EMERSON PEREIRA RAPOSO

UM MODELO ABERTO PARA ROBÔS MANIPULADORES DE SERVIÇO

FLORIANÓPOLIS – 2008

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UM MODELO ABERTO PARA ROBÔS MANIPULADORES DE SERVIÇO

Tese submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Doutor em Engenharia Elétrica.

EMERSON PEREIRA RAPOSO

Florianópolis, Março de 2008.

UM MODELO ABERTO PARA ROBÔS MANIPULADORES DE SERVIÇO

Emerson Pereira Raposo

'Esta Tese foi julgada adequada para obtenção do Título de Doutor em Engenharia Elétrica, Área de Concentração em *Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

_____
Prof. Marcelo Ricardo Stemmer, Dr. Ing.
Orientador

_____
Profa. Kátia Campos de Almeida, Ph. D.
Coordenadora do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____
Prof. Marcelo Ricardo Stemmer, Dr. Ing.
Presidente

_____
Prof. Arthur José Vieira Porto, Dr.

_____
Prof. Hans Jörg Andreas Schneebeli, Dr.

_____
Prof. Edson Roberto De Pieri, Dr.

_____
Prof. Werner Kraus Jr., Ph. D.

_____
Prof. Armando Albertazzi Gonçalves Jr., Dr. Eng.

À minha amada esposa, Danielle, e ao meu querido filho, Victor, com muito amor.

Agradecimentos

Ao professor Marcelo Ricardo Stemmer, um especial agradecimento pela orientação e amizade na realização deste trabalho;

Aos funcionários e professores do departamento de automação e sistemas (**DAS**); do departamento de engenharia elétrica e do departamento de mecânica da **UFSC**; Ao Instituto de Tecnologia para o Desenvolvimento (**LACTEC**) pelo oportuno desafio e a COPEL, FINEP, FURNAS, RHAE, PADCT pelo apoio financeiro;

Aos professores Arthur Vieira Porto, Armando Albertazzi Jr., Edson De Pieri, Hans-Jorg Schneebeli e Werner Kraus, que constituíram a banca examinadora da defesa de tese ao doutorado pelas críticas e sugestões;

Aos membros do projeto Roboturb Professor Armando Albertazzi Jr., Professor Marcelo Ricardo Stemmer, Professor Edson De Pieri, Professor Raul Guenther, Professor Daniel Martins, Professor Werner Kraus, Professor Jair Dutra, Professor Carlos Alberto Mártir, Engenheiro Walter Antônio Kapp, Engenheiro Henrique Simas, Engenheiro Milton Pereira, Engenheiro Tiago Loureiro Pinto, Engenheiro Nelso Bonarcorso e Engenheiro Rafael Leal, por saber o real sentido das palavras Ciência, Engenharia e Brasil.

Aos Amigos ...

A Lintaro Nishida, Terezinha Nishida, Alexandra e Waleska Nishida por seu apoio;

À minha esposa, Danielle Nishida Raposo, e ao meu filho Victor Nishida Raposo, por me darem o prazer de compartilhar sua existência;

Aos meus Pais, José Esteves Raposo e Terezinha Pereira Raposo, e à minha irmã Michelli Pereira Raposo, e meu irmão Heber Pereira Raposo pelo amor; Aos meus avos, João dos Santos Pereira e Maria Guiomar Pinheiro Pereira por seu amor e sabedoria;

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

UM MODELO ABERTO PARA ROBÔS MANIPULADORES DE SERVIÇO

Emerson Pereira Raposo

Março/2008

Orientador: Marcelo Ricardo Stemmer, Dr. Ing.
Área de Concentração: Automação e Sistemas
Palavras-chave: *Open Systems, Object-Oriented,* **IEEE** Standards, *Formal Specification, Control System, Controllers, Industrial Robot, Service Robot.*
Número de Páginas: 135

RESUMO:

Desde a década de 80 existem várias tentativas de criar a base funcional de *hardware* e *software* para sistemas de controle de robôs, baseados em um padrão aberto. Entretanto, somente a partir de 1990 é que realmente foram obtidos abrangência de resultados complementares. Dentre os principais trabalhos realizados de maneira unificada a partir de diversas universidades, centros de pesquisas e empresas advindas de diferentes continentes, têm-se o projeto **OSACA** que é uma iniciativa vinda da Europa. A outra proposta é o **OMAC** que tem origem nos Estados Unidos, e, por fim, a última abordagem é denominada **OSEC**, nascida no Japão. Mas, nenhuma delas chegou a um robô aberto. Em termos gerais, este trabalho pretende definir um modelo aberto para robôs que atenda a especificações rígidas de sistema aberto, respeite padrões internacionais de *software* e *hardware*, flexibilize as aplicações na área de robótica e permita, dessa forma, disseminar a utilização de robôs em áreas até então pouco exploradas. Para tal, foi definida uma especificação padrão de sistema aberto de modo a fornecer suporte para a implementação de sistemas de controle de robôs usando a filosofia de sistemas abertos. Assim, ao final deste trabalho é apresentado um modelo aberto para robôs, implementado e aplicado para demonstração do seu ineditismo.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

AN OPEN MODEL FOR SERVICE MANIPULATOR ROBOTS

Emerson Pereira Raposo

March/2008

Advisor: Marcelo Ricardo Stemmer, Dr. Ing.
Area of Concentration: Automation and Systems
Keywords: *Open Systems, Object-Oriented*, **IEEE** Standards, *Formal Specification, Control System, Controllers, Industrial Robot, Service Robot.*
Number of Pages: 135

ABSTRACT:

Since the 80es there have been several attempts to create the hardware and software functional basis for robot control systems based upon an open standard. Nevertheless, really useful complementary results were only obtained after 1990. Among the mainstream works accomplished in an unified form by several universities, research centers and companies from different continents, we have the **OSACA** project, that is an initiative coming from Europe. Another proposal is **OMAC,** which has its origin in the United States, and, finally, the latter approach is known as **OSEC**, born in Japan. But none of them reached an open robot. In general terms, this work seeks to define an open robot model that formally meets the open systems specifications, take internationally accepted software and hardware standards into account, enable flexible applications in the robotics field and permits the spreading of robot's use in lesser explored areas. For that purpose, a formal specification of an open system to provide support for the implementation of robot control systems using the open systems philosophy was defined. Starting from that definition, we present an open robot model and validate it by demonstrating its applicability to several real world case studies.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

Capítulo 1

Introdução

O robô talvez seja o elemento mais “emblemático” da automação. Inicialmente, sua utilidade era substituir o trabalho humano em tarefas associadas à manufatura com o objetivo de aumentar a produtividade de maneira flexível, com qualidade e segurança. Atualmente, os robôs são usados em diferentes tarefas, mas a área de manufatura permanece como o maior mercado onde os robôs são aplicados, incluindo soldagem, pintura, carregamento e descarregamento. A indústria automotiva ainda é o setor que mais tem usado as vantagens da robótica (*United Nations Economic Commission for Europe)* (**UNECE**, 2004).

Em 1956, os americanos George Devel, inventor de uma máquina programável para realizar manipulação, e Joseph Engelberg, empreenderam a primeira fábrica, denominada **Unimation**, para produzir robôs. Posteriormente, na década de 60, o primeiro robô começou a trabalhar na indústria automobilística. Nas décadas de 70 e 80 nasceram as associações de robôs: em 1972 a japonesa (*Japanese Industrial Robot Association* - **JIRA**) e em 1974 a americana (*Robotics Industries Association* - **RIA**). A partir dos simpósios que reuniram as comunidades internacionais para o intercâmbio de informações sobre robótica nasceu em 1987 a *International Federation of Robotics* (**IFR**) na Suíça. Existem diferentes interpretações do que seria considerado um robô. Entretanto, segundo a definição adotada pela **IFR**, norma **ISO** (*International Standards Organization*), o robô é uma máquina para manipulação automática re-programável e multipropósito com, pelo menos, três eixos, com base fixa ou móvel, e podendo ser classificada segundo sua aplicação como industrial ou de serviço (**ISO**, 2004).

O robô industrial é aquele que tem como objetivo a produtividade. E o robô de serviço é feito com o intuito de resolver um problema específico. Assim, a partir das necessidades crescentes da utilização de robôs, tanto na indústria como no dia a dia das pessoas, tornaram-se necessárias características que lhes possibilitem ser adaptáveis a diferentes situações cotidianas. As limitações existentes estão sendo minimizadas com os avanços da eletrônica, informática e mecânica, entre outras, o que proporcionará o desenvolvimento de novas pesquisas e aplicações.

