústria de eletro eletrônicos e a última muito utilizada em dispositivos da indústria automobilística. Redes CAN podem interligar até algumas dezenas de metros e, com dispositivos buffers no caminho é possível se conseguir até centenas de metros servindo, portanto, como interface de campo também.

Como regra geral, cada grande fabricante de dispositivos também desenvolveu o seu barramento de interconexão entre dispositivos. Por exemplo, a Agilent (antiga divisão de equipamentos de teste da HP) criou e mantém um padrão de comunicação

paralela chamado de *Gpib*®, o qual pode interligar diversos equipamentos de testes como voltímetros, freqüencímetros, geradores de sinais etc. em uma mesma célula a partir de um dispositivo controlador.

Exemplos de uso dessas tecnologias podem ser encontradas nos sítios dos diversos fabricantes de equipamentos. Aqui serão citados apenas dois exemplos distintos como uma amostra da diversidade:

- a Lincis, uma empresa portuguesa, com atuação principalmente na Europa ocidental disponibiliza um coletor de dados e um apontador de produção a partir de duas rede CAN, uma para cada tipo de equipamento que comercializa. A rede de apontadores é interligada a um PC de supervisão. Este PC recebe os dados coletados e pode disponibilizar as informações para sistemas de gestão. Aqui a solução pode ser utilizada tanto na construção de uma única célula funcional que adere a um processo, como pode ser utilizada para unir diversas células funcionais distintas, participando neste caso também da integração de diversas células. De fato, a solução da Lincis contém um software que realiza algumas das funções de supervisão e controle.

- a Profibus International desenvolve uma solução em 4 sub-domínios: Profibus DP para automação fabril, Profibus PA para automação de processos, Profidrive para controle de movimento (*motion control*) e Profisafe para situações em que a segurança das informações sejam fundamentais. Estas soluções podem ser construídas sobre redes RS-485. Sistemas supervisórios integram a solução permitindo uma integração

"suave" da planta fabril. Os sistemas derivados do consórcio da Profibus International é o padrão *de facto* da automação industrial, e utiliza na camada de aplicação uma variante do *FieldBus*. Uma evolução destes padrões, para operar sobre redes Ethernet e com conectividade na Internet é chamado de ProfInet e é a estrela da vez nas propagandas da companhia para esta tecnologia.

Fica claro que um sistema legado sobre CAN terá inúmeras dificuldades para interoperar com sistemas Profibus, o que será conseguido somente por meio de dispositivos chamados *gateways*.

2.7. Necessidade de Padrões Abertos

O desenvolvimento de software tem se caracterizado por uma busca incessante na reutilização de software, isto é, utiliza-se a experiência passada e, principalmente, o código já escrito como base de novos desenvolvimentos. Primariamente criou-se o conceito de funções, após o de bibliotecas de funções. Modernamente vê-se o aparecimento de conjuntos de bibliotecas específicas que implementam um determinado domínio, são chamados de *frameworks*. Todos estes são exemplos de técnicas de reutilização de software, isto é, deixar o software disponível para que um terceiro possa realizar as suas funções a partir de uma composição deste com o software de terceiro. Para estudar e propor soluções neste contexto, a engenharia de software nos abre uma gama de ferramentas e métodos para bem melhor desenvolver o trabalho de construção de um software.

Ao se basear novos desenvolvimentos na reutilização de software já escrito, mais do que manter um legado ou compatibilidade, espera-se uma diminuição de custos e um favorecimento da manipulação da complexidade. De fato, um sistema de software qualquer nos dias correntes é um conjunto de diversos arquivos com códigos fonte, talvez em várias linguagens, e milhares de linhas de código, sem falar nas diversas IPAs (Interface de Programação de Aplicações ou APIs em inglês[7]) que o programador deve ter acesso. Sabe-se que a complexidade cresce exponencialmente em relação ao número de linhas de código, então, para um só desenvolvedor fica muito difícil ou mesmo impossível gerenciar esse nível de complexidade.

Ainda mais, a engenharia de software chama a atenção para o processo de desenvolvimento de software em si mesmo, desde sua concepção até sua implementação. Ela convida a pensar o software desde as suas origens e vai além da entrega do mesmo ao cliente, pois assume que o processo será o de melhoria contínua no produto. De fato boa parte das grandes produtoras de software subexistem com atualizações de seus softwares.

Ao se pensar mais profundamente no problema da integração, isto é, tornar elementos dispersos em um recurso único, vê-se os benefícios que a reusabilidade de software pode alcançar. Se em uma planta fabril há diversos tipos de equipamentos para serem controlados e monitorados, em termos da lógica de controle ou monitoração não há tanta diversidade, pois que os mesmos algoritmos de controle podem ser utilizados para uma infinidade de equipamentos e casos.

A possibilidade de se utilizar componentes de software reutilizáveis em vários sistemas, mudando-se tão somente o acesso hardware é um ganho significativo em termos de tempo e custo de desenvolvimento.

7 Utilizaremos a partir deste ponto a sigla em inglês por ser mais usual tanto na literatura quanto no ambiente fabril.

Do ponto de vista da comunicação na rede, o uso de padrões e protocolos fechados para realizar a integração inviabiliza a reutilização, uma vez que cada equipamento ou sistema passa a ter um protocolo próprio para ser tratado. O difícil, quando o engenheiro se depara com um caso de protocolo proprietário, é justamente a obtenção da especificação do protocolo para se realizar um novo desenvolvimento, e isto somente é viável quando for legalmente possível e ainda pode necessitar pagar pelo acesso à especificação. Portanto, ao se desenvolver um sistema que não vise a integração ou utilize padrões de comunicação fechados perde-se o rumo da qualidade e da eficiência no trabalho[8].

Ao contrário, o uso de padrões abertos para integração favorece a reutilização no ambiente fabril e dissemina a cultura da padronização e da normatização, passos essenciais para a obtenção da qualidade em um processo. Isto é importante, pois parte das empresas nacionais ainda possui a visão de que os seus recursos e tecnologias são uma exclusividade sua e o seu segredo de negócio, bem longe de onde está o verdadeiro conhecimento para o negócio: na contextualização dos dados.

Outro ponto a favor da padronização está na interoperabilidade entre os diversos sistemas, o que somente será real quando os diversos sistemas forem construídos para a existência dessa interoperabilidade. Isto somente vai ser conseguido quando padrões de projeto e de interfaces forem utilizados no desenvolvimentos dos sistemas.

8 Veja-se o artigo de SAMPAIO, Lúcia Isabel. ERP, CRM e outras ferramentas de TI no site do Instituto de Educação Tecnológica, IETEC, de Minas Gerais, http://www.ietec.com.br/ietec/techoje/techoje/tecnologiadainformacao/2004/05/17/2004_05_17_0002.2xt/-template_interna, último acesso em julho de 2005. O IETEC é ligado a Federação das Indústrias de Minas Gerais.

Capítulo 3

CARACTERIZAÇÃO DO AMBIENTE FABRIL

Um sistema de automação industrial atual deve possuir algumas características básicas a fim de permitir a sua integração com outros sistemas e, por fim, da planta fabril como um todo. Para realizar esta caracterização, primeiramente devemos caracterizar o ambiente industrial em si, para o qual os sistemas de automação devem ser desenvolvidos, tanto do ponto de vista do hardware quanto do software de aplicação.

Quanto a caracterização do hardware, pode-se recorrer ao exposto nos tópicos 2.3 e 2.4. Quanto a caracterização do software, isto será visto neste capítulo. Além disto, o leitor também encontrará algumas técnicas para acesso aos dispositivos mínimos, que servem tanto para a obtenção de dados quanto para ação em um processo fabril.

3.1. Caracterização dos Softwares num Ambiente Fabril

Para tornar mais claro os estudos, as aplicações (os softwares) num ambiente industrial foram divididas em dois grandes grupos. Utilizando um jargão da indústria de telecomunicações, os dois grupos recebem os nomes *infra-estrutura de base*[9] e *última milha*[10]. Esses termos são utilizados quando o engenheiro se refere a infra-estrutura da

9 Também serão aqui referidos como software de base, mas não devem ser confundidos com os sistemas operacionais, os quais também são conhecidos como softwares de base. O contexto fará a distinção.

10 Também serão aqui referidos como software da aplicação final.

rede que alcança as grandes artérias das cidades ou chega até a entrada de pequenos povoados no meio rural, no primeiro caso, e quando esta infra-estrutura alcança o assinante, isto é, o usuário final do serviço, na última milha da rede. Essa distinção é importante porque o mais caro para a indústria de telecomunicações não é gerar a infra-estrutura de base, mas a última milha.

Aqui será utilizado este jargão para o desenvolvimento das aplicações de um ambiente fabril. Uma discussão específica sobre a rede em que estas aplicações funcionarão será efetuada mais adiante. No que tange às aplicações, os custos se invertem: é bem mais caro gerar uma infra-estrutura de base que uma aplicação última milha. Entretanto uma aplicação última milha jamais poderá ser construída sem uma infra-estrutura de base, de forma que as soluções de última milha atuais somente podem oferecer os serviços que a base seja capaz de gerir. De fato as soluções de última milha realimentam o processo de desenvolvimento da infra-estrutura de base com suas necessidades e, também, a partir do código desenvolvido sob medida que puder ser generalizado para se tornar uma infra-estrutura de base.

3.1.1. Aplicações de Infra-estrutura de Base

As aplicações que criam uma infra-estrutura de base para o desenvolvimento de aplicações de integração são o conjunto de ferramentas, bibliotecas de funções e componentes de software (os *frameworks*) que são fornecidos pelos fabricantes para o desenvolvimento de soluções completas. Elas tendem a criar uma camada de software entre a rede e a aplicação de última milha e, mais recentemente, tendem a transformar os dispositivos sobre controle em objetos da rede. Também fornecem serviços de persistência, colaboração e uma grande variedade de protocolos de aplicações, normalmente os já consagrados para escritórios como FTP, HTTP, SMTP, POP etc.

Alguns trazem consigo já implementados protocolos de níveis mais baixo ou mesmo alguns protocolos industriais de campo como *ModBus*®. Detalhes sobre *frameworks* podem ser vistos [4] . Algumas dessas soluções acabaram se tornando um padrão *de facto* para o desenvolvimento de aplicações.

Para a construção deste tipo de soluções, isto é, sistemas de infra-estrutura de base, utiliza-se sistemas de acesso remoto a objetos e seus métodos. As tecnologias mais comuns que devem ser citadas são:

- DAC Active X®: é uma forma de componentes COM utilizados para aplicações de automação e controle[11];
- RMI: é um método para troca de mensagens, isto é, chamadas de métodos remotos em Java.
- CORBA: uma arquitetura de comunicação de objetos distribuídos definido pela OMG.

Com base nestas tecnologias, pode-se desenvolver algumas soluções para automação industrial, tal como a AIP (*Aspect Integrator Platform*), que é uma plataforma de desenvolvimento de aplicações de controle de processos, construída sobre COM[12].

Podemos citar igualmente a solução Jini™ que é uma arquitetura de componentes e objetos em Java para aplicações de colaboração em sistemas embarcados e que pode ser utilizada para automação fabril, que utiliza RMI para troca de mensagens.

11 Veja detalhes em [14]

12 Veja detalhes em [4]

Ainda no mundo Java, pode-se citar um *framework* chamdo JIM® (*Java for Industrial Monitoring Framework*)[13], desenvolvido por um consórcio de indústrias e fabricantes de equipamentos para indústrias, que também utiliza RMI para troca de mensagens, mas também permite o uso de CORBA. Esta solução é voltada eminentemente para sistemas industriais e provê diversos mecanismos de monitoração de dispositivos, mudanças de estado, armazenamento de dados em memória, transformação de dados dos dispositivos em objetos Java etc.

Há sistemas híbridos como o JTRON[14], desenvolvido pelo ITRON project, consórcio de empresas de hardware de consumo japonês, que é construído em Java e C++. É uma arquitetura híbrida não somente pelo uso de duas linguagens de programação mas também porque inclui um núcleo de tempo-real e uma máquina virtual Java em um único sistema, com objetivos de ser multi plataforma e suficientemente pequeno para ser compatível com processadores de 8 bits e 16 bits além dos mais atuais de 32 bits. Estes tipos de processadores são os mais comuns do mercado de eletrônicos de consumo. O JTRON também permite o uso de CORBA. Apesar de ter sido desenvolvido para o mercado de consumo, a união de C++ e Java é bastante procurada no meio industrial devido a questões de performance quando comparadas com as aplicações desenvolvidas puramente em Java.

Pode-se citar também esforços como o projeto ADOORATA (*A Distributed Object-Oriented Architecture for Real-Time Automation*), um consórcio de cooperação entre Brasil e Alemanha e que gerou um ambiente completo de desenvolvimento de aplicações complexas em tempo-real, como pode se visto em [3] .

13 Veja detalhes em [6]

14 Veja detalhes em [9]

Alguns leitores poderiam estar se lembrando do sistema LabView®, da National Instruments. Este sistema é, na verdade, um aplicativo de última milha que tem evoluído nas últimas décadas para se tornar uma verdadeira plataforma de desenvolvimento de soluções de teste, medição e automação. Nela o engenheiro pode realizar desde a concepção até a implementação e o acompanhamento do sistema instalado, que inclusive, é instalado sobre a plataforma LabView®, mas que somente funciona com base em dispositivos que lhe são compatíveis. Estes dispositivos devem possuir um ou mais DAC Active X®, ou então são fornecidos componentes com conexão ao formato nativo do LabView®. A maioria dos fabricantes de hardware para automação hoje, tem se preocupado em fornecer conexão ao LabView®, normalmente via um DAC Active X®. Apesar deste sistema ter evoluído a partir de uma sistema última-milha, está citado aqui por causa das suas características estarem já como as características dos sistemas de base.

Uma característica comum a todos estes sistemas é a existência de um conjunto de APIs que são disponibilizadas através de uma linguagem de programação. O ambiente de programação pode prover um mecanismo de compilação ou interpretação de *scripts*. Não raro, entretanto, estas APIs aparecerem sob forma gráfica para o desenvolvedor, isto é, na forma de ícones que representam os componentes de software e hardware utilizados no desenvolvimento da aplicação final.

Juntamente com o ambiente de programação dos sistemas de infra-estrutura de base, também podem ser encontradas inúmeras pequenas aplicações prontas, chamadas de *ferramentas* que facilitam o trabalho de configuração do ambiente de programação. Algumas delas até mesmo geram código para o desenvolvedor. Uma ferramenta muito

útil, por exemplo, é a criação de classes que representam os dispositivos reais da fábrica; algumas soluções podem inclusive buscar na rede por mensagens padronizadas e criar um espelho em software do ambiente fabril.

Estes sistemas tendem a forçar o desenvolvedor a realizar determinadas tarefas para desenvolver novas soluções, seja limitando a ação do desenvolvedor dentro de um *script*, seja através da necessidade do uso de ferramentas próprias para sua configuração. Os sistemas finais tendem a ser muito parecidos, pois são baseados em uma mesma plataforma de desenvolvimento e são desenvolvidos com as mesmas APIs. Portanto, as aplicações de infra-estrutura de base podem ser encaradas como arquiteturas as quais são utilizadas para o desenvolvimento dos aplicativos verdadeiros, isto é, os aplicativos de última-milha.

3.1.2. Aplicações última-milha

As aplicações finais que os usuários utilizarão, dentro do ambiente fabril, denominamos aqui de aplicações de última-milha. Este tipo de aplicação é tipicamente desenvolvida dentro de uma especificação de requisitos da fábrica, e utiliza as infra-estruturas de software, tais como os sistemas anteriormente vistos.

Para o desenvolvimento dos aplicativos finais, as grandes empresas de software operam um formidável trabalho para “facilitar” a integração em uma indústria. As aspas são propositais, uma vez que cada empresa tem interesse em manter-se sozinha no mercado fornecendo soluções completas a seus clientes, em todos os níveis hierárquicos da empresa. Isto mostra que é possível desenvolver um conjunto de tecnologias capazes de realizar o desenvolvimento de aplicações de integração entre o chão-de-fábrica e os níveis superiores da empresa, bastando que fossem abertas para permitir o seu uso intenso no chão-de-fábrica, conforme visto no item 2.7.

Caberia agora conhecer melhor os tipos de aplicações que são desenvolvidas em um ambiente industrial. Para isso fez-se uma tabulação de alguns dos diversos tipos de aplicações de última milha que encontramos em um ambiente fabril. Nos tipos elencados abaixo utilizaremos para a tabulação os seguintes requisitos: de tempo, de processamento e memória, de flexibilidade, de robustez.

Através do requisito de tempo é possível classificar se o sistema é de tempo real ou não. Através do requisito de processamento ou memória verifica-se se há a possibilidade do uso de dispositivos mínimos. O requisito flexibilidade refere-se a capacidade de um sistema ou dispositivo se adaptar a diversas situações, talvez em circunstâncias não previstas ou ser facilmente modificado para esta adaptação. A técnica de flexibilidade mais comum é o super dimensionamento do hardware de processamento e dos números de pontos a serem monitorados e/ou acionados. Já a robustez se refere a capacidade de o dispositivo operar mesmo em presença de falhas. É claro que um sistema não irá operar da mesma forma em falha ou quando está em operação normal. A robustez é relativa também ao tipo de falha que se espera em um determinado equipamento. A técnica de robustez mais comum é a redundância.

3.1.2.1. Aplicações de Controle

São aplicações de ação direta no processo, tais como: sistema de controle de motores, supervisão de processo, teste de produtos etc. Muitas delas são realizadas por sistemas mínimos. Isso é importante, pois integrar estas soluções exige muitas vezes uma adaptação ou um aumento das capacidades dos dispositivos para incluir as funções de integração ao sistema.

- Requisitos de tempo: aplicações de controle são aplicações de tempo real, pois devem monitorar ou agir sobre o sistema em tempos bem definidos e com tolerâncias pequenas. Por exemplo, um certo conjunto de chaves pode ser monitorado a cada segundo, entretanto o tempo de ocorrência de um fechamento ou abertura de uma chave é da ordem de milissegundos; assim, mesmo que o evento “monitoração completa nas chaves” seja realizado em um tempo relativamente alto, a monitoração de cada uma das chaves possui uma janela de eventos bem restrita e definida.

- Requisitos de processamento e memória: algumas aplicações de controle operam em sistemas mínimos e outras em PCs industriais. A necessidade de processamento e memória é função dos algoritmos de controle utilizados e para o que ele será utilizado. Controle de motores exige muita velocidade e, portanto, muita capacidade de processamento, enquanto sistemas de testes exigem uma certa quantidade de memória para o armazenamento temporário dos resultados dos testes.

- Flexibilidade: sistemas de controle não necessitam ser muito flexíveis pois são criados para aplicações específicas, entretanto eles devem ser facilmente programáveis, para permitir ajustes durante o processo produtivo, dentro dos limites impostos pelas especificações do sistema a esses ajustes.

- Robustez: sistemas de controle devem ser inerentemente robustos, até por questões de segurança. Em presença de falhas o sistema deve permitir algum tipo de interação com processo, mesmo que seja a parada do processo. Também ele deve permitir o

seu uso em situações ambientais muito diversas e exigentes do chão-de-fábrica. Vale lembrar que o custo de uma linha parada é muitas vezes superior ao custo de se fazer um sistema robusto que a opere.

3.1.2.2. Aplicações de Engenharia Industrial

São as aplicações de suporte a produção, isto é, aplicações que realizam atividades como acompanhamento de processo e manutenção de equipamentos. Exemplo de aplicações: Monitoração e Manutenções de Remotas, Indicador de Produção etc.

- Requisitos de tempo: aplicações de engenharia industrial são aplicações de tempo real, pois devem monitorar ou agir sobre sistemas de tempo real do processo produtivo. Entretanto, este tipo pode ser qualificado de *soft real time*, pois o tempo entre os eventos e as suas tolerâncias são maiores que os sistemas do sub-tópico anterior, admitindo-se alguma sobreposição entre eventos (veja-se um exemplo no próximo requisito).

- Requisitos de processamento e memória: são bem maiores que no sub-tópico anterior. Não raro uma parte das funções deste tipo de sistema está compartilhando o mesmo hardware de um sistema mínimo. Outra parte fica em hardware próprio, normalmente um PC ou estação de trabalho. Exemplo: num sistema de monitoração de remotas, uma das tarefas da remota é emitir um aviso de seu estado, se em produção ou em falha, para o sistema monitor; neste caso, o sistema de alarme fica na remota e o monitor fica em hardware próprio, por exemplo um PC. Percebe-se claramente, no exemplo, que os tempos aqui ainda são rígidos, isto é, as janelas de

eventos são bem definidas mas já admitem uma certa sobreposição, isto é, o PC pode possuir um sistema de captura das informações emitidas pelas remotas, mas pode colocá-las dentro de uma fila para que um único processo tratador possa realizar o seu trabalho; o tempo máximo permitido para isto será o tempo de ocorrência de todos os eventos conjuntamente sem a sobreposição de dois eventos da mesma remota, mas podendo haver a sobreposição de vários eventos de remotas distintas, os quais vão para a fila no caso desta ocorrência.

- Flexibilidade: devem ser sistemas bastante flexíveis, principalmente do ponto de vista do software. Quaisquer alterações no processo produtivo devem ser acompanhadas por estes sistemas. Na prática somente possuem uma flexibilidade razoável, dados os custos da manutenção de equipamentos neste nível que, para algumas empresas, não é tão prioritária quanto nos sistemas de gestão.

- Robustez: quando compartilhando o hardware com sistemas de controle são naturalmente robustos. Entretanto, algum nível de falha é permitido o que coloca o nível de robustez que deva ter em menor escala que no caso anterior. Admite-se que uma falha em um sistema neste nível não afetará irremediavelmente a produção.

3.1.2.3. Aplicações de Gerência

São os sistemas chamados de *back-end* da empresa, tais como ERP[15], MES[16] e GED[17] e servem para as funções de gerenciamento da planta. Estes sistemas são arquiteturalmente do tipo cliente/servidor; aqui será utilizado somente a visão do lado do servidor, que pode ser distribuído.

- Requisitos de tempo: aplicações de gerência não são aplicações de tempo real, pois seus tempos de resposta são do ponto de visto das relações humanas e suas tolerâncias são muito grandes. Então, os eventos podem ser colocados dentro de uma fila para a sua execução sem grandes problemas.

- Requisitos de processamento e memória: os computadores em que sistemas deste tipo são instalados devem possuir grandes capacidades de processamento e memória, visto que estão operando uma grande quantidade de informações, vindas de vários clientes e emitidas para vários clientes simultâneos. Exige uma memória de massa rápida, eficiente e que garanta a qualidade dos dados, sem perdas.

- Flexibilidade: devem ser flexíveis pois quaisquer alterações nos processos inferiores, podem provocar mudanças neste nível. Entretanto, na prática os sistemas são muito pouco flexíveis, tanto do ponto de vista do software quando do hardware, uma vez que é muito caro realizar quaisquer alterações neste nível. Então o que se faz é

15 ERP – Enterprise Resource Manager: sistema responsável por gerenciar os recursos da empresa, onde recursos são: materia-prima, elementos humanos, produtos acabados etc.

16 MES – Manufacturing Execution System: sistema responsável por gerenciar o processo de execução do processo produtivo

17 GED – Gerenciamento Eletrônico de Documentos: sistema responsável pelo armazenamento e controle de documentos eletrônicos e em meio físico da organização.

conviver com sistemas eternamente limitados e que não acompanham as mudanças ambientais. Por isso é tão comum, quando na presença de um atendente, frases do tipo: “eu entendo a sua solicitação e a acho aceitável, mas isso eu não posso fazer porque o sistema não permite”.

- Robustez: sistemas de *back-end* são sistemas servidores que devem operar no ritmo 24x7, isto é, 24 horas por dia, 7 dias por semana, sem intervalos. São chamados de dispositivos de alta disponibilidade ou de missão crítica. Dispositivos deste tipo devem ser robustos. Aqui o problema não será o ambiente ruidoso e com poeira do chão-de-fábrica e sim a necessidade da alta disponibilidade dos equipamentos.

3.1.2.4. Aplicações de Engenharia de Produto

São as aplicações de suporte ao desenvolvimento de produtos, isto é, são aplicações que realizam atividades como CAD, CAE, CAM. Exemplo de aplicações: sistemas de CAD e CAE para Projeto de PCIs ou Projeto Mecânico, Sistemas de Simulação e Testes de Esforços etc.

- Requisitos de tempo: são da ordem das relações humanas, portanto não são de tempo real.

- Requisitos de processamento e memória: a exigência de processamento e memória é intensiva. Simulações exigem um alto grau de processamento e memória. Na prática, os usuários aprendem a conviver com tempos não reais da simulação, dado que a infra-estrutura de hardware e software existentes normalmente não correspondem a essas exigências.

- Flexibilidade: sistemas de desenvolvimento de produtos devem ter uma flexibilidade apenas o suficiente para acomodar o avanço tecnológico. Não raro uma empresa tem que refazer investimentos para se manter atualizada, o que dificulta a popularização deste tipo de Engenharia nas empresas. Outro fator limitante para a manutenção de uma engenharia de produto é de ordem política: empresas multinacionais tendem a manter o projeto de seus produtos nas matrizes ou em centros de desenvolvimento em pontos estratégicos do mundo.

- Robustez: por não se tratar de atividades de missão crítica, nem de ambientes hostis, os requisitos são pequenos. Normalmente os dados críticos serão delegados para sistemas de missão crítica como uma ferramenta GED.

As aplicações de engenharia de produto não fazem parte do escopo deste trabalho dado que ele se baseia em um modelo em camadas plano, entretanto, um modelo de empresa integrada também necessita levar em conta tais atividades de desenvolvimento do produto. Isto gera a criação de camadas em profundidade no quadro de camadas mostrado na Figura 1, aumentando a complexidade da integração.

3.2. Características da Rede para Integração

Uma rede para integração deve ser construída mediante os tipos de aplicações a que dará vazão. Em um ambiente onde as aplicações são de tempo real a rede de comunicações entre os elementos a serem integrados deve permitir a troca de mensagens em tempos bem específicos, com pouca ou nenhuma retransmissão ou perda

de pacotes. Por outro lado, as diversas aplicações dos níveis superiores da fábrica requerem bem menos restrições de tal forma que pode-se utilizar redes com colisões e perdas de pacotes.

A topologia de rede mais utilizada no chão-de-fábrica é a topologia em barramento, isto é, os diversos dipositivos compartilham o mesmo cabo ou segmento físico de interligação. Na prática costuma-se derivar pequenos segmentos a partir do segmento principal. Isto é especialmente verdadeiro para a camada física RS-485.

Modernamente observa-se uma invasão da rede Ethernet em aplicações de coleta de dados no ambiente do chão-de-fábrica. Redes Ethernet possuem preponderantemente sua camada física ligada em estrela. Como se espera para breve o uso mais ostensivo desta tecnologia, a topologia mais comum se tornará a estrela.

As disciplinas de acesso ao meio nas diversas redes também dependem do tipo de aplicação que se tem em mente. Para aplicações de tempo real, a colisão e as retransmissões são evitadas ao máximo. Para isso constuma-se utilizar uma abordagem “mestre/escravo” ou “multi-mestre” com passagem de *tokens*.

Na ambordagem mestre/escravo, a rede possui um único dispositivo que é chamado de Mestre e que inicia a comunicação, enquanto os demais (os escravos) apenas respondem às solicitações do Mestre. Na abordagem multi-mestre, diversos dispositivos podem se tornar o Mestre da rede, desde que possuam um *token*. Um *token* é uma mensagem especial que trafega na rede e que todos os nós tomam conhecimento. Qualquer nó que “tomar posse” do *token*, emite uma mensagem dessa posse na rede e passa a ser o mestre.

Para aplicações que não são de tempo real, pode-se utilizar uma outra abordagem que é a chamada “comunicação por mudança de estado”, onde o dispositivo que quer se comunicar somente o faz quando possui dados novos a serem transmitidos.

Também ele o faz tão logo estes dados estejam disponíveis para transmissão. Fica claro que neste tipo de abordagem no acesso ao meio, a rede deve prover mecanismos para evitar ou detectar a colisão, como é o caso de redes Ethernet.