O **IFR** reconhece somente duas classificações de robôs: industrial e de serviços. Porém, com a crescente demanda e com a evolução tecnológica, têm surgido outras classes diferentes de aplicações nas quais os robôs são incorporados, tais como: Autônomos, Domésticos, Artrópodes, Sociais e Educacionais (KATZ, et al., 2000).

Segundo a **ONU** e a **RIA**, na década passada o preço médio de robôs, de modo geral, caiu. Estima-se que este mercado tenha um crescimento médio mundial de 7.4% ao ano. A expectativa para 2006, figura 1, era de pelo menos 1.000.000 robôs sendo usados na indústria mundial, havendo claros indícios de que o mercado esteja em expansão (**UNECE**, 2004).

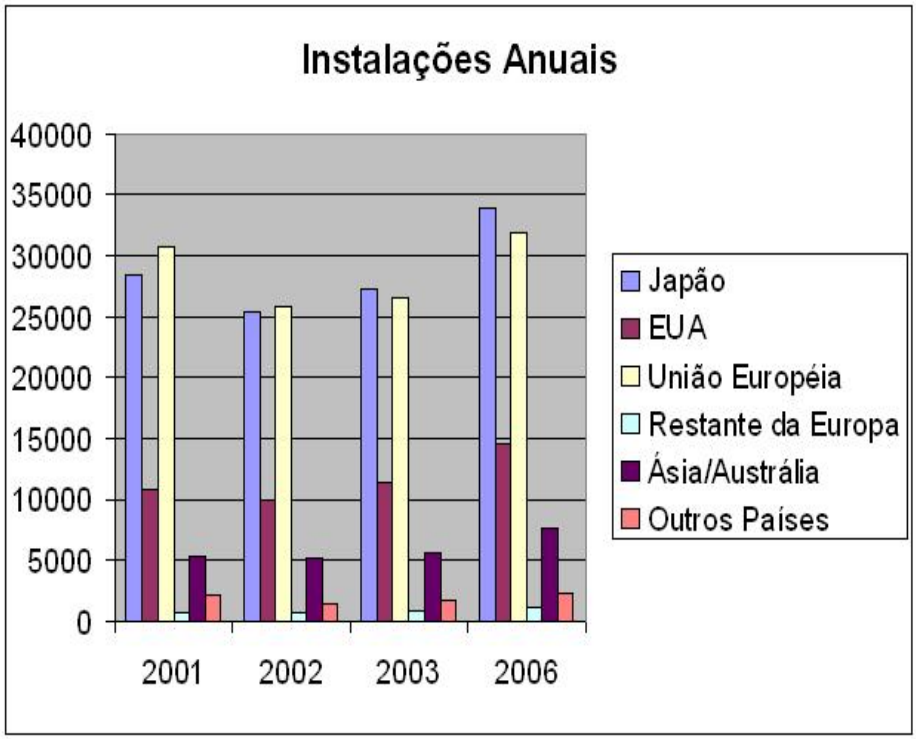


Figura 1: Mercado de Robôs.

1.1 – O Projeto Roboturb

No Brasil também é crescente o número de robôs. Atualmente são sete mil robôs instalados no país, em sua maioria utilizada no setor automobilístico (LEAL, et al., 2005). Estes robôs são quase exclusivamente industriais. Contrariando esta tendência e contribuindo para a independência científica e tecnológica do país, desde 1998 no Brasil se vem trabalhando para conceber e aplicar um robô de serviço com o objetivo de realizar a automatização da tarefa de recuperação das pás de rotores de turbinas hidráulica grande porte erodidas por cavitação. Assim, foi estabelecida uma parceria entre a **UFSC** (Universidade Federal de Santa Catarina) e o **LACTEC** (Instituto de Tecnologia para o Desenvolvimento) apoiada inicialmente pela COPEL, denominado Roboturb, e mais tarde também por FURNAS (RoboFurnas) através dos órgãos de fomento à **P&D** (Pesquisa e Desenvolvimento) ANEEL, PADCT, RHAE e FINEP. O projeto contou com o orçamento financeiro de aproximadamente US$ 4.000.000,00 para um período de oito anos de trabalho e reuniu em torno de 50 pessoas, dentre técnicos, engenheiros, professores, estudantes de graduação e pós-graduação. A equipe era distribuída nos seguintes laboratórios:

- **LACTEC** – Projeto mecânico e gerência;
- Departamento de Engenharia Mecânica da **UFSC**: Laboratório de Metrologia (**LABMETRO**) – Projeto do sistema de medição;
- Departamento de Engenharia Mecânica da **UFSC**: Laboratório de Soldagem (**LABSOLDA**) – Projeto do sistema de soldagem;
- Departamento de Engenharia Mecânica da **UFSC**: Laboratório de *Hardware* (**LHW**) – Projeto do acionamento;
- Departamento de Engenharia Mecânica da **UFSC**: Laboratório de Robótica (**LAR**) – Projeto cinemático do manipulador;
- Departamento de Automação e Sistemas da **UFSC**: Laboratório de Controle e Micro-informática (**LCMI**) – Projeto do *software* e integração.

1.2 – Explanação do Problema

No Brasil, diferentemente de outros países, 92% da energia consumida é limpa e renovável (LEAL, et al., 2005). Esta energia é gerada através de usinas hidrelétricas, figura 2.

Figura 2: Maquete de uma Usina Hidrelétrica.

A energia elétrica é gerada porque as turbinas das usinas, figura 3, recebem a força das águas dos rios represados atravessando-as e assim movimentando geradores elétricos tal que transforme energia hidráulica em energia elétrica levada à casa das pessoas e indústrias utilizando linhas de transmissão.

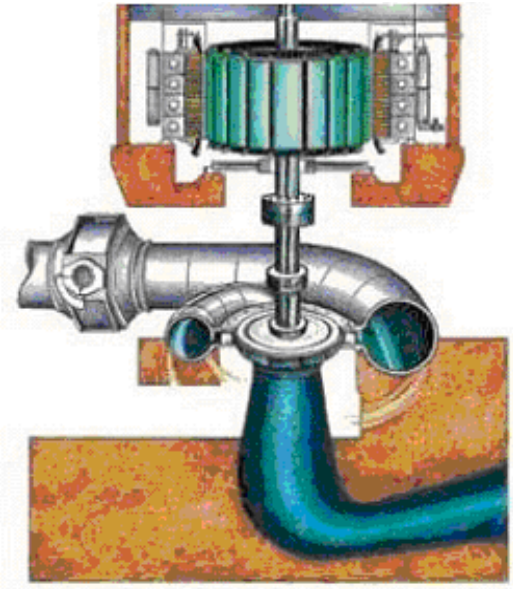


Figura 3: Turbina de uma Hidrelétrica.

Um dos elementos que faz parte da turbina, denominado rotor, figura 4, sofre erosão. E por seu tamanho e peso, é conveniente evitar sua substituição.