As redes mais comuns no ambiente industrial são:

- RS-232: há diversos sistemas legados que são ponto-a-ponto sobre RS-232. Também é possível encontrar sistemas Mestre/Escravo que utilizam tal tecnologia;
- RS-485: de longe é a camada física mais empregada em células de automação e acesso a remotas, utilizando basicamente uma metodologia mestre/escravo ou multi-mestres.
- Ethernet: é a rede mais empregada nos escritórios e nos níveis superiores das fábricas. Diversos estudos são feitos para permitir o uso de Ethernet nos níveis inferiores.

Em todas elas é possível implementar tanto uma abordagem mestre/escravo quanto uma abordagem de mudança de estado. Redes Ethernet são mais performáticas, isto é, possem uma maior taxa de transmissão de dados, enquanto que redes RS-485 são mais imunes a ruído. Para o desenvolvimento de sistemas de tempo real, com a opção de utilização de redes Ethernet, dada a sua naturza não determinística, deve-se tomar alguns cuidados para se evitar colisões, por exemplo através de uma abordagem mestre/escravo, de tal forma que ela se torne o mais determinística possível.

3.3. Características de uma Arquitetura para Integração

Uma vez caracterizado o ambiente industrial, em termos dos tipos de aplicações encontrados naquele ambiente e sua rede, pode-se apontar algumas características desejáveis em uma arquitetura para integração.

As seguintes características foram elencadas a partir do trabalho de [15]:

- Integração com dispositivos de campo inteligentes: um sistema de base para integração deve permitir o desenvolvimento de aplicações que interajam com dispositivos de campo inteligentes, isto é, dispositivos com capacidade de processamento, memória e comunicação. Isto permite que se desenvolva a partir desse sistema de base, um sistema de controle e supervisão da planta fabril o qual deverá realizar medições e ações sobre um processo produtivo. Uma característica a mais é a possibilidade de se fazer isso em tempo real.

- Permitir que diversos protocolos coexistam, quando viável: uma conseqüência da característica anterior é que o sistema de base deve permitir a construção de sistemas que utilizem diversos protocolos de comunicação, o que é necessário para a comunicação com os dispositivos de campo inteligentes de diversos fabricantes ou que entendem somente algum tipo de protocolo específico.

- Funções inteligentes intrínsecas à arquitetura: deve possibilitar a construção de sistemas com filtros para os dados, permitindo a sua contextualização prévia, próxima da fonte geradora dos dados, o que transforma o sistema em um gerador de informações. Esses filtros podem direcionar os dados brutos ou contextualizados para os sistemas de tratamento adequados, ou replicá-los, conforme a circunstância.

- Permitir o desenvolvimento de supervisório próprio: deve prover conjuntos de APIs para o desenvolvimento de aplicações completas, com IHM em navegadores da Web ou outros tipos, tratamento da modelagem de negócios, classes controladoras etc.

- Permitir a integração com supervisórios externos: esta característica acaba conflitando com a anterior, pois o uso de supervisórios externos inibe o desenvolvimento de ferramentas para supervisório próprio. Mas, como característica desejada, uma arquitetura para integração deve permitir a integração do sistema em desenvolvimento ao supervisório externo. Isto significa que ela deve prover o desenvolvimento apenas de aplicativos tipo “casca” (*wrappers*), o que significa que um supervisório externo pode ter acesso aos dados colhidos através do sistema desenvolvido na arquitetura transparentemente. De fato, o supervisório externo não deve diferenciar se está se comunicando com um dispositivo de campo ou se com um objeto do sistema dentro da arquitetura.

- Diversas capacidades de persistência: uma arquitetura para integração deve prever capacidades de persistência em diferentes níveis, desde a ligação com banco de dados de terceiro como um SGBDR ou banco de objetos, até o armazenamento em sistemas de arquivo em memórias flash, dentro de sistemas mínimos. Uma outra forma é proporcionar o seu próprio sistema de persistência para minimizar o uso de outras ferramentas de persistência, em especial SGBDRs, quando estes não seriam tão relevantes para certos tipos de aplicações. Também deve ser transparente se o uso será de uma persistência de objetos ou relacional. Essa característica é de difícil implementação, pois se o sistema for orientado a objetos há diversos problemas para a sua representação dentro de uma base relacional.

- Possibilitar serviços básicos: a arquitetura deve prover: gerenciamento dos recursos físicos do processo; controle do estado operacional das remotas; mudança do modo de operação das remotas; monitoração de condições e manipulação de exceções;

- Extensibilidade de serviços: deve permitir que o desenvolvedor estenda os serviços disponíveis na arquitetura e flexibilidade na adaptação dos componentes existentes ou inclusão de novos componentes.

- Redes heterogêneas: deve permitir seu uso em diversos tipos de redes. É uma característica também derivada da segunda característica anteriormente citada. Notar que um hardware para cada tipo de rede também deve estar disponível.

- Tratamento de sistemas legados: deve permitir a construção de aplicativos que interajam com sistemas legados e/ou a criação de aplicativos tipo “casca”.

- A arquitetura deve ser aberta: o que permite uma maior disseminação da tecnologia e que um maior número de profissionais possam lhe dar suporte.

- Fácil de programar: a fim de minimizar o tempo de elaboração do código da solução.

- Fazer uso de diferentes linguagens: obtendo-se o melhor que uma dada linguagem de programação pode fornecer, bem como manter o *expertise* da equipe.

Capítulo 4

O PROXY DAS REMOTAS

A Internet tem modificado as relações entre clientes e fornecedores. Isto é um fato. A questão é: a manufatura está apta à responder com a velocidade da Internet e seus novos paradigmas? Acredita-se que ainda existam alguns problemas de integração pendentes, alguns dos quais foram elencados nos capítulos anteriores, resumidamente aqui colocados:

- em uma fábrica há inúmeras tecnologias convivendo o que gera problemas de fragmentação da tecnologia e baixa qualidade dos sistemas com vistas a integração;
- o desenvolvimento de novos softwares sem incluir requisitos de intgração, seja atual ou futura e a ausência de um modelo para integração gera aplicações que depois, quando se tornarem legadas, são de difícil manutenção; e
- sistemas anteriormente existentes em uma fábrica devem ser reutilizados em um ambiente integrado;

O desafio maior, portanto, é a integração dos diversos sistemas da planta fabril a fim de que os sistemas de gestão possam realizar a correlação dos dados e seus contextos num ambiente orientado a conhecimento.

Neste contexto, dentro da arquitetura do Proxy, há a possibilidade do desenvolvimento de um *middleware*, isto é, uma camada de software que permita a implementação dos dicionários de dados de cada um dos mundos, o de gestão e o do controle. Como visto, este é um dos problemas encontrados para a execução de um processo de integração fabril.

Para o desenvolvimento de uma solução em automação, deve-se procurar utilizar sistemas de base que permitam uma maior facilidade na construção de sistemas integrados, em especial sistemas novos. Entretanto, muitos sistemas que já existem e estão em operação necessitam de uma atualização para permitir a integração. Também existem sistemas legados que não admitiriam nem mesmo uma atualização para fins de integração.

Para sistemas legados, uma solução possível está na criação de sistemas tipo casca (sistemas *wrappers*, no jargão em inglês), isto é, sistemas que possam interagir com os sistemas antigos de tal forma que os demais sistemas "pensem" que estão se comunicando com os sistemas antigos transparentemente.

Também não há tempo ou recursos disponíveis para novas soluções quando existem sistemas que já estão em operação de forma satisfatória, o que leva ao desenvolvimento de pontes ou *gateways*, isto é, sistemas de interface para permitir a interoperabilidade entre dois sistemas existentes, eventualmente entre duas redes distintas.

Para a implementação do Proxy de Remotas utilizou-se a linguagem Java sobre um container da Web. A escolha de Java se deu pelos seguintes motivos:

- É uma linguagem recente, portanto, sintaticamente mais limpa que C/C++, fácil de aprender e semanticamente consistente com a orientação a objetos;

- é multiplataforma, isto é, pode ser utilizada em diversos sistemas operacionais e tipos de processadores, o que é muito interessante para um ambiente tão heterogêneo quanto o ambiente fabril; de fato há diversas Máquinas Virtuais Java (JVMs na sigla em inglês) já implementadas para a maioria das plataformas utilizadas em sistemas mínimos.
- apesar do código-fonte da máquina virtual Java, fornecida pela Sun, ou sua especificação não serem softwares livres, há uma grande comunidade que desenvolve uma grande quantidade de APIs (Interface de Programação de Aplicações) em Java que são livres, o que minimiza os esforços e custos no desenvolvimento[18];
- há inúmeras implementações dos principais padrões internacionais de comunicação e interoperabilidade. E esse foi o principal motivo da escolha: um container da Web e um *parser*[19] livre XML podem ser baixados da Internet e, em poucas horas de trabalho, um servidor Web que pode realizar tratamento dos dados vindos das remotas está no ar.

Observar que outras linguagens também poderiam ter sido utilizadas, por exemplo: C# com componentes DCOM sobre a tecnologia .NET pode ser uma alternativa. Não implementamos esta alternativa por se tratar de tecnologias proprietárias, o que vai de encontro ao objetivo de desenvolver um sistema aberto.

No restante deste capítulo apresenta-se a concepção e a implementação de uma solução para um problema real de integração no chão-de-fábrica que incorpora os aspectos discutidos anteriores, na medida em que se mostrem apropriados. Esta solução

18 Existem máquinas virtuais para Java que são livres (ver Projeto GNU: www.gnu.org)

19 Veja *www.apache.org* para maiores detalhes

realiza tanto a interface entre dois sistemas em produção como a criação de aplicativos tipo casca. Ela também deve facilitar a criação de aplicativos tipo última-milha, para o chão-de-fábrica ou para a Engenharia Industrial considerando desde o início de sua concepção, o requisito de permitir e facilitar a integração.

Esta solução foi chamada de Proxy de Remotas e, para descrevê-la, nos tópicos seguintes discute-se os seus elementos arquiteturais e suas possíveis implementações, inclusive em dispositivos mínimos. Este capítulo finaliza descrevendo uma implementação que já está em produção, bem como alguns dos estudos feitos em laboratório.

4.1. Arquitetura do Proxy de Remotas

O Proxy de Remotas pode possuir dois elementos distribuídos: um deles pode residir em um computador mestre mais completo como um PC Industrial e o outro elemento pode residir em um dispositivo mínimo. Neste tópico descreve-se os elementos arquiteturais do Proxy de Remotas e como eles podem ser implementados no PC Industrial. O tópico seguinte mostra o Proxy de Remotas em dispositivos mínimos. Utilizamos nos exemplos de implementação, softwares livres, por acreditar que sejam mais fáceis de se obter e de se verificar o trabalho efetuado.

Pode-se ver, na Figura 3, os elementos arquiteturais do Proxy de Remotas. Os sub-tópicos abaixo descrevem cada um deles com detalhes.

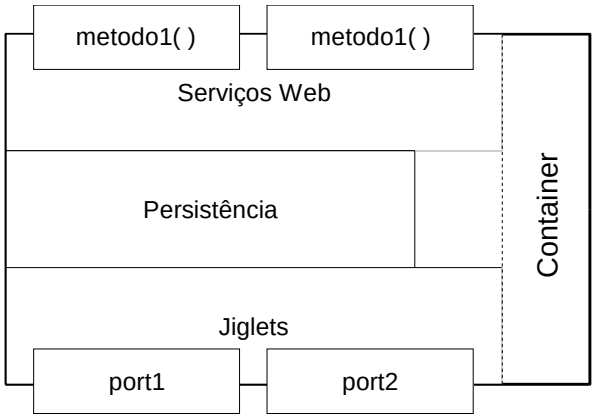


Figura 3: Arquitetura do Proxy de Remotas

4.1.1. Container

Um container é o elemento arquitetural que forma a infra-estrutura de base para o desenvolvimento das aplicações de integração. A função do container é ser um elemento gerenciador dos componentes de software que fazem parte da arquitetura do Proxy de Remotas: os *ports* e os *jiglets*.

Um container deve ter uma maneira de gerenciar e se comunicar com os componentes que irá possuir. Isto é conseguido através de um interface bem definida e padronizada. Uma abordagem de implementação seria utilizar um container do tipo Inversão de Controle (Inversion of Control, IoC na sigla em inglês), isto é, um container que gerencia o ciclo de vida dos componentes de software que são desenvolvidos com o padrão de projeto IoC. O ciclo de vida deve ser um conjunto padrão de APIs e interfaces que permite a instanciação, configuração, chamada do serviço, finalização etc., de um componente.

Um exemplo de um tal container é o Excalibur, da Apache Software Foudation (ASF)[20], que implementa as definições e interfaces do Avalon Framework, também um projeto da ASF.

20 Veja http://www.apache.org para maiores detalhes

Pode ser utilizado igualmente um container Web, pois este também gerencia componentes: os *servlets*, e possuem controle do seu ciclo de vida.

Outra alternativa seria a criação de seu próprio container.

Para aplicações de tempo real sugere-se o uso de um container IoC, para o caso em que performance não seja tão importante, ou a criação de um container próprio, no caso de necessidade de performance. Para uma implementação em Java, deve-se utilizar uma JVM que adere ao *Java Real-Time Spec*. Para aplicações que não são de tempo real, pode-se usar, sem nenhum problema, um container da Web.

Um container para uma aplicação de tempo real também pode ser construído como um servidor multitarefa diretamente com soquetes TCP ou UDP. A interface entre componentes poderá ser feita via mensagens TCP ou UPD que serão padronizadas para a aplicação. Os componentes podem ser *threads* (processos leves) dentro do container e que serão instanciados quando da chegada ou envio de uma mensagem. A instanciação, disparo de *threads* e disseminação de mensagem entre os processos fica a cargo do container. O parâmetro a ser passado a *thread* pode ser a mensagem. Em laboratório chegou-se a implementar um tal container para fins de estudo, sem contudo tê-lo tornado robusto o suficiente para uma aplicação comercial.

4.1.2. Ports

Um *port* é um componente de software que realiza a comunicação direta com as remotas, isto é, ele é o responsável pelo protocolo de comunicação com a remota e torna transparente para o restante da arquitetura qual é a rede com a qual o dispositivo opera. Como é um componente, é instalado no container e deve sofrer os mesmos mecanismos de controle e gerência de ciclo de vida que os demais componentes da arquitetura sofrem. Entretanto, pode ser interessante que o componente dentro da

arquitetura nada mais seja que uma casca para uma aplicação externa. Visualiza-se tal necessidade quando o protocolo exige tempos tão específicos que a performance do gerenciamento de ciclo de vida não seja suficiente e possa prejudicar aqueles tempos.