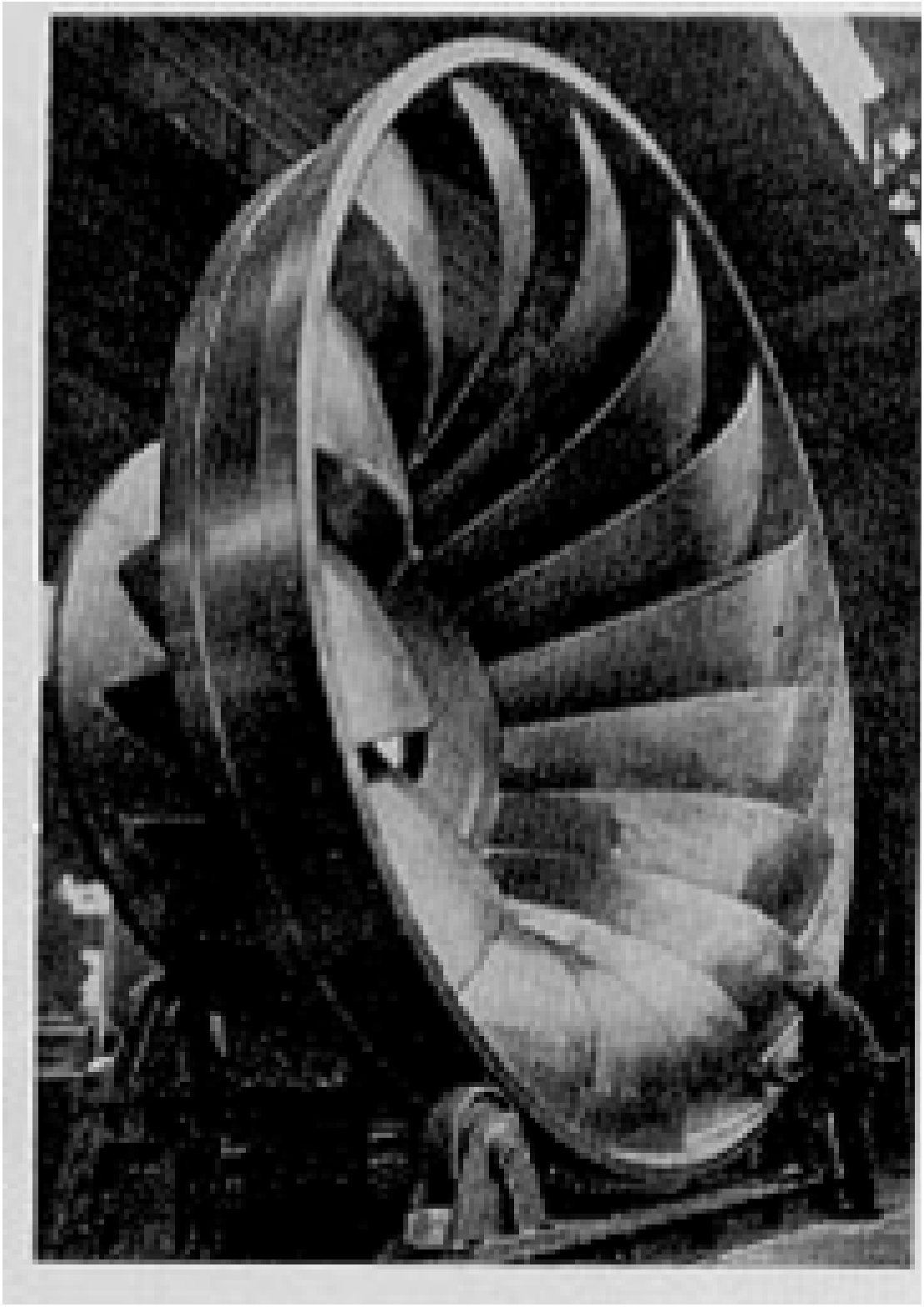


Figura 4: Rotor tipo Francis.

A erosão, figura 5, é ocasionada pelo fenômeno denominado cavitação. A cavitação é causada pela diferença de pressão ocorrida quando as bolhas de água passam através da turbina. Esta diferença de pressão causa a implosão destas bolhas contra as pás que compõem o rotor. A ação desta implica no desgaste das pás que, se não cuidadas, podem evoluir até perfurar o perfil de aço das pás de um rotor. Desta maneira, para não haver a perda do rotor, faz-se a recuperação de suas pás através do preenchimento com ligas de solda nas regiões afetadas pela cavitação.

Figura 5: Região erodida por cavitação.

Inicialmente, para realizar a recuperação é preciso tornar o local onde se localizam as crateras acessível à equipe que fará a recuperação. Como a erosão ocorre nas pás do rotor, local por onde passa a água que gera a energia, é necessário parar a turbina para a tarefa de reparo. Portanto, é feito o fechamento da entrada de água para, em seguida, ser drenada a água da parte interna da turbina. Uma vez que o ambiente foi drenado, é armada uma infra-estrutura para manutenção. Posteriormente, os processos de goivagem e esmerilhamento são realizados a fim de preparar a cratera para receber a soldagem. Uma vez esmerilhada, a cratera adquire uma superfície suave, própria para a aplicação da solda. Assim a recuperação poderá ser feita depositando-se então tantas camadas de soldagem quanto forem necessárias para o preenchimento da cratera. A etapa seguinte é um novo esmerilhamento para obter-se uma superfície lisa próxima ao perfil original da pá. A qualificação da soldagem é feita usando-se um líquido penetrante especial que identifica poros e evidencia crateras que precisarão ser refeitas. E por fim, uma vez que as crateras foram recuperadas, é desmontado o andaime, evacuado o local, o canal de água é liberado e a turbina é posta em operação novamente.

1.3 – Motivação à Concepção de um Controlador de Arquitetura Aberta

A **ISO** 8373 define que um robô industrial é formado pelo sistema de manipulação e o sistema de controle. E o controlador, segundo a **ISO** 8373:1994, é o sistema de controle do robô incluindo *hardware* e *software* (**ISO**, 2007). O controlador para robôs é formado pelo servo-amplificador de potência e pelo seu sistema computacional.

O projeto de um sistema automatizável para efetuar a recuperação de rotores de turbinas de hidrelétricas de grande porte erodidos por cavitação necessitava ter um controlador que fosse possível evoluir no decorrer de seu desenvolvimento. Entretanto, observou-se que não existia um controlador com características tais que pudesse servir ao projeto, pois os requisitos necessários que o controlador deveria suportar não eram conhecidos em sua totalidade. Assim, nasceu a motivação para idealizar uma arquitetura de sistema aberta embutida em um controlador, ou seja, permitisse adaptações em seus componentes à medida que o sistema automatizado fosse evoluindo.

A seguir serão listados os requisitos dos componentes que compreendem o controlador de arquitetura aberta para robôs no âmbito dos projetos citados e utilizada como universo de pesquisa no decorrer do desenvolvimento deste trabalho de tese. São eles:

- Customizar a interface com usuário;
- Sintonia da estratégia de controle;
- Interoperação do gerador de trajetórias;
- Modificação do modelo cinemático;
- Substituição de ferramentas e processos;
- Uso de diferentes morfologias de manipuladores;
- Adequação do sistema tempo real;
- Avaliação de desempenho dos robôs aplicados;

1.4 – Objetivos e Contribuições

A estrutura organizacional de um sistema é a arquitetura (**IEEE**, 1990). Um dos problemas fundamentais da robótica é que os robôs possuem uma arquitetura "proprietária" ou "fechada" (FORD, 1994) (KAHMEN, 2004) (KOREN, 2005) (PRITSCHOW, 2005). A concepção dos robôs não prevê modificações por pessoas que não o próprio fabricante. Isto torna inviável modificar suas características tanto de *software* quanto de *hardware*, mas também dificulta integrá-lo a outros dispositivos de fabricantes diferentes. Por exemplo, existem situações em que é necessário modificar ou integrar tanto aspectos de *hardware* quanto de *software*, tabela 1, de um sistema robotizado. E, conseqüentemente, pode-se ter um desempenho global inadequado às necessidades da atividade a qual o robô está sendo aplicado(BUTTAZZO, 1997).

Tabela 1: Exemplos de componentes de um Robô.

Hardware	*Software*
• Sensores	• Interface com usuário
• Atuadores	• Estratégia de controle
• Amplificadores de potência	• Gerador de trajetória
• Efetuadores	• Modelagem cinemática/dinâmica
• Estrutura mecânica	• Programação
• Barramento de comunicação	• Comunicação
• Computador	• Calibração juntas, elos, etc.
• Periféricos	• Ferramentas
• **HMI** (*human-Machine interface*)	• Processos
• *Teach Pendant*	• Executivo tempo real distribuído
• Sistemas de controle de movimento	• Sistema operacional

O padrão **IEEE** 1003 ou **POSIX** (*Portable Operating System Interface*) define um sistema aberto como:

"Um sistema aberto fornece as potencialidades que permitem as aplicações serem implementadas corretamente para executar em uma variedade de plataformas de diferentes fornecedores, inter-operar com outras aplicações do sistema e apresentam um estilo consistente de interação com o usuário".

A última versão da norma 1003.1, (**IEEE**, 1995), atualiza esta definição:

"Um sistema que implementa especificações abertas suficiente ou padrões para interfaces, serviços e suporte de formatos para permitir que o software de aplicação projetado possa:

- *Portar com mínimas mudanças através de uma larga faixa de sistemas a partir de um ou mais fornecedores;*
- *Inter-operar com outras aplicações de sistemas locais ou remotos;*
- *Interagir com pessoas em um estilo que facilite a portabilidade do usuário"*.