Um *port* deve direcionar sua saída a um *jiglet*. Se ele for o gerador de algum evento de comunicação, também deve propagá-lo ao *jiglet*. Uma implementação de um *port* somente tem sentido se há uma diferença entre o protocolo utilizado para a comunicação com as remotas e a interface com um *jiglet*. Por exemplo: se um *jiglet* for implementado como um servlet dentro de container da Web e as remotas também estiverem sobre a Web, isto é, utilizam o protocolo HTTP para comunicação, não há necessidade de um *port*, pois que o container Web deve prover este mecanismo de comunicação por padrão. Entretanto, se a remota implementar um protocolo proprietário, deve-se então criar um *port* que mapeie aquele protocolo com chamadas de método ou mensagens equivalentes para os *jiglets*. Isto é mostrado na Figura 4.

Um exemplo pode tornar tudo mais claro: uma remota utiliza RS-485 como camada física e ModBus como protocolo de aplicação, enquanto outra utiliza TelNet sobre TCP/IP sobre Ethernet. Para ambos os casos há a necessidade de *ports*. Todos podem ser implementados dentro do container, cada um como um componente separado e cuja interface com o *jiglet* pode ser padronizada com chamadas HTTP sobre TCP/IP.

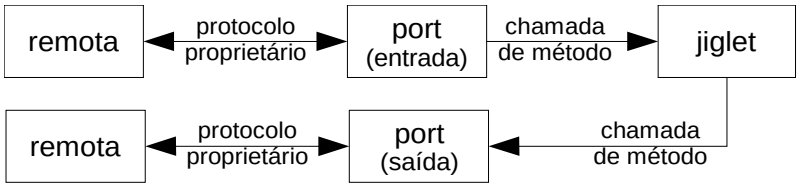


Figura 4 - Visão de Entrada e Saída de *ports* e *jiglets*.

Na Figura 4 pode-se ter uma visão de entrada e saída para *ports* e *jiglets*, isto é, ela mostra a função de um *port* para recepção de dados enviados por uma ou mais remotas e realizando chamadas de métodos do *jiglet*. Este por sua vez, pode igualmente emitir dados para as remotas, chamando métodos padronizados em um *port* de saída e este realiza a comunicação com a remota.

Um *port* pode ser implementado como um *servlet* (formalmente um GenericServlet) dentro de um container da Web para compatibilizar algum protocolo proprietário, que não HTTP. Também pode ser implementado como um processo leve em um container criado *do zero*, ou em um container IoC. No primeiro caso, o *port* deve realizar um redirecionamento para o *jiglet* ou uma chamada HTTP Request/Response para a implementação do *jiglet*. No segundo caso, o *port* pode fazer uma chamada de método, diretamente ou através do container.

4.1.3. Jiglets

Um *jiglet* é um componente de software que realiza o processamento principal do Proxy de Remotas, isto é, é nele que a lógica da aplicação é realizada. Um *jiglet* pega dados de sua entrada padronizada de um *port*, processa-os e leva os dados de saída a um *port* de saída, ou à camada de persistência. A partir de chamadas de um serviço Web, o *jiglet* serve como um fornecedor de dados. Isto é mostrado na Figura 5.

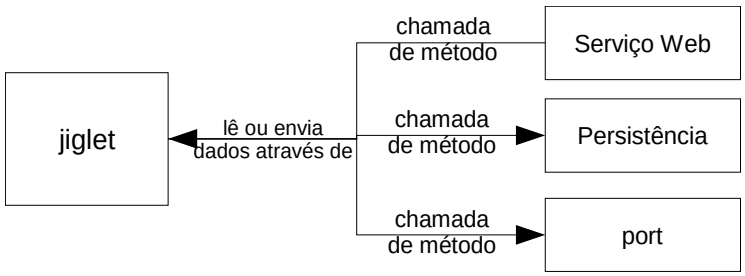


Figura 5- Relacionamentos de um *jiglet*

Como é um componente, ele é instalado no container e deve sofrer os mesmos mecanismos de controle e gerência de ciclo de vida que os demais componentes da arquitetura sofrem, isto é, as chamadas de métodos mostradas na Figura 5 podem ser feitas indiretamente através do container, onde o *jiglet* faz uma chamada ao container e e este faz a chamada ao componente de destino.

Funções inteligentes, atualização de banco de dados etc., são todas funções que são implementadas como *jiglets*. Entretanto, para a melhor codificação destas lógicas deve-se levar em conta algumas técnicas da engenharia de software.

Uma delas é a Separação de Responsabilidades, isto é, dividir a aplicação em diversos componentes que possuem, cada um, uma tarefa específica a cumprir no sistema. Uma das divisões de responsabilidades possível é através do padrão MVC (o padrão de projeto MVC é mostrado na Figura 6), sendo as tarefas de uma das seguintes categorias[21]:

- Modelo (*Model*) – onde o domínio da aplicação é retratado;
- Visualizador (*View*) – onde as interfaces da aplicação são implementadas;
- Controlador (*Controller*) – onde a integração entre os elementos do modelo e de visualização é feita sob um conjunto de regras bem estabelecidas.

21 Esta divisão faz parte do padrão de projetos MVC, o qual foi escolhido para orientar a arquitetura do sistema. Há inúmeros outros padrões que podem e devem ser seguidos no desenvolvimento dos sistemas finais.

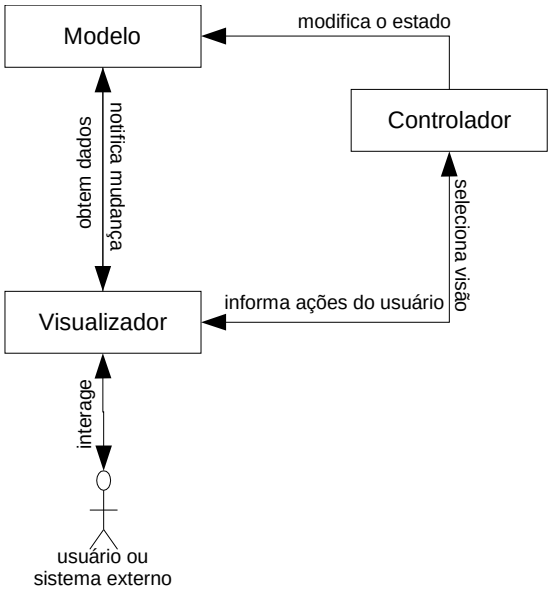


Figura 6 - Padrão de Projetos MVC

No padrão MVC, um controlador inicia tanto o modelo quanto o visualizador, selecionando a visão padrão do sistema. O usuário ou sistema externo interage com o a visão escolhida, dentro do bloco Visualizador, que informa ao Controlador as ações que foram efetuadas pelo usuário. Então o controlador dispara duas novas ações: alteração dos dados no Modelo e seleção de nova visão (caso necessário). O Modelo, por sua vez, notifica ao Visualizador que houve uma mudança nos dados. Isso faz com que a visão atual mostre os dados mais recentes.

Dentro do Proxy de Remotas, um *port* pode ser utilizado como Visualizador, isto é, mantém uma interface apropriada para interações com usuários ou sistemas externos. Um *jiglet* pode implementar o Controlador e um outro *jiglet* o Modelo.

Dentre as funções dos *jiglets*, a mais básica é o reflexo das características da remota como um objeto distribuído na rede. Com isso as remotas podem ser vistas dentro da arquitetura como objetos.

Alguns leitores poderão estar recordando do conceito de JavaBeans da tecnologia Java. Em princípio, também poderia ser utilizada uma tecnologia como JavaBeans para representar remotas como objetos, entretanto esta tecnologia carece de

algumas características básicas: não possui mecanismos de acesso remoto aos seus métodos, de forma que necessitaria de um outro objeto que se lhe fizesse a ponte; também não permitiria um controle de ciclo de vida, a não ser dentro de um ambiente de desenvolvimento rápido de aplicações em que os JavaBeans podem ser tratados como componentes o que, neste caso, ainda necessitaria de uma classe meta-dado para aquele ambiente, mas esta característica se torna muito difícil de ser replicada em tempo de execução. Também ficaríamos fechados em uma solução puramente em Java. Portanto, houve a necessidade de se construir este novo conceito: o de *jiglet*.

Um *jiglet*, como um *port*, pode ser implementado como um *servlet* ou um processo leve em um servidor multitarefa, dependendo de como será a escolha do acesso às remotas e como será feita a representação da remota dentro da arquitetura. Outras tecnologias podem suprir as características necessárias para a implementação de um *jiglet*.

Veja-se um exemplo de como representar as remotas dentro da arquitetura: um *jiglet* pode ser instanciado para cada remota e seus dados refletem o estado operacional da remota. Então se quisermos ler um relatório de testes da remota, o Controlador pode simplesmente chamar um método: *remota*.getRT(), onde *remota* é o nome do objeto que representa a remota real. Esta chamada esconde os detalhes de implementação de como aquele relatório de testes será obtido, provavelmente passando-se alguma mensagem para um *port* e este se comunicando diretamente com a remota real.

Observar que um *jiglet* possui métodos que podem ser acessados remotamente, o que significa que a arquitetura se torna eminentemente distribuída. Daí a necessidade de ser construída sobre tecnologias como COM ou RMI. Também alguns métodos dos *jilets* podem ser externados como Serviços Web.

4.1.4. Serviços Web

O Word Wide Web Consortium (W3C)[22] define um Serviço Web[23] como um software desenvolvido para realizar interações entre máquinas. A interface de um Serviço Web é descrita através da linguagem WSDL e os outros sistemas interagem com o serviço através de mensagens SOAP, normalmente sobre HTTP. Tanto SOAP quanto WSDL são descritas em termos de uma aplicação XML. Em outras palavras, um Serviço Web é um software que consegue trocar mensagens através de padrões bem estabelecidos e, como não faz distinção em qual linguagem os elementos que estão se comunicando foram implementados, é independente da linguagem de programação.

Na arquitetura proposta para o Proxy de Remotas, os serviços Web são objetos que externam alguns ou todos os seus métodos através de um servidor SOAP (um servidor de serviços Web). Estes serviços formam uma camada que é a interface do sistema para as aplicações externas, de supervisão, controle e gestão, desde que estas aplicações não estejam sendo desenvolvidas dentro dos *jiglets*. Nesta camada o programador necessitará criar um Serviço Web que busca as informações necessárias através de chamadas à camada de persistência e as externe para a aplicação chamadora do serviço. Outra alternativa seria externar alguns métodos dos próprios *jiglets*, daí o ponto de contato entre os Serviços da Web e os *jiglets* como mostrado na Figura 3.

22 Veja http://www.w3c.org.

23 Vamos utilizar Serviços Web como uma tradução direta de *Web Services* do inglês.

Como o protocolo SOAP prevê o trânsito de mensagens em XML puro, soluções XML, como o padrão B2MML[24] ou VIML[25], também são possíveis, o que dá a arquitetura mais flexibilidade.

O ponto fraco para o uso de serviços Web é a performance. Como há uma série de processamentos que devem ser feitos nas duas pontas para a comunicação existir, há certos atrasos na rede, os quais podem não ser compatíveis com o ambiente industrial. Outra possibilidade seria o uso de CORBA, que é um pouco mais performático que os serviços Web, mas ainda é independente da linguagem. A perda de performance é a desvantagem da generalização e independência da linguagem.

Em casos que a performance seja fundamental, pode-se optar por outra tecnologia de acesso a procedimentos remotos, que não serviços Web, como por exemplo, RMI. Entretanto fica-se sujeito, neste caso, à linguagem Java.

4.1.5. Persistência

Uma arquitetura para integração deve prover acesso a algum tipo de persistência, própria ou externa. Isto é conseguido através do uso de um padrão de projeto chamado DAO (Data Access Object). Uma extensão deste padrão é conhecido como DAO com Fábrica (*DAO with Factory*, no jargão em inglês) cujo diagrama UML é mostrado na Figura 7. Neste padrão de projeto, um objeto, desenvolvido para realizar

24 B2MML: Business To Manufacturing Markup Language é uma implementação em XML do conjunto de padrões ISA-95, também conhecidos como IEC/ISO 62264, os quais são modelos de dados para comunicação entre os sistemas de controle e sistemas de gestão. Este padrão está disponível a partir da WBF – World Batch Forum: http//www.wbf.org.

25 VIML: Virtual Instrument Markup Language, http://viml.org acessado em junho de 2005. Vide também http://www.xml.org, o site oficial da tecnologia XML, da OASIS a qual mantém este portal para fins de busca de recursos e aplicações padronizadas em XML.

acesso aos dados, encapsula toda a complexidade do acesso, seja ele feito através de um banco de dados nativo em XML ou um SGBDR, através de uma interface bem definida, como por exemplo os métodos `add()`, `remove()`, `get()` e `set()`.

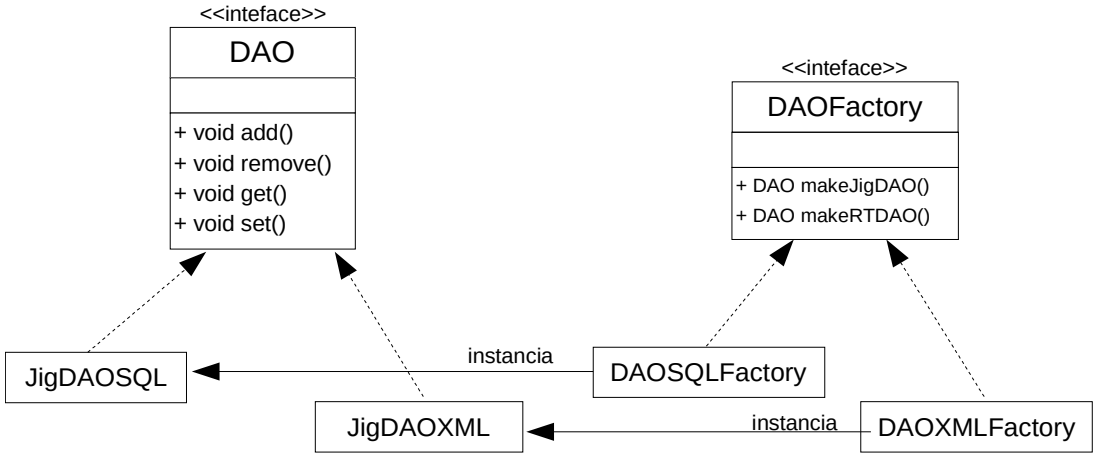


Figura 7: Padrão de Projeto DAO com Factory

Entretanto um objeto fábrica é o responsável pela instanciação do objeto de acesso a dados. Então a aplicação pode ser configurada para usar uma ou outra tecnologia e, em seu código de inicialização, pode conter a chamada para `makeDAO()` a partir de uma ou outra fábrica, como no exemplo mostrado na Figura 7, onde uma instância da classe `DAOXMLFactory` instancia um objeto `JigDAOXML` e uma instância da classe `DAOSQLFactory` intancia um objeto `JigDAOSQL`.