A proposta desta tese é uma arquitetura aberta para os controladores dos robôs. A originalidade do trabalho é a concepção de um modelo aberto baseado, sempre que possível, em soluções padronizadas já existentes. O objetivo é responder se é ou não possível utilizar o modelo proposto para flexibilizar a aplicação de um robô.

A implementação do modelo permite experimentá-lo em diferentes situações tal que se possam avaliar resultados reais, e assim, criar um sistema de controle aberto para robôs.

A aplicação deste robô aberto à recuperação de rotores de turbina hidráulicas de grande porte em usinas hidrelétricas é uma contribuição inédita, culminando na demonstração prática da proposta abordada no trabalho e resultando em consistência nas afirmações referentes à concepção de sistemas abertos na área de robótica.

1.5 – Estrutura do Texto

Neste capítulo introdutório foi apresentada a problemática dos sistemas abertos para robôs justificando as necessidades que levaram a concepção deste trabalho. Depois de comentados os problemas que levaram a realização do trabalho, foram apresentados a solução proposta e quais as contribuições científicas que a tese trará como resultado. A seguir, é descrita a organização da estrutura deste documento como forma de orientar o leitor.

O segundo capítulo apresentará o estado da arte em arquiteturas de controladores subdivididos em arquiteturas de controladores e arquiteturas de controladores abertos. Em seguida, serão comentados detalhadamente os tipos de controladores existentes juntamente com suas respectivas características. E, por fim, será feita uma comparação entre os controladores abertos destacando os pontos fortes e fracos de cada um deles com o objetivo de enfatizar o que falta ser feito neste contexto.

A proposta da tese será detalhada no capítulo terceiro, no qual será especificado o que foi realizado enfatizando as características inovadoras do trabalho. Posteriormente, será mostrada como foi feita a construção do modelo proposto, esclarecida qual a metodologia de implementação adotada e suas aplicações.

No quarto capítulo são relatados os resultados obtidos a partir das avaliações experimentais realizadas em laboratório e em campo no âmbito dos projetos citados.

Por fim, no quinto capítulo são apresentadas as conclusões. E a seguir citam-se as referências bibliográficas adotadas a fim de embasar o trabalho.

Capítulo 2

Estado da Arte em Arquiteturas de Controladores

A primeira máquina ferramenta com comando numérico surgiu no **MIT** (*Massachussetts Institute of Technology*) somente por volta de 1950. Esta se assemelha conceitualmente ao que hoje se define como o controlador de um robô industrial. Existem basicamente três tipos de variações de arquitetura aberta de controladores (PRITSCHOW, 1997). A primeira arquitetura de controlador é aberta no nível da interface homem-máquina; No segundo tipo de arquitetura aberta é oferecida a possibilidade de inserir funcionalidades no núcleo do sistema através da **API** (*Application Program Interface*) fornecida pelo fabricante; E, por fim, o terceiro tipo de arquitetura aberta é independente de fabricante. Neste caso, um grupo, através de um consórcio, define todos os aspectos de interface dos vários módulos que compõem o controlador.

2.1 – Tipos de Arquiteturas

Do ponto de vista do suporte computacional, o controlador de robô pode ser:

- Baseado em Controlador Dedicado ou Clássico: Existem sistemas distintos de computação. Por exemplo, um para gerenciar o sistema e outro para calcular as referências de controle dos atuadores das respectivas juntas do manipulador (NAKAMURA, et al., 1986).

- Baseado em *Personal Computer* – **PC** (computador pessoal ou compatível): O sistema computacional utilizado é o **PC**. Nele ocorre grande parte da computação do sistema (FIEDLER, 1998).

A figura 6 mostra um exemplo clássico de uma arquitetura baseada em controlador dedicado.

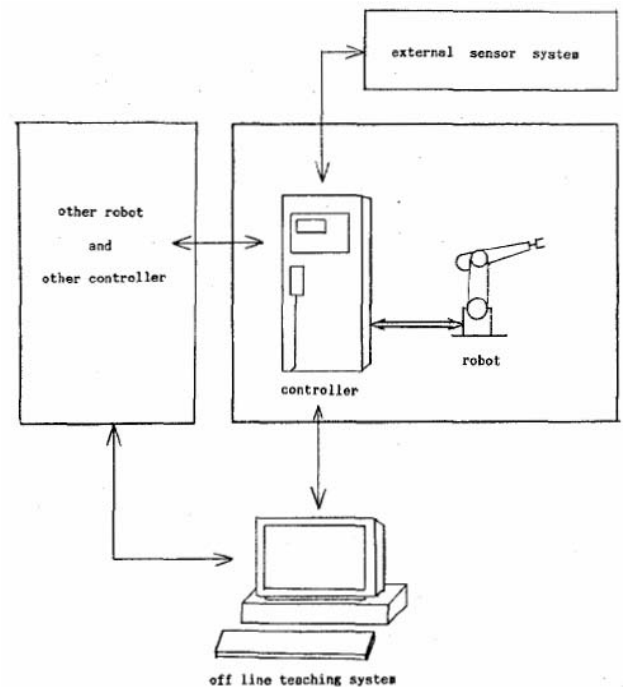


Figura 6: Configuração de um Sistema de Controle de Robô Clássico (NAKAMURA, et al., 1986).

Já em um controlador baseado em **PC**, seu diagrama difere pelo sistema computacional utilizado no robô, figura 7.

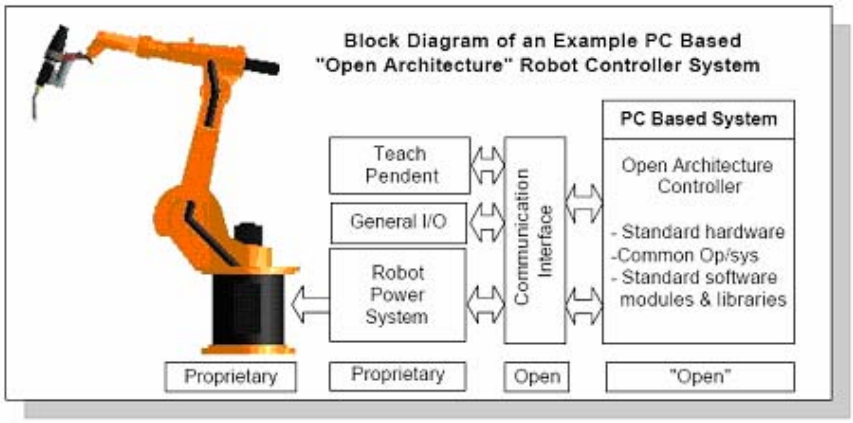


Figura 7: Configuração de um Sistema de Controle de Robô Baseado em **PC** (FIEDLER, 1998).

A partir dos anos 80 houveram várias iniciativas de criação de controladores abertos para máquinas ferramentas bem como para robôs. Efetivamente, somente na década de 90 é que se alcançou maturidade nas pesquisas e desenvolvimento de protótipos apresentando características de controladores abertos tanto na área de máquinas ferramentas como na robótica.

2.2 – Arquiteturas de Controladores

A seguir serão relatadas as principais iniciativas e sua evolução até os dias atuais no sentido de abrir um controlador seja para máquinas ferramenta ou para robôs, a partir dos trabalhos de FORD (1994) e outros (BRUYNINCK, 2001) (KOREN, 2005).

2.2.1 – SARIDIS

Uma arquitetura para máquinas inteligentes foi desenvolvida com o trabalho do professor SARIDIS (1979). Esta abordagem é baseada em uma estrutura hierárquica de controle em três níveis:

- Primeiramente, o nível organizacional por razões de abstração é modelado por uma máquina de estados que compreende as atividades de planejamento e decisão;
- O segundo, nível de coordenação, é composto de uma estrutura modelada tecnicamente por rede de Petri que coordena as ações e realiza a interface entre o primeiro e o terceiro nível;
- Finalmente, o terceiro nível, denominado de nível de execução, consiste do sensoriamento e do *hardware* de controle de movimento que interage com os eixos do sistema.

A arquitetura é baseada no princípio de que as tarefas no nível organizacional são definidas como o nível de maior inteligência da máquina e abstração. E as tarefas pertencentes ao nível de execução são definidas para o nível de menor inteligência da máquina.