A arquitetura do Proxy de Remotas visa exatamente essas características, portanto foi a escolha mais lógica o uso desse padrão de projeto.

A implementação da persistência foi originariamente feita em um banco de dados nativo em XML, no laboratório; em produção está sendo utilizado um banco de dados SQL embutido, o HSQLDB, o mesmo que é utilizado como motor do banco de dados da suite livre de escritórios OpenOffice.org. A mudança foi motivada pela

estabilidade do banco de dados SQL. O fato deste banco ser implementado em Java e rodar embutido no container da Web também ajudou no quesito performance, uma vez que optamos também por implementar o Proxy de Remotas em Java, enquanto que no laboratório estes itens não são tão relevantes.

Entretanto, uma solução de persistência em XML é mais elegante para uma aplicação que foi pensada em troca de mensagens em XML e cuja representação de dados é também em XML. Fica claro que outras tecnologias de persistência também poderiam ser utilizadas.

A mudança de uma ou outra tecnologia de armazenamento não interfere no restante do sistema que utiliza as chamadas aos métodos do DAO para fazer acesso aos dados. Por exemplo, se o sistema for implementado em Java, nem mesmo é necessário uma recompilação do código, apenas a colocação das classes fábrica e DAO corretas e uma pequena configuração no ambiente é suficiente para a mudança da tecnologia.

4.2. O Proxy de Remotas em Dispositivos Mínimos

Quando se estudou a integração nos níveis inferiores da empresa, deu-se ênfase em dispositivos mínimos, pois que eles são os elementos que tornam possíveis a inclusão de uma certa inteligência no processo produtivo. Daí a necessidade de se desenvolver uma arquitetura de integração que possa ser utilizada também em dispositivos mínimos.

Como se trata de uma arquitetura de objetos distribuidos, o Proxy de Remotas pode possuir dois elementos distribuidos: um deles pode residir em um computador mais completo, tal como um PC industrial, e o outro elemento pode residir em um dispositivo minimo. Este ultimo elemento é o objeto deste tópico.

Em situações, em que pode haver por exemplo a necessidade de persistência no dispositivo mínimo, um objeto deve ser implementado de tal forma que seja transparente para a aplicação no dispositivo mínimo se ela está fazendo acesso ao sistema externo (a parte do Proxy de Remotas no mestre) ou a um módulo do sistema interno (a parte do Proxy de Remotas no dispositivo mínimo).

É claro que este tipo de aplicação em dispositivos mínimos requer uma certa capacidade de processamento e memória e, especificamente, uma memória de massa relativamente confiável, tal como uma memória *flash*.

Por se tratar de dispositivos mínimos, o desenvolvimento de aplicações de integração deve portanto passar pela elaboração de uma arquitetura de software multiplataforma. Como arquitetura, também deve possuir requisitos minimalistas para uso de processamento e memória. O Proxy de Remotas se propõe a ser uma arquitetura deste tipo, que pode ser utilizada tanto em dispositivos mais potentes (com maior capacidade de processamento e memória) como em dispositivos mínimos.

Para o desenvolvimento em dispositivos mínimos, a idéia central da arquitetura também é preservada, isto é, os elementos são aqueles mesmos mostrados na Figura 3. Nos subtópicos abaixo, demonstra-se como estes elementos arquiteturais poderão ser implementados em um dispositivos mínimos.

4.2.1. Container

Um container é o elemento que controla o ciclo de vida dos demais componentes do sistema. Logo, em dispositivos mínimos ele seria o responsável pelo interfaceamento entre os diferentes componentes. Por questões de performance e como tudo estará sendo compilado como um único bloco na maior parte das vezes, o container deverá utilizar um mecanismo rápido de comunicação com os componentes que fazem

parte do sistema. Se um sistema operacional de tempo real (RTOS) for utilizado, muito provavelmente será um mecanismo de sinal fornecido pelo próprio RTOS. Se não for utilizado um RTOS o mecanismo pode ser simplesmente a chamada de funções de interface.

O fato de haver uma montagem única de arquivos objetos, gerando um único arquivo de distribuição para o dispositivo mínimo quebra o conceito de componente de software, uma vez que este exige uma distribuição independente. No contexto dos dispositivos mínimos, será utilizado um conceito mais intuitivo: um componente será tão somente uma parte da aplicação com características próprias e interface bem definida com os demais elementos do sistema, provavelmente com os arquivos fontes sendo desenvolvidos separadamente do restante da aplicação.

Como uma aplicação em um dispositivo mínimo está muito associada ao hardware em que ela roda, no container também estará implementado o conjuto de funções de acesso hardware. Resumidamente, ele será responsável pelo ciclo de vida dos demais componentes, pelo acesso hardware e, provavelmente, deverá ser construído sobre um núcleo (sistema operacional) de tempo real.

4.2.2. Ports

O conceito de *port* se mantém em dispositivos mínimos, isto é, ele é o elemento responsável pela comunicação com o mundo exterior provendo um mecanismo padrão para a comunicação se realizar. Como tal, um *port* deve ser desenvolvido de acordo com a especificação do hardware de comunicação utilizado.

Não se deve confundir o conceito de *port* com o uso das APIs normalmente fornecidas pelos ambientes de desenvolvimento integrado que acompanham os hardwares de dispositivos mínimos. De fato, na implementação de um *port* se fará uso

extenso dessas APIs, se elas existirem. Entretanto um *port* se caracterizará fundamentalmente por receber os dados de um *jiglet* a partir de uma interface padronizada e enviá-los sob o formato necessário do hardware de comunicação disponível. O inverso também será realizado, isto é, ele tratará um evento hardware de comunicação e os dados recebidos externamente serão repassados a um *jiglet* sob uma forma padronizada.

Como na imensa maioria dos casos haverá somente um[26] tipo de interface de comunicação disponível no hardware, somente haverá a necessidade de se utilizar um único *port*.

4.2.3. Jiglets

Os *jiglets* também permancem com o mesmo conceito dentro de dispositivos mínimos, isto é, eles são os responsáveis pela lógica da aplicação, fazendo chamadas as APIs de acesso hardware do container, recebendo e emitindo dados para o *port*. Eventualmente enviará os dados para uma persistência interna ou externa (via *port*).

Apesar de a lógica da aplicação ser específica, no contexto dos dispositivos mínimos a engenharia de software poderia ajudar e muito o desenvolvedor em seu trabalho; entretanto, por questões de performance e minimização do tamanho do código, são bem pouco utilizadas tais técnicas de programação. Nota-se também que o profissional que normalmente desenvolve tais software são, por formação, da área de engenharia elétrica ou mecatrônica, de tal forma que poucos tem a capacitação adequada em utilizar tais técnicas. Este ponto é importante e este trabalho vem também esclarecer a este grupo de profissionais.

26 Isso para comunicação com vistas a integração com níveis superios, visto que os dispositivos mínimos se caracterizam por ter vários tipos de interfaces, como entradas e saídas analógicas e digitais, freqüencímetros, contadores, SPP, I2C etc., para acesso aos serviços do nível mais baixos do modelo em camadas utilizado.

A arquitetura do Proxy de Remotas poderia ser utilizada e, até mesmo, uma técnica tipo MVC e padrões de projeto como IoC, para programação de dispositivos mínimos. Mas as características aqui apresentadas diferem bastante da filosofia atual de desenvolvimento, pois os engenheiros eletricistas e eletrônicos costumam utilizar um método *bottom-up*, isto é, primeiramente ele pensam em partes pequenas, como o acesso hardware a cada um dos dispositivos físicos e depois vão acrescentando as funcionalidades. Por outro lado, os profissionais de computação constumam utilizar um processo *top-down*, onde primeiramente se pensa a arquitetura do sistema como um todo, depois divide-se o mesmo em pedaços menores para a sua execução.

Outra diferença interessante entre estes profissionais é que o programador de dispositivos mínimos dificilmente se conformaria com o fato de ter que fazer várias chamadas a um container a fim de fazer um acesso a um determinado componente, enquanto ele pudesse fazer a chamada à função alvo diretamente. Desta maneira, a performance é máxima enquanto na primeira maneira a performance fica sofrida. Entretanto, deixar que a função alvo seja escolhida pelo container implica em podermos substituir completamente a implementação do componente destino sem qualquer necessidade de reprogramação do componente chamador e, se um processo de distribuição de binários independentes estiver presente, nem mesmo há necessidade de se recompilar o código do componente chamador.

4.2.4. Persistência

A persistência em dispositivo mínimos vem aumentando em importância nos últimos anos, dado o seu barateamento. Entretanto é bom se esclarecer que sempre se trata de uma persistência temporária e “muito” limitada.

Para realizar a persistência em dispositivos mínimos recorre-se a memórias em estado sólido, tais como memórias *flash*, ou NVRAMs. Um pequeno componente de software faz a interface entre o hardware e o restante do sistema. Este componente é conhecido como Sistema de Arquivo, isto é, a maneira como a memória é mapeada e acessada. Funções de interface são criadas e formam uma API, que são anexadas ao RTOS e ficam disponíveis para o programador.

Para fins do Proxy de Remotas, não há necessidade do uso de um padrão de projetos como DAO com Fábrica, uma vez que o hardware e o sistema de arquivos jamais irão mudar (de fato, se o hardware mudar o código de acesso hardware deverá mudar igualmente, necessitando-se realizar um tedioso trabalho de recodificação). O acesso a esta funcionalidade pode ser feito diretamente por chamadas a API do sistema de arquivos dentro da lógica do *jiglet*.

Para uma modularidade maior do sistema, pode-se usar um outro *jiglet* ou mesmo um DAO puro (sem fábrica) para acesso à persistência, enquanto os demais *jiglets* cuidam exclusivamente da lógica.

Uma outra abordagem é colocar um DAO puro dentro de um *port*, isso faz com que os *jiglets* acessem uma persistência interna ou externa transparentemente, isto é, eles simplesmente enviam dados para um *port* e este se encarrega de transmitir para uma persistência externa, no caso de a rede estar ativa, ou para uma persistência temporária interna, no caso de a rede estar em falha.

4.3. Exemplo de Implementação do Proxy de Remotas

Abaixo será descrita uma alternativa de implementação do Proxy de Remotas. Entretanto, outras alternativas também serão indicadas. Nota-se que as discussões sobre como implementar uma alternativa do Proxy de Remotas também indica quais os

problemas de projeto e codificação de tais soluções e suas limitações práticas. A implementação aqui descrita ocorreu para uma indústria de displays de telefones celulares e está colocada aqui apenas como um exemplo.

4.3.1. Características do Sistema

O sistema existente, que não tinha nenhum requisito de integração era composto de remotas que realizavam testes (jigs) sobre um dispositivo sob teste (DUT). Estes jigs operavam fora da rede, isto é, eram ilhas de automação. Os dados a respeito da qualidade do produto sendo testado era visualizado pelo operador do jig, através de uma codificação, e este etiquetava o DUT que eventualmente apresentasse defeito com aquela codificação. O DUT era retirado da linha e posteriormente feito um retrabalho sobre ele.

Cada linha de produção operava com no mínimo dois postos de testes, um para testes funcionais, o que era realizado na base de teste em 100%, e outro para testes de garantia da qualidade, ao final da linha, na base de teste em amostras, o que pode ser visualizada na Figura 8.

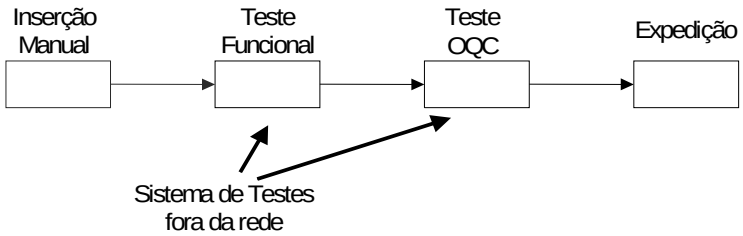


Figura 8: - Sistemas de teste não integrado

O sistema de coleta de dados era manual, onde um operador recolhia os dados sobre os DUTs com defeito ou retrabalhados de hora em hora. O sistema MES da companhia era alimentado com estes dados já sumarizados pelo operador, através de uma planilha eletrônica que continha a formulação estatística da massa de dados recolhida. Quaisquer ações sobre o processo somente poderiam ser executadas quando os engenheiros fossem alertados pelo MES ou pelos operadores nas linhas de produção, que percebiam erros em série muito antes do sistema.

Dentre as características desejadas para o novo sistema, indica-se:

- que os jigs fossem colocados em rede de tal forma que os dados pudessem ser processados tão logo estivessem disponíveis;
- que o sistema de estatísticas fosse incorporado ao sistema de jigs;
- que o novo sistema fosse 100% compatível com o sistema antigo; de fato os sistemas antigo e novo deveriam funcionar conjuntamente sem que o sistema novo interferisse no processo existente;
- o sistema novo não necessita realizar ações em tempo real no processo, pois trata-se de um sistema típico de coleta de dados;
- o sistema novo tem necessidade de algumas funcionalidades de *setup* da linha, como por exemplo, a transferência de parâmetros e atualização de código via rede, atualização do calendário (data e hora) do jig etc.; e
- o sistema realiza testes sobre dispositivos de teste e disponibiliza estes dados aos sistema de nível superior, o qual é uma aplicação de geração de estatística do processo de testes nos dispositivos, e não somente um sumarizador de dados.