O sistema foi implementado sobre um robô projetado para servir de transportador de construções espaciais. O prof. Saridis mostra que a estrutura de *software* resultante é extremamente eficiente, versátil e capaz de operações remotas em relação a outras arquiteturas propostas (FORD, 1994).

A principal contribuição desta abordagem é o fato de estabelecer uma arquitetura baseada em três níveis de abstração. Além disso, neste trabalho foi introduzida a utilização de especificação formal através de redes de Petri na organização das atividades da máquina.

2.2.2 – NASREM

O modelo de referência padrão para arquitetura de sistemas de controle para tele-robôs (*Standard Reference Model for Telerobot Control System Architecture* - **NASREM**) é baseado em pesquisas da **NASA** (*National Aeronautics and Space Administration*), **NIST** (*National Institute of Standards and Technology),* **DOE** (*U. S. Department of Energy*) e universidades. O **NASREM** é uma das primeiras tentativas de especificação completa de todos os componentes de um sistema inteligente (MILLER, et al., 1991). O **NASREM** é um modelo conceitual decomposto também em três níveis de hierarquia(ALBUS, et al., 1989):

- Modelagem do mundo;
- Decomposição de tarefa;
- Processamento sensorial.

Nestes níveis são realizadas diferentes transformações matemáticas fundamentais. A arquitetura tem diferentes módulos. Cada módulo é uma máquina de estado finita que aceita entradas, calcula algumas funções baseadas nos estados e entradas e, finalmente, produz uma saída. As entradas consistem de comandos a partir de um nível mais alto, dados sensoriais do mesmo nível e estado/dados a partir de níveis mais baixos. As saídas são comandas para os níveis baixos, processando dados dos sensores para o mesmo nível, e dados/estado realimentados para níveis acima. Esta arquitetura fornece uma metodologia para implementação de sistemas de grande porte. Diferentes sistemas baseados no **NASREM** têm sido apresentados para vários patrocinadores do projeto. Existem várias aplicações que utilizam o **NASREM** (FIALA, et al., 1987) (LEAKE, et al., 1989) (CHACONAS, et al., 1990) (LUMIA, 1989) (LUMIA, et al., 1990) (LUMIA, 1994) (LUMIA, 1994) (RESSA, et al., 1991) (ALBUS, et al., 1994).

A principal contribuição desta abordagem é o fato de estabelecer o conceito da necessidade de uma arquitetura de referência e conseqüentemente, uma metodologia para o desenvolvimento de **OAC** (*Open Architecture Controller*). Outros subsídios desta proposta foram o pioneirismo ao introduzir a preocupação e efetivação da padronização dos componentes utilizados no desenvolvimento de sistemas de controle aberto juntamente com a validação através de um grande número de aplicações.

2.2.3 – NGC

O centro nacional de ciências da manufatura (*National Center for Manufacturing Sciences*) e a força aérea dos Estados Unidos da América (*U. S. Air Force*) co-patrocinaram o programa **NGC** (*Next Generation Controller*). A empresa *Martin Marietta Corp.*, atualmente denominada *Lockheed Martin*, coletou as necessidades da indústria (MARTIN MARIETTA, 1992) e preparou uma especificação para um padrão de arquitetura de sistemas abertos (*Open System Arquitecture Standard* – **SOSAS**) na área de máquinas ferramentas. O próximo passo depois da conclusão do **SOSAS** foi dado a partir do **NIST** buscando o **EMC** (*Enhanced Machine Controller*) (PARK, et al., 1995).

O **NGC** é baseado em um modelo de máquina virtual no qual cada módulo tem especificação publicada para suas funcionalidades e interfaces. Isto permite estender ou modificar as capacidades de um controlador sem a necessidade de re-projeto.

Existem três conceitos contidos no *framework*. O primeiro é uma arquitetura de integração que fornece um conjunto de serviços para as aplicações. O segundo é um conjunto de quatro aplicativos que formam a base funcional de um controlador de máquina ferramenta com facilidades implementadas utilizando o mecanismo de passagem de mensagem denominado linguagem de manufatura neutra (*Neutral Manufacturing Language*). Finalmente, o ambiente de integração e configuração fornece um conjunto padrão de ferramentas para aumentar, modificar e integrar módulos no sistema.

Protótipos usando a abordagem **NGC** foram desenvolvidos. O **SOSAS** foi redefinido (LEAHY, et al., 1994) com enfoque na padronização de *hardware* e *software* e utilização dos paradigmas da orientação a objetos. As aplicações estão sendo substituídas por componentes expressos em termos de responsabilidades, necessidades (entrada) e produto (saída).

A principal contribuição desta abordagem foi ter sido a precursora na preocupação em definir quais as necessidades de um **OAC,** para só então, posteriormente, criar a arquitetura propriamente dita.

2.2.4 – DICAM

O **DICAM** (*Distributed Intelligent Control and Applications Management*) é a versão militar do **NASREM** (TATE, 2001). A *CimFLex Teknowledge Corp*. concebeu a tecnologia associada a uma metodologia para o desenvolvimento rápido de controladores inteligentes de alto desempenho (HAYESROTH, et al., 1992a). O objetivo do controlador é ser empregado em um controlador inteligente distribuído e servir de gerenciador de aplicações. Esta pesquisa foi uma iniciativa de domínio específico da arquitetura de *software* que faz parte da agência de projetos de pesquisas avançadas americana (HAYESROTH, et al., 1992b). A empresa juntou as melhores idéias das áreas de engenharia do conhecimento e engenharia de *software* para aplicar no desenvolvimento de um *software* de controle tempo real inteligente bem como em seu próprio processo de desenvolvimento. Esta arquitetura de controle genérica é orientada a tarefa e cria um meta controlador para realizar as atividades de escalonamento de tarefas. O projeto tem quatro elementos:

- A arquitetura de referência para controle inteligente;
- Um ambiente de desenvolvimento (*Integrated Development Enviroment* - **IDE**) para construção de aplicações a partir de componentes;
- Os módulos reutilizáveis para um repositório de construção de aplicações;
- Um conjunto de ferramentas de desenvolvimento que incorporam várias técnicas da engenharia de *software* (lei de controle especificas, gerador de código, protocolos, compiladores, depuradores) e da engenharia do conhecimento (modeladores de domínio, gerenciadores de necessidades, assistentes de projeto para base de conhecimento).

A CimFlex apresentou em 1992 o primeiro protótipo de seu **IDE**. Ela mostrou como os requisitos do controlador podem direcionar o processo de desenvolvimento de *software* a uma arquitetura de referência do controlador. Atualmente, existem algumas aplicações que utilizam o **DICAM** (METTALA, et al., 1992) (TERRY, et al., 1994a) (TERRY, et al., 1994b). O **DICAM** é uma conseqüência do **NASREM**, porém mais robusto. Sua principal contribuição é o fato de introduzir e estabelecer a proposta de utilização de **IDE** no desenvolvimento e implementação de um **OAC**. E reforçar o conceito de que um modelo, para ser concebido, deve-se ater primeiramente a quais serão suas respectivas necessidades ou especificação antes mesmo de ser criado.

2.2.5 – CHIMERA

O departamento de engenharia elétrica e computação da *Carnegie Mellon University* (**CMU**) produziu um *framework* de *software* fundamentado no conceito de *software* dinamicamente re-configurável. A abordagem combina projeto orientado a objetos com a idéia de portas (STEWART, et al., 1992) (STEWART, et al., 1997).

Um objeto baseado em portas é definido como um objeto que tem várias portas para comunicação. Os estados internos e métodos são ocultos para outros objetos. Somente as portas de um objeto são visíveis para outros objetos. A base de dados global de informação do estado é o mecanismo pelo qual os módulos trocam informação. A base de dados é implementada utilizando compartilhamento de memória global. A tabela de estado global, que contêm todas as variáveis de entrada e saída das portas de cada módulo, está armazenada na memória compartilhada. Tarefas correspondentes a cada módulo de controle não podem acessar a tabela diretamente. Preferencialmente, cada tarefa tem sua própria copia local da tabela. Somente as variáveis usadas pela tarefa são mantidas atualizadas. No início de cada ciclo da tarefa, as variáveis que são portas de entrada são copiadas a partir da tabela compartilhada. No fim de cada ciclo das tarefas, as portas de saída são bloqueadas toda vez que uma tarefa está transferindo dados entre sua tabela e a tabela global. O mecanismo de tabela de variáveis de estado global foi incorporado no sistema operacional tempo real *Chimera 3.0* desenvolvido pela **CMU**.

Recentemente, a **CMU** está trabalhando conjuntamente com a **ABB** no sentido de desenvolver um controlador aberto para robô aplicado. Este trabalho começou a partir do relatório que modela a problemática envolvida na concepção de um **OAC** e, conseqüentemente, espera-se, a partir do problema modelado, achar um modelo de solução (HISSAM, et al., 2004).

Esta abordagem tem como contribuições a utilização dos paradigmas relacionados à orientação a objeto (**OO**) no desenvolvimento e implementação de **OAC** para robôs, e além disso, o conceito de portas, semelhante ao que se conhece atualmente por componente, bem como o tratamento de tempo real à área de sistema de controle para robôs.

2.2.6 – GISC

Os laboratórios do Sandia e o **DOE** desenvolveram um conceito de controladores de sistemas inteligentes genéricos (*Generic Intelligent System Controller* – **GISC**). O **GISC** é baseado em comunicação e segue a premissa de que em sistemas inteligentes sofisticados o desempenho é alcançado pela coordenação de uma coleção de subsistemas semi-autônomos, cada um com capacidades complementares (FORD, 1994). Assim, um programa de controle supervisório coordena as atividades do sistema através deste subsistema de interfaces. Subsistemas individuais podem também possuir funções de controle de baixo nível tempo real, que podem ser realizadas de maneira autônoma e assíncrona. Com a correta combinação do supervisor e das capacidades dos subsistemas, tal abordagem suporta a implementação de controle baseado em modelo e integração de sensores com estrutura de *software* re-configurável.

O **GISC** possui os seguintes componentes: **ISOE** (*Intelligent System Operating Enviroment*), que fornece as facilidades de comunicação baixo nível e a interface geral para o supervisor e subsistemas (DIAZ-CALDERON, et al., 1999); **GENISAS** (*General Interface for Supervisor and Subsistems*), que fornece a *framework* para comunicação alto nível necessário para supervisão distribuída e arranjo dos subsistemas (DIXON, et al., 2003). Ele utiliza a abordagem cliente/servidor para dividir o sistema em processos separados que se comunicam através de mensagens; **RIPE/RIPL** (*Robot Independent Programming Enviroment and Language*), que permitem o desenvolvimento de subsistemas genéricos através de interfaces orientadas a objetos para dispositivos de sistemas inteligentes (MILLER, et al., 1991); **SMART** (*Sequential Modular Architecture for Robotics and Teleoperation*), que fornece um sistema de controle básico com desempenho e flexibilidade para controle baseado em sensores e tele-operação incluindo reflexão de força (ANDERSON, 1993); Por fim, o ambiente de programação gráfica, denominado de Sancho, que disponibiliza uma interface com o operador baseado em *menu* re-configurável e um ambiente de simulação *on-line*.

Esta abordagem tem como contribuições encorajar o uso de modularidade, ambiente multiprocessado distribuído e interfaces comerciais padronizadas utilizando os paradigmas da programação orientada a objetos.

2.2.7 – NOMAD

A empresa Trellis desenvolveu módulos de *software* para controle de movimento chamados NOMAD, que podem juntamente com outros *softwares* e *hardwares* servir para a concepção de controladores de arquitetura aberta para robôs (FORD, 1994). O sistema operacional para suporte ao desenvolvimento é o LynxOS utilizando a linguagem C para programação e X Windows como base para a criação da interface gráfica com o usuário. O sistema NOMAD, é composto por:

- Gerador de trajetória
- **IDE**
- Utilitários

O papel do gerador de trajetória é fornecer posição aos servo-controladores a partir das interfaces de entrada e saída. O **IDE** consiste de bibliotecas para comunicação em C, juntamente com ferramentas de configuração do sistema e programas exemplos. Os utilitários que compreendem o restante do NOMAD são: ferramenta gráfica para sintonia de controladores, simulador, interface linha de comando, *teach pendant* e etc.

Em 2001, a empresa americana localizada em *Rochester Hills*, Michigan, foi vendida a KUKA. A Trellis *Software and Control* passou a ser chamada KUKA *Development Laboratories, Inc*.

Atualmente, esta é uma das concepções que conseguiram evoluir no sentido de disponibilizar um robô utilizando alguns dos conceitos de arquitetura de controlador aberto. As principais contribuições do NOMAD são sua disponibilidade e aplicabilidade através da KUKA e o seu **IDE** que serve de suporte ao processo de desenvolvimento e a implementação de um **OAC** para robôs.

2.2.8 – ROBLINE

A empresa Cimetrix concebeu um controlador de arquitetura aberta para máquinas baseado em padrões de *hardware* e *software* utilizando o sistema operacional *Lynx* denominado de ROBLINE. Além do processador principal existe um **DSP** (*Digital Signal Processor*) para cada servo eixo e para monitorar as entradas/saídas (MATSUMOTO, et al., 2000).

O controlador é baseado no modelo cliente/servidor que divide o ambiente de programação das funções de controle de baixo nível. Este sistema de controle é dividido em vários processos que se comunicam através de mensagens. O servidor consiste de processos *multithread* para: o seqüenciador de tarefas, o planejamento de movimento e entradas/saídas. O cliente pode ser outro componente do sistema de controle, uma ferramenta geral ou um programa de aplicação. Além disso, existe a possibilidade de trabalhar com o sistema real ou simulação gráfica da máquina.

Atualmente, a Cimetrix tem produtos na área de controladores de arquitetura aberta para máquina ferramenta com aplicações em diversos segmentos da industrial. Além da existência de exemplos utilizando em retífica do ROBLINE (YANG, et al., 1997) (PI, et al., 1998).

As principais contribuições desta arquitetura aberta de controlador são a sua disponibilidade na forma de um produto e também de exemplos diversificados de aplicações na área de máquinas ferramenta. Outra contribuição é o fato de seus componentes utilizarem fortemente padrões, tanto do ponto de vista de *software* quanto de *hardware*.

2.2.9 – UMOAC

A universidade de Michigan, através dos trabalhos dos professores Dr. Yoram Koren e Dr. Sushil Birla, foi uma das primeiras a desenvolver e aplicar arquitetura de controlador aberto baseado em **PC** para uma máquina de comando numérico computadorizado (**CNC** - *Computer Numerical Control*) fresadora de 5 eixos. No projeto *Flander`s*, que começou em 1986 e terminou em 1993, os módulos de servo controle, interpolação, interpretador de código G e interface com usuário foram implementados usando como infra-estrutura um computador **PC** Intel 486 com sistema operacional MS-DOS. Ao final do projeto, o foco de estudo foi o projeto orientado a objeto para controladores de máquina ferramenta e comunicação interprocesso entre os módulos tempo-real usando os sistemas operacionais QNX e Windows NT.

As atividades de pesquisa continuam no Centro de Pesquisa em Engenharia (*Engineering Research Cente*r – **ERC**) na área de Sistemas de Manufatura Re-configuráveis (*Reconfigurable Manufacturing Systems* – **RMS**). Os projetos apontam para o **UMOAC** (*University of Michigan Open Architecture Controller*), que seria a unificação padronizada de controladores de arquitetura aberta e aplicações (KOREN, 2005). Os tópicos de pesquisa incluem:

- Projeto de controle de máquinas baseado em autômato,
- Implementação de controle supervisório e
- Desenvolvimento de interfaces homem-máquina.

As principais contribuições desta abordagem são sua aplicação dos conceitos relacionados, inclusive de manufatura re-configurável, aos **OAC** bem como sua especificação de infra-estrutura utilizada nas aplicações na área de máquinas ferramenta.

2.2.10 – OROCOS

A idéia de começar um projeto de *software* livre para controle de robô denominado **OROCOS** (*Open Robot Control Software*) (BRUYNINCK, 2001) foi motivada por experiências decepcionantes na tentativa de usar *software* de controle comercial para robô em pesquisas avançadas de robótica. Assim, uma proposta foi lançada na **EURON** (*European Robotics Network*) e tornou-se um projeto financiado pela União Européia - **UE** com duração de dois anos envolvendo o departamento de engenharia mecânica da universidade **K.U. Leuven** (*Katholieke Universiteit Leuven*) na Bélgica, o **LAAS** (*Laboratory for Analysis and Architecture of Systems*) do **CNRS** (*Centre National de la Recherche Scientifique*) em Toulouse na França, e o Instituto de Tecnologia **KTH** (*Kungliga Tekniska Högskolan*) na Suécia.