Para enfrentar estes desafios foi proposto o Proxy de Remotas, que foi estudado e implementado. Os subtópicos seguintes ilustram a implementação.

4.3.2. Topologia da Rede

A topologia da rede implementada para o funcionamento do Proxy de Remotas é a mostrada na Figura 9, isto é, é uma topologia estrela. O Proxy das Remotas pode implementar dois tipos de aplicações: para aplicações de controle pode-se utilizar uma abordagem *mestre/escravo* no acesso à rede; e para aplicações de supervisão pode-se usar a abordagem padrão de mudança de estado, deixando que todos os nós possam se comunicar a qualquer momento. Na primeira abordagem, consegue-se operações em tempo real, na segunda abordagem, somente operações em tempo não real.

Em princípio, o Proxy de Remotas foi implementado para operar sobre uma rede Ethernet. No laboratório foi implementada sobre uma rede RS-232/485 para testes.

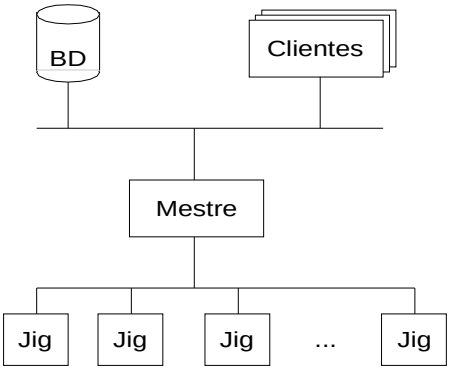


Figura 9: Topologia da Rede

A Figura 9 mostra os elementos da solução: jigs, os quais são dispositivos mínimos, um computador chamado de Mestre, o banco de dados corporativo e as estações clientes do banco de dados.

4.3.3. Elementos do Proxy de Remotas

Descreve-se agora todos os elementos da implementação do Proxy de Remotas mostrados na Figura 9.

4.3.3.1. Jigs

Na implementação, os jigs são os equipamentos que realizam os testes sobre os displays nas linhas de produção, eles são então as remotas para a arquitetura do Proxy de Remotas, isto é, são sistemas mínimos que possuem processamento, memória e capacidade de comunicação. No caso implementado, estes dispositivos têm como características principais:

- Sistemas mínimos microcontrolados. Em particular utilizam um microcontrolador de 8 bits, com entradas e saídas analógicas e digitais;
- Possuem comunicação Ethernet, RS-485 e RS-232 (*CellBus* e *FieldBus*) e barramentos SPI, PPI e I2C (acesso a *chips*/dispositivos);
- Ambiente de programação em linguagem C, com RTOS, stack TCP-UDP/IP e diversas APIs para controle de periféricos, incluindo protocolo *ModBus*;
- Foi implementada uma biblioteca para tratamento de XML, embora não se tenha realizado um *parse* completo, nem implementado um padrão como DOM ou SAX; esta biblioteca foi o suficiente para o tratamento requerido na aplicação;

- Foi implementada uma biblioteca para persistência em memória *flash* dentro do jig. Isto significa que a persistência do Proxy de Remotas, nesta aplicação, passa a ter dois elementos: um no Mestre e outro dentro do Jig.

4.3.3.2. Mestre

É o equipamento responsável pelo controle, supervisão e coleta de dados na rede de jigs. Como é um PC com duas placas de rede, também faz o papel de *gateway* entre a rede de jigs e a rede corporativa. Também nele foram instalados softwares para gerenciamento e configurção da rede. Como fisicamente estava no ambiente de produção, também se instalou um cliente para o banco de dados corporativo.

No Mestre é onde fica a maior parte do código implementado da arquitetura do Proxy de Remotas. Este aplicativo recebeu o nome de Esquilo, em homenagem aos animais que frequentemente apareciam no ambiente dos Laboratórios de Software e Hardware do NUTELI, dentro do Campus Universitário da UFAM, o local onde foi estudada e implementada esta solução. O Esquilo é também uma interface via Web para as funções do Proxy de Remotas e a ferramenta preliminar para visualização da rede de jigs. Deve-se visualizar esta *webapp* como uma ferramenta de depuração, pois ela reflete a rede de jigs como um todo e os relatórios de teste de um jig em particular. O Mestre também realiza funções de configuração da rede e é o nó central de comunicação.

As mensagens passadas pelo Mestre às remotas e virce-versa, são apenas uma outra maneira de se chamar um procedimento remoto. Interessante observar que no jargão dos engenheiros eletrônicos, essas mensagens são conhecidas como chamadas de

tarefas, onde uma tarefa é uma função que o dispositivo remoto executa, podendo ou não retornar um mensagem de resposta, enquanto que o profissional de computação conhece essas mensagens como chamada de procedimento remoto.

Por se tratar de um sistema distribuído, o Proxy de Remotas pode ser encarado como se possuísse dois componentes: um componente cliente, no Mestre, e outro componente servidor nas remotas. O Mestre também fica responsável por armazenamento temporário, configuração da rede de jigs etc.

É interessante notar aqui a inversão do processo padrão da arquitetura cliente/servidor: normalmente as aplicações servidoras ficam na máquina relativamente com maior capacidade de processamento e memória, que no caso fabril é o PC Industrial, uma vez que as remotas são, na maioria das vezes, sistemas mínimos. Na arquitetura do Proxy de Remotas, os servidores, isto é, as entidades que possuem a informação que será colhida são as remotas e, então, o Mestre funciona como um cliente para elas.

Também o Mestre irá funcionar como cliente para o banco de dados corporativo ou o aplicativo de gestão. Neste caso, o Proxy de Remotas mantém a visão tradicional e o desenvolvimento da aplicação é quase idêntica ao desenvolvimento tradicional de um aplicativo de banco de dados.

4.3.3.3. PC SGBD

É o computador servidor onde estará instalado o banco de dados da rede corporativa. A conexão entre o Mestre e ele é feita através de um aplicativo que transforma chamadas RMI em chamadas SQL. Essa camada RMI a mais, foi introduzida para dar modularidade a esta implementação do Proxy de Remotas e permitir que possa haver interação futura do Mestre com outros softwares de gestão que

não somente aquele que foi desenvolvido para a empresa. No SGBDR foi realizado um trabalho de mapeamento objeto-relacional do conjunto de objetos criados para a solução.

Note que em laboratório foi desenvolvida uma camada em serviços Web, de forma que a independência fica completa, isto é, há independência dos aplicativos e mesmo da linguagem em que os aplicativos foram desenvolvidos. Entretanto ainda falta amadurecimento na implementação e correção de defeitos destes serviços para operação em produção.

4.3.3.4. Clientes do Banco de Dados

São os computadores onde rodam os aplicativos clientes do banco de dados e que realizam as atividades de estatística do processo, análise de defeitos, ordem de produção etc. Estes aplicativos de estatística são programas Java que realizam os casos de uso pedidos pela empresa; são instalados nas máquinas clientes e acessam os recursos do banco de dados via TCP/IP. Também realizam chamadas RMI diretamente ao Mestre (ao aplicativo Esquilo).

4.3.4. Aplicativos Desenvolvidos

A Figura 10 mostra o diagrama de distribuição do sistema implementado. Ele mostra os aplicativos desenvolvidos e em quais nós eles são instalados. Abaixo, segue uma descrição dos principais.

- TM (Terminal de Monitoração): é um aplicativo que monitora o banco de dados;

- Coletor: é um serviço que roda ao lado do SGBD e faz uma ponte do mundo Java (RMI) na qual a aplicação foi desenvolvida e o mundo SQL do SGBD; foi implementado como um serviço (um *daemon* no mundo Linux);

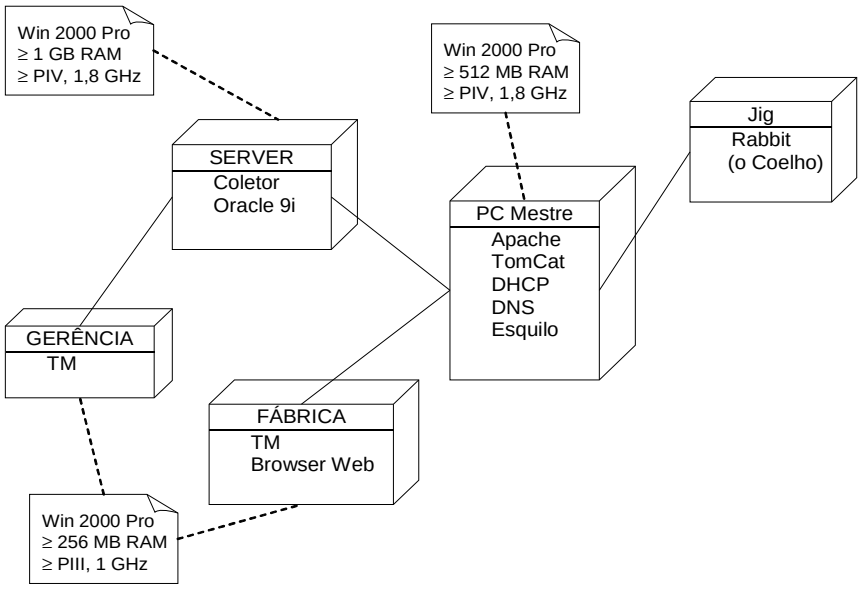


Figura 10: Diagrama de Distribuição

- Esquilo: é uma aplicação da Web que serve de interface visual para o Proxy de Remotas. O Proxy de Remotas em si é um conjunto de serviços e APIs, portanto, sem interface. Dentro do Esquilo foi implementada uma *thread* (um processo leve) que é chamado de Caçador. Sua função é ser cliente da rede fazendo acesso aos jigs através de uma disciplina *round-robin*;
- Rabbit: é o nome da aplicação que roda nos jigs (remotas);

- Apache Tomcat: é o nome do container Web mais usado no mundo; sua escolha se deu pela facilidade de configuração, boa quantidade de atualizações, estabilidade e por ser um software livre, ou seja, temos acesso ao seu código fonte;
- DHCP: foi colocado um servidor DHCP no Mestre para autoconfiguração dos números IPs e outros dados de configuração da rede dos jigs.

Em resumo são estes os elementos que constituem a implementação de exemplo do Proxy de Remotas, mas, como estes componentes conseguem realizar os requisitos pedidos? É o que será visto nos próximos sub-tópicos.

4.3.5. Mapeando Jigs de Teste em Objetos

Entre o Mestre e o jig, trafegam quatro conjuntos de mensagens básicas, a saber: um de Configuração, um para a Tabela de Parâmetros dos Testes; um de Relatório de Testes e um de Atualização de Software. As mensagens entre o jig e o Mestre, que formam estes conjuntos, serão vistas mais abaixo.

Do ponto de vista tanto das aplicações externas quanto de supervisórios a serem construídos dentro do Proxy, estas mensagens podem ser encaradas como acesso a métodos remotos. Alguns destes métodos são chamados pelo jig e outros pelo supervisório.

Essas mensagens foram construídas com sintaxe XML. A escolha se deu pelo seguinte:

- as mensagens são em formato texto, portanto de fácil manipulação, leitura e depuração por um operador humano;

- em termos de integração futura, pode-se facilmente encapsular tal conteúdo em mensagens SOAP, bastando para isso uma simples transformação XSLT;
- facilidade de obtenção de *parses* XML livres disponíveis;
- XML está se tornando o padrão *de facto* para documentos de forma geral e, particularmente, para troca de dados em rede.

Essas mensagens tomam parte de uma cadeia de interrelacionamentos que formam a solução Proxy de Remotas. Todos os interrelacionamentos são mostrados na Figura 11 que traz o diagrama de componentes do sistema implementado. As setas indicam um sentido de comunicação. Lembrar que os grupos de mensagens citadas acima fazem parte somente do interrelacionamento entre o Jig e o Mestre. Os componentes de software que são implementados nestes hardwares são o módulo Rabbit e o módulo Esquilo, como indicado na Figura 10.

Abaixo a descrição de cada um dos conjuntos de mensagens.

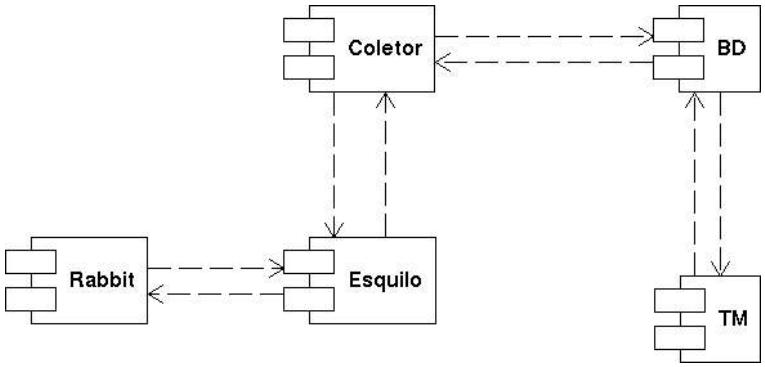
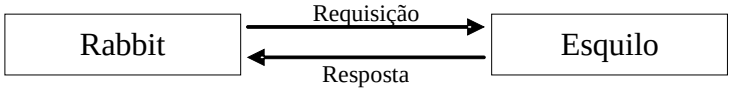


Figura 11 - Diagrama de Componentes do Sistema Implantado

4.3.5.1. Configuração

O jig emite uma mensagem de configuração toda vez que necessita enviar também uma mensagem de relatório de teste. Isto garante que os dados do jig nos programas de gestão e o relógio do jig estejam sempre sincronizados. Informações importantes são transmitidas utilizando este conjunto de mensagens, como por exemplo: a identificação do jig, sua localização, as versões do hardware, do software e da mecânica associados ao jig, etc. para o sistema de gestão ou outros softwares da rede. Notar que o sistema possui uma função inteligente que é a auto detecção do tipo do modelo do dispositivo sob teste (o que é definido pela versão da mecânica) e o download automático do software necessário para a realização dos testes naquele tipo de modelo de dispositivo sob teste. Abaixo o par requisição/resposta de configuração, gerado pelo jig:



Requisição:

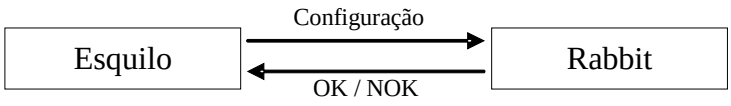
```
<?xml version="1.0"?>
<REQUISICAO codigo="0">
   <CONFIGURACAO versao="1.0" jig="01" autoteste="100"/>
      <VERSAO hardware="2.0" software='"3.0" mecanica="4.0"
              parametros=”1.0” />
      <REDE ip="192.168.10.20"/>
      <MODELO identificacao="DCT4" />
      <LOCALIZACAO linha="01" posto="02" />
   </CONFIGURACAO>
</REQUISICAO>
```

Resposta:

```
<?xml version="1.0"?>
<RESPOSTA codigo="0">
   <CONFIGURACAO versao="1.0" jig="JIG1">
      <LOCALIZACAO linha="01" posto="02" />
      <DATAHORA data=”365” hora=”82000” ciclo=”2560”/>
   </CONFIGURACAO>
```

```
</RESPOSTA>
```

De outras vezes, o sistema de gestão necessita atualizar os dados de configuração do jig, dando ensejo ao seguinte par requisição/resposta:



Configuração:

```
<?xml version="1.0"?>
<REQUISICAO codigo="0">
   <CONFIGURACAO versao="1.0" jig="JIG1">
      <LOCALIZACAO linha="01" posto="02" />
      <DATAHORA data=”365” hora=”82000” ciclo=”2560”/>
   </CONFIGURACAO>
</REQUISICAO>
```

Resposta:

```
<?xml version="1.0"?>
<RESPOSTA codigo="OK"/>
```

ou

```
<?xml version="1.0"?>
<RESPOSTA codigo="NOK"/>
```

4.3.5.2. Tabela de Parâmetros dos Testes

A tabela de parâmetros dos testes é também parte da configuração do jig. Entretanto, dado o seu tamanho e os casos especiais que engendra, criou-se um conjunto de mensagens próprias para este tratamento, dentre elas:

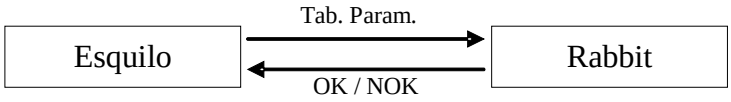


Tabela de Parâmetros:

```
<?xml version=”1.0”?>
<REQUISICAO codigo=”1”>
   <PARAMS versao=”1.0” jig=”01” modelo=”DCT4”>
      <PARAM id=”20” li=”10.3” ls=”10.5”/>
      <PARAM id=”21” li=”2.8”  ls=”3.0”/>
      <PARAM id=”22” li=”2.8”  ls=”3.2”/>
      .
      .
      .
   </PARAMS>
</REQUISICAO>
```

Resposta:

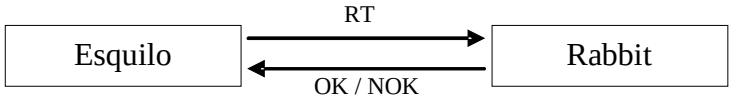
```
<?xml version="1.0"?>
<RESPOSTA codigo="OK"/>
```

ou

```
<?xml version="1.0"?>
<RESPOSTA codigo="NOK"/>
```

4.3.5.3. Relatórios de Testes

Os relatórios de testes são, em verdade, as mensagens do estado do jig. Há quatro tipos de relatórios de testes: relatórios de anúncio, relatórios de produção, relatórios de parada automática e relatórios de dispositivo de teste inválido, mas todos eles possuem a mesma estrutura, modificando-se tão somente alguns atributo e o conjunto de elementos <TEST>.



Relatório de Testes:

```
<?xml version=”1.0”?>
<REQUISICAO codigo=”2”>
   <RT versao=”1.0” jig=”01” identificacao=”DCT4” serie=”123”
          quantfila=”1” autoteste=” 100” data=”365” hora=”123”
          ticksteste=”234” tickscicio=”1222” reteste=”S” >
      <TEST id=”20” res=”00.01” val=”1.1”/>
      <TEST id=”21” res=”00.02” val=”2.3”/>
      <TEST id=”22” res=”00.03” val=”4.5”/>
      .
      .
      .
   </RT>
</REQUISICAO>
```

Resposta:

```
<?xml version="1.0"?>
<RESPOSTA codigo="OK"/>
```

ou

```
<?xml version="1.0"?>
<RESPOSTA codigo="NOK"/>
```

4.3.5.4. Atualização de Software

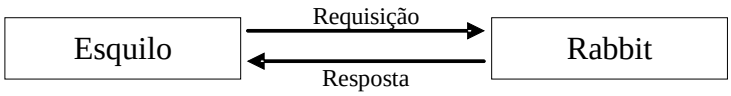
O sistema implementado possui uma função inteligente, que é a detecção automática do tipo de dispositivo sob teste e o download também automático do software adequado para a realização dos testes sobre aquele dispositivo descoberto.

Para isso, foi desenvolvido um conjunto de mensagens para o reinício do sistema e, a partir de um aplicativo de carga, chamado de Download Manager realiza as seguintes atividades:

- recebe da rede o software correto via ftp, atualizando o sistema com o novo software;
- realiza um novo reinício e, em caso do software estar válido, executa-o; caso o software não estar válido, reinicia o jig para manter-se a espera de um novo software via ftp.

Esta seqüência de operações é devida ao fato do sistema da remota não possuir memória de massa independente, isto é, a sua memória principal é igualmente a sua memória de massa. É impossível realizar a substituição de um software em uma memória *flash* a partir de um programa residente na própria memória e, dado o tamanho do programa sendo transmitido, fica inviável igualmente copiá-lo para uma memória de trabalho (RAM) e depois para a memória permanente.

A mensagem de reinício é a seguinte:



Requisição

```
<?xml version="1.0"?>
<REQUISICAO codigo="5" usuario=”admin” senha=”admin”/>
```

Resposta:

```
<?xml version="1.0"?>
<RESPOSTA codigo="OK"/>
```

ou

```
<?xml version="1.0"?>
<RESPOSTA codigo="NOK"/>
```

Com base nas mensagens acima, fica facilitada a construção de um conjunto de objetos que representem os dados das remotas. Nos próximos sub-tópicos verifica-se como o Proxy de Remotas realiza esta tarefa.

4.3.6. Comunicação entre Remotas e o Mestre

Do ponto de vista do Mestre, é possível construir dois *jiglets*: um que tem a função de ser um objeto distribuído acessado pelo Rabbit (o componente dentro da remota) e o outro um cliente que acessa os serviços do Rabbit. Este, por sua vez, também pode ser visualizado como possuindo dois *jiglets,* uma para ser acessado pela aplicação externa, isto é, a *thread* Caçador, e o outro sendo um cliente para realizar acesso à aplicação externa, isto é, um módulo de tratamento das remotas do Esquilo.

Os objetos que representam os dados das remotas foram modelados conforme mostrado na Figura 12.

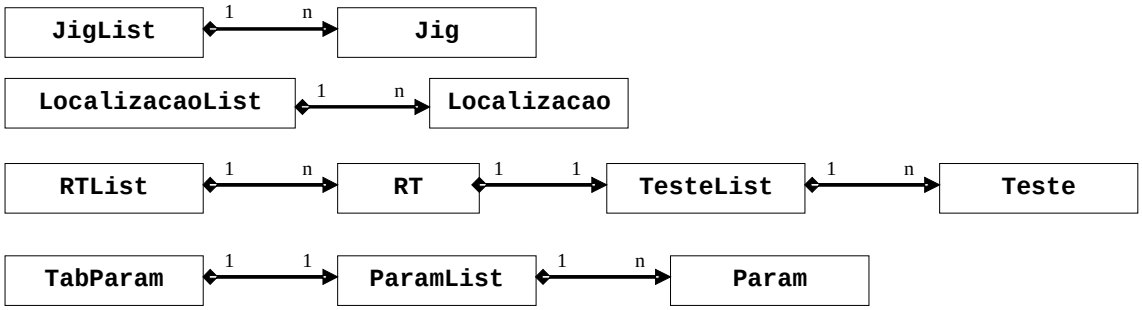


Figura 12: Modelo de Dados das Remotas - Visão do Mestre

O *jiglet* de entrada foi chamado de *InputJiglet* para ser o objeto distribuído acessado pelas remotas. O *port* de entrada foi chamado de *Listener*. Ele implementa a troca de mensagens em XML sobre soquetes TCP/IP, sem um protocolo de transporte e emite para o *InputJiglet* mensagens passando como parâmetro os objetos Jig, Localizacao, RT ou TabParam, conforme o caso. Prevê-se a utilização de HTTP como protocolo de transporte e sobre ele SOAP, transformando os jigs em servidores SOAPs, permitindo o futuro uso serviços Web.

O *jiglet* de saída foi chamado de *OutputJiglet* para ser o objeto cliente a acessar as remotas. O *port* de saída foi chamado de *Talker*.

4.3.7. Comunicação com os níveis superiores

Para a comunicação entre os nós Mestre (componente Esquilo) e Banco de Dados (componente BD) há um programa de interface que transforma chamadas RMI em chamadas SQL. Ele é chamado de Coletor, como mostrado na Figura 11. Este software fica residente no nó Banco de Dados.

Do ponto de vista do Proxy de Remotas, na sua implementação com RMI, a comunicação entre os sistemas opera-se dentro de uma hierarquia de classes. É mostrada, na Figura 13, uma das implementações utilizadas em laboratório para o sistema em questão:

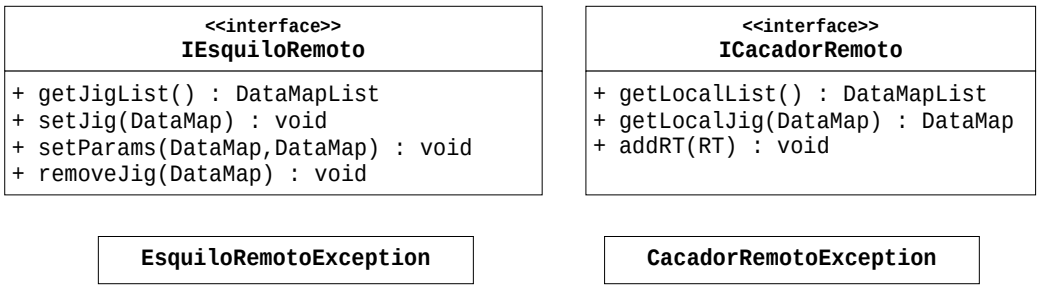


Figura 13: Interfaces para RMI

4.4. A Integração com o Proxy de Remotas

O Proxy de Remotas favorece a integração fabril pois com ele é possível o desenvolvimento de aplicações abertas, favorecendo a interoperabilidade entre as diversas aplicações. Também com ele é possível o tratamento do legado. Há igualmente a disseminação da cultura do desenvolvimento dentro de padrões. De fato, o uso de softwares livres e padrões de projeto para o desenvolvimento de aplicações faz com que a grande comunidade de desenvolvedores crie maneiras para interoperar os seus softwares e, mais ainda, criam-se arquiteturas e *frameworks* que não necessitam de

licenciamento para serem utilizados, isto é, são abertos. Isto impõe aos novos desenvolvimentos uma visão centrada na integração. Como resultado, se obtém uma redução substancial nos custos do desenvolvimento dos softwares devido a reutilização de software e o uso de padrões bem estabelecidos para projetos e implementação.

O Proxy de Remotas vem, ao seu tempo, ser mais uma peça nesta engrenagem, pois a sua concepção adere às características elencadas em Caracterização do Ambiente Fabril e abaixo reelencadas indicando-se como o Proxy de Remotas as atende:

- integração com dispositivos de campo inteligentes, isto é, com dispositivos mínimos que possuem capacidade de processamento, memória e comunicação em rede. De fato o Proxy de Remotas permite o desenvolvimento de sistemas distribuidos, sendo que alguns objetos ficam residentes nos dispositivos mínimos e são os responsáveis pela comunicação, a qual deverá ser sempre feita através de um protocolo padrão;
- permitir que diversos protocolos coexistam, quando viável: o conceito de *port* possibilita essa coexistência;
- funções inteligentes intrínsecas à arquitetura: como mostrado, diversas funções podem ser implementadas através do conceito de *jiglets*. Na prática implementaram-se funções de replicação de dados para um banco de dados relacional e foi realizado o tratamento imediato do estado das remotas;
- permitir o desenvolvimento de supervisório próprio: isto pode ser obtido na arquitetura por meio dos *jiglets* e de chamadas aos serviços Web;
- permitir integração com supervisórios externos: de fato o padrão do Proxy de Remotas é emitir métodos de seus objetos como serviços Web, o que permite integração com supervisórios externos e os outros softwares de gestão. Na

implementação realizada utilizou-se RMI, por motivos de performance e facilidade de implementação, pois o supervisório externo foi construído em Java;

- diversas capacidades de persistência: usando-se o padrão DAO com Fábrica é possível o desenvolvimento de soluções com múltiplos destinos para persistência; em especial cita-se o desenvolvimento de um DAO dentro das remotas para permitir persistência em memórias flash e também o uso de persistência em XML e persistência em um SGBDR em uma máquina Mestre da rede;
- possibilitar serviços básicos: implementou-se com o Proxy de Remotas gerenciamento automático dos jigs de testes, controle do estado operacional dos jigs, coleta de dados etc.;
- extensibilidade de serviços: com o conceito de *jiglets* agindo como filtros, o Proxy de Remotas permite que novos serviços sejam adicionados ao sistema e à própria arquitetura;
- redes heterogêneas: de fato, com o conceito de *ports* fica viável o desenvolvimento de aplicações para diversos tipos de redes;
- tratamento de sistemas legados: também com o conceito de *jiglet* é possível criar objetos que espelhem as informações e funcionalidades de remotas, implementando-se assim um meio de tratar o legado;
- a arquitetura deve ser aberta: de fato o Proxy de Remotas foi imaginado utilizando-se padrões abertos, tal como XML para a troca de informações;

- fácil de programar: foi utilizada a linguagem Java pelos motivos já expostos anteriormente, entretanto este item merece uma atenção especial, pois que todas as linguagens e arquiteturas tem seus pontos fortes e fracos que gerarão problemas quando se tornarem eles mesmos legados como mostrado por [7] .
- fazer uso de diferentes linguagens: o uso de serviços Web no Proxy de Remotas permite a comunicação entre aplicações escritas em diferentes linguagens.