No **OROCOS** existe uma hierarquia de classes para as funcionalidades básicas seguindo uma abordagem de projeto baseado em componentes. Um conjunto de bibliotecas de classe e um *framework* de aplicação oferecem funcionalidades genéricas para máquinas ferramentas e robôs, tais como: componentes de controle, geração de movimento e interpolação; cinemática e dinâmica; algoritmos de controle específicos para robôs; estimação e identificação; etc.. Em novembro 2002, a primeira versão foi liberada com um sistema de controle de posição e velocidade para um robô de seis graus de liberdade. As seguintes aplicações disponíveis são: Máquina ferramentas cartesiana XYZ e Um robô com seis graus de liberdade com controle de força. No **KTH** em Estocolmo, diversas liberações foram feitas para sistemas robóticos baseados em componentes, e o projeto foi rebatizado de ORCA. Por causa de sua adequação às aplicações industriais, a *framework* do **OROCOS** *real-time* cresceu no campo de controle de máquina e afastou-se de sua origem na robótica.

A principal contribuição do **OROCOS** é a abordagem no desenvolvimento e implementação de sua arquitetura como *software* livre e assim em pouquíssimo tempo, quando comparada com outras abordagens, disponibilizar uma aplicação de fato na área de **OAC** para robótica.

2.3 – Arquiteturas de Controladores Abertos

Houve várias iniciativas por parte de universidades, centro de pesquisas e empresas em todo o mundo a fim de conceber uma arquitetura de controlador aberto. Tais iniciativas culminaram em três abordagens que padronizam internacionalmente as necessidades que envolvem um sistema aberto para automação. A primeira delas consiste no projeto **OSACA (***Open System Architecture for Controls within Automation Systems),* advindo da Comunidade Européia. A segunda é o projeto **OMAC (***Open Modular Architecture Controllers*), originário do Estados Unidos e, a última é denominada projeto **OSEC** (*Open System Environment for Controllers*), do Japão. Assim como nas outras áreas a robótica também fez uso do **PC** e dos conceitos de sistemas abertos, como será mostrado a seguir. Fundamentalmente, as abordagens para controladores abertos para robôs são divididas em: Definições, Infra-estrutura, Arquitetura de Referência, **API** e Aplicações. Estes aspectos serão abordados em cada uma das propostas de padronização de controladores de abertos nas secções seguintes.

2.3.1 – OSACA

A iniciativa pioneira na concepção formal de um controlador com arquitetura aberta para a automação foi denominado **OSACA**. O objetivo inicial do **OSACA** (PRITSCHOW, et al., 1994) foi reunir universidades, fornecedores de sistemas de controle e construtores de máquinas ferramenta imbuídos em conceber uma arquitetura de controlador aberto independente de fabricante. O projeto **OSACA** é uma tentativa européia de desenvolver um controlador de arquitetura aberta para máquinas ferramenta bem como robôs, que surgiu em 1992 e finalizou oficialmente em 1998. A frente deste projeto há o **ISW** (*Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen)* do FISW em Stuttgart representado pelo Prof. Dr. Gunther Pritscow, Prof. Dr. Wolfgang Sperling, Prof. Dr. Peter Lutz, **WZL** (*WerkZeugmachinenLabor*) da RWTH em Aachen e as empresas Siemens, Bosch, Comau, dentre outras.

Definições

Existe uma grande variabilidade de sistemas de controle que podem ser utilizados em sistemas abertos. Isto requer uma definição clara de sistemas de controle abertos. A partir do ponto de vista do usuário, *openness* é uma característica focada em potencialidades para integrar, estender e reusar os módulos de *software* em sistemas de controle, como apresentadas na tabela 2. As potencialidades requeridas têm que ser fornecidas principalmente pela plataforma do sistema de controle (SPERLING W, 1995).

Tabela 2: Capacidades dos módulos em Sistemas de Controle Aberto.

Capacidades dos módulos	**Significado**
Portável	Um módulo pode executar em diferentes sistemas de controle.
Estendível	A funcionalidade de um módulo pode ser estendida.
Intercambiável	Um módulo pode ser substituído por outro com funcionalidades comparáveis.
Escalável	Múltiplas instâncias dos módulos são possíveis para aumentar desempenho.
Interoperável	Cooperação de módulos (Troca de dados).

Generalizando, pode-se dizer que existem três níveis de “abertura” de um sistema que podem ser definidos com base na disponibilidade dos resultados das pesquisas na área de arquitetura de controladores abertos, como mostra a figura 8.

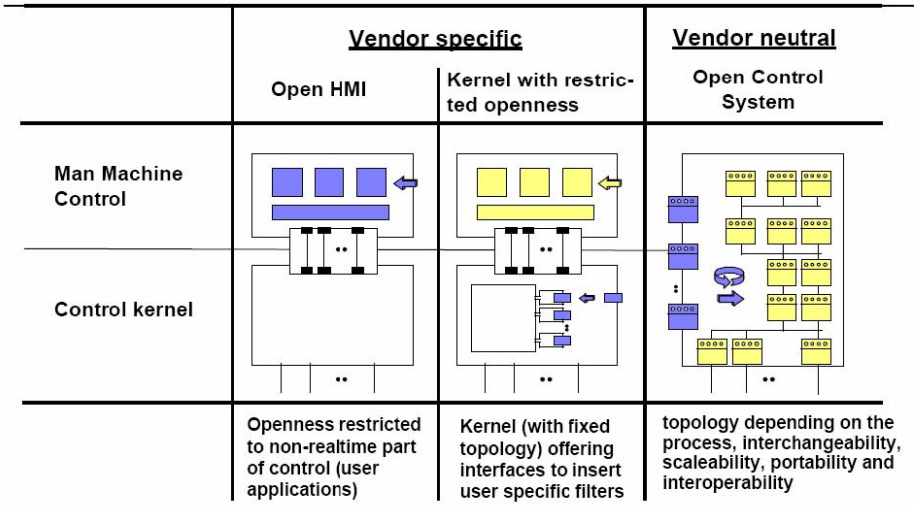


Figura 8: Diferentes tipos de arquiteturas de controladores abertos (PRITSCHOW, 2005).

Classificação dos diferentes tipos de arquitetura de controladores abertos (PRITSCHOW, 2005):

- A primeira arquitetura é aberta no nível de interface humano-máquina.
- No segundo tipo de arquitetura aberta é oferecida a possibilidade de inserir funcionalidades no núcleo do sistema através da **API** fornecida pelo fabricante.
- E, por fim, o terceiro tipo de arquitetura aberta é independente de fabricante. Neste caso, um grupo, através de um consórcio, define todos os aspectos de interface dos vários módulos que compõem o controlador.

Infra-estrutura

As funcionalidades da infra-estrutura ou plataforma do sistema, tabela 3, podem diretamente ser derivadas das necessidades impostas ao sistema de controle aberto (SPERLING W, 1995).

Tabela 3: Necessidades para uma plataforma de sistema de controle aberto (SPERLING W, 1995).

Capacidades dos módulos	**Significado**
Portável	Uniformidade de **API** entre plataformas
Estendível	A funcionalidade de um módulo pode ser estendida.
Intercambiável	Sistema de configuração modular.
Escalável	Múltiplas instâncias a partir do sistema de configuração.
Interoperável	Padronização de protocolo referente ao sistema de comunicação.

Três elementos básicos devem estar presentes em uma plataforma para o sistema de controle aberto (LUTZ, 1997):

- Sistema Operacional: garante a execução paralela dos módulos e assegura a independência a partir de um *hardware* específico
- Sistema de Comunicação: assegura a cooperação dos módulos de maneira padronizada.
- Sistema de Configuração: constrói uma topologia de *software* a partir de um conjunto de módulos disponíveis, através da instanciação e conexão destes diferentes módulos.

Estes três elementos integrados na plataforma são acessíveis através da **API**. Por causa da natureza orientada a objetos do sistema, incluindo encapsulamento e múltiplas instâncias, os módulos de aplicação serão chamados de objetos da arquitetura (*Architecture Object* - **AO**), como apresentados na figura 9.