É claro que o Proxy de Remotas não resolve todos os problemas por ser uma arquitetura e, portanto, um desenho geral de como um sistema deva ser pensado com a finalidade de integração e não um sistema comercial acabado.

Há inúmeros obstáculos do ponto de vista de implementação a vencer: ainda não foi feito um trabalho de comparação entre as diversas tecnologia disponíveis para a escolha das melhores em termos de alguns parâmetros necessários para o ambiente industrial, como performance, robustez e flexibilidade; há diversos sub-domínios dentro do grande domínio que são as aplicações do chão-de-fábrica, e este trabalho preocupou-se com a implementação de questões como o estado das remotas e as estatísticas dos testes efetuados pelos jigs no ambiente de produção. Outros componentes necessitam ainda serem implementados para permitir uma abordagem mais ampla dentro do conceito de CIM[27].

27 CIM: *Computer Integration Manufacturing*, Manufatura Integrada por Computador. Este conceito estende a automação industrial para um processo de integração completa da produção. Veja-se em [10] para maiores detalhes.

Também não foram inseridos elementos que pudessem mapear não somente as remotas da fábrica mas todo o conjunto de equipamentos da fábrica, mesmo os que não possuam processamento e memória, o que permite a análise e o controle do fluxo de informações na empresa. Isto passa pelo estudo do modelo da empresa e possível incorporação do Proxy de Remotas dentro de uma arquitetura de modelagem mais ampla.

Capítulo 5

CONCLUSÕES

A realização da integração em uma fábrica pode ser obtida através da solução de dois grandes problemas: a comunicação entre os elementos a serem integrados e a interoperabilidade entre as aplicações. Com isso é possível integrar sistemas novos e sistemas legados.

O primeiro problema pode ser resolvido adotando-se padrões de comunicação. Essa padronização é feita sobre coisas tais como quais as redes a serem utilizadas e como serão as interconexões entre elas. Em especial, as interconexões são feitas através de equipamentos e softwares que agem como *gateways*. É o caminho que a Profibus International adota, em que vários *gateways* foram desenvolvidos com a finalidade de permitir que redes heterogêneas, como as redes Ethernet e RS-485 por exemplo, coexistam em um mesmo ambiente.

O segundo problema pode ser resolvido adotando-se também uma padronização da forma como os diversos componentes de software podem comunicar-se, independente da linguagem em que foram escritos ou da plataforma em que executam. As soluções mais proeminentes são o *middleware* CORBA, método padronizado pela OMG, e o uso de serviços Web, desenvolvido e padronizado pela W3C, a partir de uma iniciativa da Microsoft. Ambas são fundamentalmente soluções de chamada de procedimentos remotos acopladas a inúmeras APIs e objetos de suporte.

Mas a integração, vista como um problema em si, deve ser encarada pela empresa como uma das questões estruturais, isto é, deve ser estudada dentro do contexto do modelo da empresa. Entende-se por modelo de empresa os seus processos, estrutura, níveis de operação e integração desejados, níveis de automação etc., isto é, como a empresa pode ser concebida, estudada e implementada.

Para se desenvolver uma proposta de integração fabril deve-se ter em mente esse modelo. Este trabalho apresentou um modelo hierárquico e sobre ele desenvolveu sua tese de como uma proposta de integração deve ser. Resumidamente podemos dizer que uma proposta de integração fabril entre os níveis operacional e supervisão deve observar algumas características:

- desenvolver aplicações de tempo real ou não;
- possuir uma interface de aplicação clara e precisa, usando protocolos abertos;
- possuir um *framework* comum de aplicação;
- possibilitar o desenvolvimento de aplicações descentralizadas.

Com a visão nestas características propôs-se uma arquitetura para integração que ao mesmo tempo resolvesse o problema de comunicação e de interoperabilidade entre aplicações.

Para realizar a comunicação entre redes heterogêneas, propomos o uso de um *gateway*, chamado de Proxy de Remotas, o qual permite o desenvolvimento de aplicações de troca de dados entre as diversas redes sendo instalado em uma máquina que disponibiliza acesso hardware para cada uma das rede. O elemento da arquitetura desta solução que permite este desenvolvimento é o conceito de *port*, isto é, pequenos componentes de software que agem como porta de comunicação para cada rede.

Para realizar a interoperabilidade entre aplicações propomos também o uso de um *gateway* entre aplicações. Isto é feito a patir do desenvolvimento de aplicações que realizam o tunelamento (ou conectorização) entre as diversas aplicações da fábrica, o que pode ser feito dentro do mesmo Proxy de Remotas que então passa a ser um elemento único para a integração, tanto para comunicação quanto para aplicações. O elemento da arquitetura da solução que permite este desenvolvimento é o conceito de *jiglets*, isto é, pequenos componentes de software que agem como tratadores dos dados entre aplicações. Um *jiglet* é um componente de software que usa os serviços de um *port* para comunicação. As funcionalidades e dados das remotas podem ser implementadas dentro do *jiglet*. Ele também é o responsável pela lógica da aplicação a ser desenvolvida.

Além desses elementos, o Proxy da Remota também possui dois outros elementos: uma camada de persistência e uma camada de serviços. A camada de persistência utiliza o padrão de projetos *Data Access Object with Factory* (Objeto de Acesso a Dados com Fábrica) o qual isola os problemas relacionados com o tipo de tecnologia utilizada para persistência. A camada de serviços tem por objetivo externar, para as demais aplicações da fábrica, notadamente as aplicações de gestão, métodos de objetos que representam os equipamentos da fábrica.

Isto quer dizer que o Proxy de Remotas realiza, no modelo arquitetural utilizado, o papel de um sistema de supervisão, isto é, está colocado entre as camadas de Controle e de Gestão. De um lado participa da integração das diversas células de automação, cada uma delas implementada com uma remota; de outro participa da integração vertical, disponibilizando serviços para as aplicações de gestão. Também participa da integração horizontal, pois pode permitir que outras aplicações de supervisão, não desenvolvidas dentro da arquitetura, possam interoperar com ela.

Podemos resumir dizendo que as características mais marcantes do Proxy de Remotas são:

- todas as características acima citadas para um proposta de integração fabril;
- possui persistência; e
- é orientado a componentes.

Transformar remotas reais de uma linha de produção, tais como CLPs, jigs, atuadores etc., em objetos de software permite uma grande flexibilidade para o programador dos softwares de supervisão e controle de uma planta fabril. Realizar isto de maneira aberta e expansível torna o sistema quando implementado um grande produto de integração das redes de automação. Além disso, esta capacidade abre uma interface para que os softwares de gestão possam acessar os serviços da fábrica, até então caros de ser obtidos. Isto tudo redunda na realização do aperto de mão entre a engenharia de automação e o processamento de dados da empresa.

A proposta aqui descrita é inspirada em *gateways* e *proxies* de redes, mas aplica o conceito em um nível de abstração maior, utilizando-se de elementos modernos, tais como componentes de software e protocolos abertos, como serviços Web.

Cada remota ou aplicação que se queira interconectar passa a ter um objeto que se lhe reflete as características sendo possível gerar objetos distribuídos, de forma flexível, extensível e aberta. Flexível, por não assumir nenhum pressuposto sobre os objetos em si e como eles devem ser implementados, porque isso depende do domínio da aplicação; extensível, por se utilizar técnicas que possibilitam a agregação gradativa de novas funcionalidades, através de componentes; e aberta, por utilizar protocolos

padronizados. De fato o Proxy de Remotas foi inicialmente pensado para realizar troca de informações e armazenamento em XML em todos os seus níveis, mas admite outras soluções.

Com este trabalho acreditamos que há um melhora nos problemas relacionados à integração e apresentados de forma detalhada anteriormente. Por exemplo, a fragmentação de tecnologias para automação industrial, apesar de inerente ao processo, passa a ser utilizada em favor da integração, pois é necessário somente desenvolver uma aplicação *gateway* entre as diversas tecnologia encontradas na fábrica. Claro que isso é possível se as tecnologias permitirem tal interoperabilidade, pois há casos em que as tecnologias subjacentes utilizadas são conflitantes e, então, não são passíveis de serem integradas.

Entretanto, um esforço grande ainda deve ser feito pela empresa para compreender melhor como realizar a sua integração, ou melhor, como desenvolver o seu modelo de empresa. O Proxy de Remotas não contempla a modelagem da empresa, isso requer um estudo mais aprofundado a parte. Mas ele permite que desenvolvedores possam rapidamente levar dados do chão-de-fábrica para os sistemas de gestão. Isto quer dizer que a empresa terá chances de contextualizar o máximo de dados e, se essas contextualizações forem bem modeladas, formam o conjunto de conhecimentos da empresa, que é o maior bem na atualidade.

Ao propor agentes intermediários para o processo de automação e a geração de células de automação, os equipamentos e o software tornam-se extremamente modulares. Quando transformamos as remotas em objetos de software em uma rede distribuída também estamos aderindo às novas técnicas de modularização e reutilização de recursos da engenharia de software. A modularidade no software pode ser conseguida com o uso de componentes.

Com a idéia do Proxy de Remotas implementada em uma unidade fabril, os sistemas passam a ter ao menos um ponto de interligação: o Proxy das Remotas, o qual passa a agir como um *middleware* em que os diversos desenvolvedores das diversas áreas da empresa podem desenvolver a sua linguagem e seus conceitos conjuntamente. Com isso a visão integrada da empresa é ampliada, pois a integração entre os softwares aumenta, apesar da diversidade de soluções possíveis.

Do trabalho realizado junto às empresas do Pólo Industrial de Manaus, e aqui descritos, como maior resultado, temos um retorno empírico das empresas e instituições envolvidas, que podem ser pontuadas:

- a disseminação das técnicas de engenharia de software no meio fabril, permitindo um maior preparo dos profissionais da área para a nova era de colaboração, vai Web que se forma entre o chão de fábrica e os níveis superiores da empresa;
- a melhoria da qualidade dos produtos sendo testados, dado que os dispositivos sob teste são submetidos a algoritmos e processos de teste mais rigorosos e controlados;
- a melhoria do processo produtivo, sendo isto comprovado pela diminuição do números de rejeitos e defeitos encontrados pelos clientes das fábricas em que um processo de integração é objetivado;
- aumento da capacidade de uso do conhecimento da empresa, uma vez que o sistema integrado disponibiliza as informações necessárias para as pessoas adequadas e o mais rápido possível, permitindo uma tomada de decisão e ações em tempo hábil.

Sabe-se que ainda há diversos pontos a serem estudados acerca da integração, sobre seus problemas, projeto e execução, e sobre a padronização, o que envolve questões políticas entre grandes empresas no mundo. Também há questões humanas, como a formação e a capacitação de profissionais aptos à realizarem a integração.

Demais, diversas empresas no mundo geraram suas propostas para os problemas aqui apresentados, entretanto, todas elas apresentam problemas, sendo o principal deles o interesse que a empresa tem de manter-se no mercado, o que as estimulam a desenvolver tecnologia voltada apenas para si e menos para a interoperabilidade.

Entretanto, algumas propostas **devem** ser colocadas mesmo em um ambiente não favorável a fim de fazer seus atores pensarem em alternativas. Somente com alternativas em mãos as empresas e os desenvolvedores poderão se beneficiar da diversificação da tecnologia que se vê hoje e, por fim, alcançarem os objetivos e os benefícios da integração.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Apache Licence, Version 2.0. **Apache License, Version 2.0 - Apache Software Foundation**. http://www.apache.org/licenses/LICENSE-2.0, acesso em julho/2005.

[2] ARAGON, Doris F.; ARAÚJO, Evandro de O. *et al*. **Sistemas Inteligentes: Fundamentos e Aplicações**. / organização, Solange Oliveira Rezende. Barueri, SP: Manole, 2003.

[3] BECKER, L. B.; GERGELEIT, M.; NETT, E.; *et al*. "An Integrated Environment for the Complete Development Cycle of an Object-Oriented Distributed Real-Time System". **Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing**, 1999, p.165.

[4] GARCÍA, R. G.; GELLE, Esther; STRONHMEIER, Strohmeier. A Software Architecture for Industrial Automation. **Technical Report IC**, 2003 nº 28.

[5] GNU General Public Licence, Version 2.0 (GPL). **GNU General Public Licence, Version 2.0, 1991**. http://www.gnu.org/copyleft.

[6] Java Consortium. **JIM Reference Implementation**, 2001. http://www.javacons.gr.jp, acesso em março/2005.

[7] LEWIS, Tec. If Java is the Answer, What is the Question? **Computer**. IEEE Computer Society, 1997, p.136 . ISSN 0018-9162.

[8] MATOS, Luís M. C. Integração de Sistemas de Manufatura – Das ilhas de automação às empresas virtuais. **Revista INGENIUM**, 2ª Série, nº 56, março de 2001, ISSN: 0870-5968.

[9] NAKAMOTO, Yukikazu; HACHIYA, Shoichi. **JTRON: A Hybrid Architecture Integrating an Object-oriented and Real-time System**. IEEE: 2001. ISBN: 0-7695-1089-2/01.

[10] NUMA. **Integração de Empresas/CIM**, compilado por Prof. Henrique Rozenfeld. Última atualização: novembro de 1999. Veja-se o sítio do NUMA: http://www.numa.org.br/conhecimentos/conhecimentos_port/index.html para os links sobre o assunto.

[11] da ROCHA, Maria Margarete. **Integração Vertical e Incerteza**. D. Sc. Universidade de São Paulo. São Paulo: 2002.

[12] Sun Microsystems, Inc. **Sun Community Source License (SCSL)**.

[13] Sun Microsystems, Inc. **The Jini™ Architecture Specification, version 2.0**. http://www.sun.com/software/jini/whitepapers/architecture.pdf, acesso em agosto/2004.

[14] SZYPERSKI, Clemens. **Component Software: Beyond Object-Oriented Programming** /com Dominik Gruntz e Stephan Murrer. 2ª ed. New York, Addison-Wesley, 2002. ISBN 0-201-74572-0

[15] TOMMILA, T.; VENTÄ, O.; KOSKINEN, K. Next Generation Industrial Automation – Needs and Opportunities. **Automation Technology Review**, 2001 [ATR-2001], p.34 a 41.