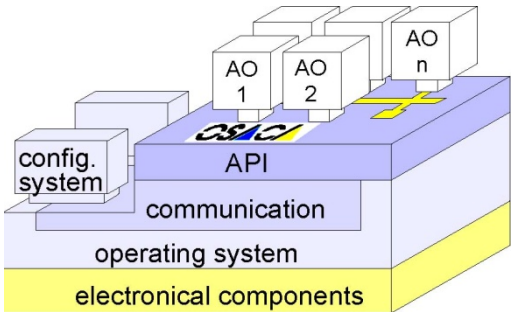


Figura 9: Arquitetura de Controle para Sistemas Abertos.

A base para toda a plataforma do sistema é o *hardware*, consistindo de placas processadoras, placas de entrada e saída e outros equipamentos periféricos. Como o **POSIX** (ISAAK, 2005) é o único padrão estabelecido para sistemas operacionais que inclui também definições tempo real, ele foi selecionado como suporte utilizado pelo **OSACA**. O sistema de comunicação é o único meio que a plataforma do sistema utiliza para o intercâmbio de informação entre **AO**'s. Assim, ele deve suportar a troca de informação entre os **AO**'s situados na mesma placa processadora bem como entre os **AO**'s situados nas placas processadoras diferentes conectadas através de um barramento de comunicação. Um protocolo padronizado tem que ser definido para assegurar os formatos de dados uniformes e um conjunto fixo de mensagens. A arquitetura do protocolo escolhida é derivada do modelo de referência **OSI** *(Open Systems Interconnection)*. Entretanto, ela compreende somente duas camadas, conforme a figura 10.

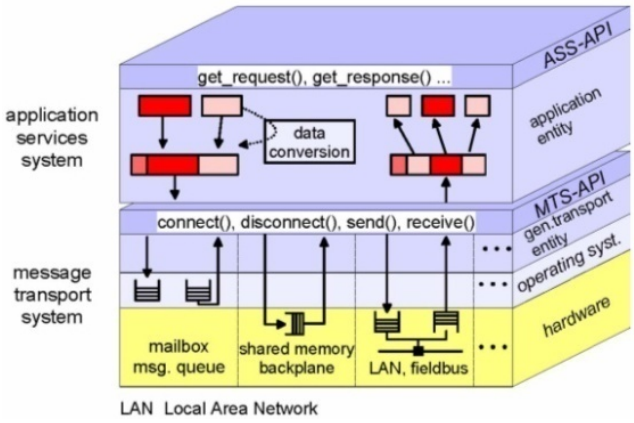


Figura 10: Sistema de Comunicação (LUTZ, 1997).

A primeira camada, denominada sistema de transporte de mensagens (**MTS** – *Message Transport System*) é equivalente às camadas de 1 a 4 do modelo **OSI**. A segunda, denominada serviços à aplicação (**ASS** – *Application Services System*) equivale às camadas de 5 a 7 do modelo **OSI**.

A **MTS** oferece serviços orientados a conexão para transporte transparente de mensagens arbitrárias entre **AO**'s. Ela pode ser adaptada para usar qualquer tipo de mecanismos existentes para a troca de informações em serviços do sistema operacional e protocolos de rede local.

A **ASS** tem a tarefa de gerir o protocolo da aplicação. Isto inclui o gerenciamento da conexão, a montagem e desmontagem das mensagens, e a conversão de dados entre diferentes representações. O protocolo de aplicação é realizado baseado em uma estrutura cliente/servidor usando princípios de programação orientada a objeto.

Em um servidor **AO,** cada informação, dados ou serviços que serão acessíveis externamente serão mapeados para um objeto de comunicação, como apresentado na figura 11.

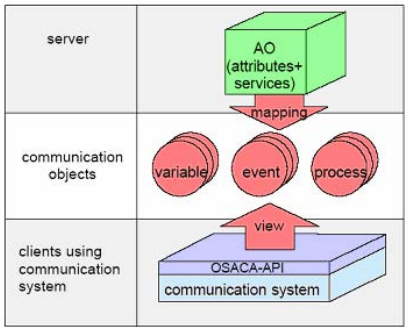


Figura 11: Objetos para Comunicação (LUTZ, 1997).

A partir do ponto de vista do cliente, o servidor é um conjunto de objetos de comunicação que podem ser acessíveis pelo uso de serviços **ASS** para enviar e receber mensagens. **AO**'s podem ser servidores e clientes ao mesmo tempo. Existe um conjunto fixo de classes para comunicação de objetos, as mais importantes são: *Variable*, *Event* e *Process*.

- Classe *Variable:* é responsável pela leitura e escrita de dados;
- Classe *Process:* é para disparar ações em máquinas de estados;
- Classe *Event:* é usada para envio de eventos e relatórios não solicitados.

Geralmente, a realização de uma configuração específica para um sistema de controle é de natureza estática. Antes da execução, o software é compilado e instalado no controlador. Esta solução é inflexível e requer esforços elevados se modificações posteriores forem necessárias. Para superar esta dificuldade, um sistema inteiramente modular é necessário, sendo a topologia real do *software* gerada na fase de inicialização do sistema caracterizando uma configuração dinâmica. Neste caso, a plataforma tem que conter um sistema de configuração que possa gerenciar uma biblioteca de classes de **AO`s**. Na inicialização, os **AO`s** das diferentes classes são instanciados e as conexões de comunicação entre **AO`s** são estabelecidas, como exemplifica a figura 12.

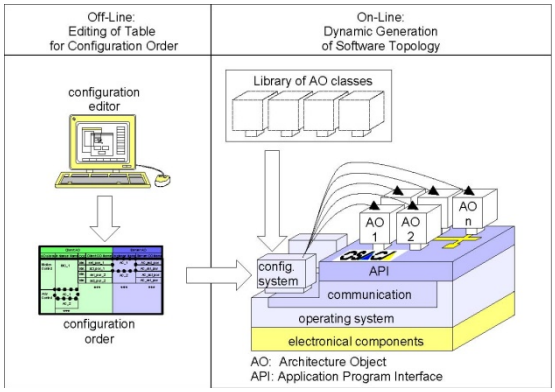


Figura 12: Elementos para Configuração Dinâmica de Sistemas de Controle (LUTZ, 1997).

A topologia real do sistema de controle é descrita e um arquivo com o pedido ou ordem de configuração é gerado para ser interpretado pelo sistema. Um editor gráfico pode ser usado para criar este arquivo, de forma semelhante ao sistema de configuração de um *driver* para dispositivo de *hardware* em um sistema operacional. A ordem de configuração contém uma lista de todas as instâncias dos **AO** que serão encontrados em um sistema de controle específico, figura 13.

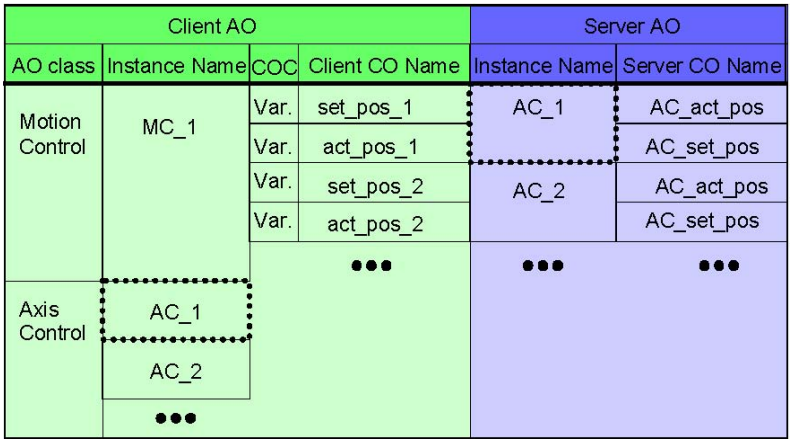


Figura 13: Pedido de Configuração (LUTZ, 1997).

É possível que existam diversas instâncias da mesma classe do **AO**, por exemplo, classe *Axis Control* no caso de uma máquina de dois eixos AC_1 e AC_2. Para cada **AO**, os relacionamentos *client/server* são definidos. Cada relacionamento entre um cliente e um servidor **AO** consiste em uma lista de objetos de comunicação do servidor que será utilizado através do cliente.

Arquitetura de Referência

Os objetos da arquitetura de referência (ESPRIT, 1996) (SPERLING, 1997a), **AO**´s, contêm o *software* de aplicação seguindo os preceitos da orientação a objetos. Estes objetos são agrupados de acordo com as suas características, como mostra a figura 14.

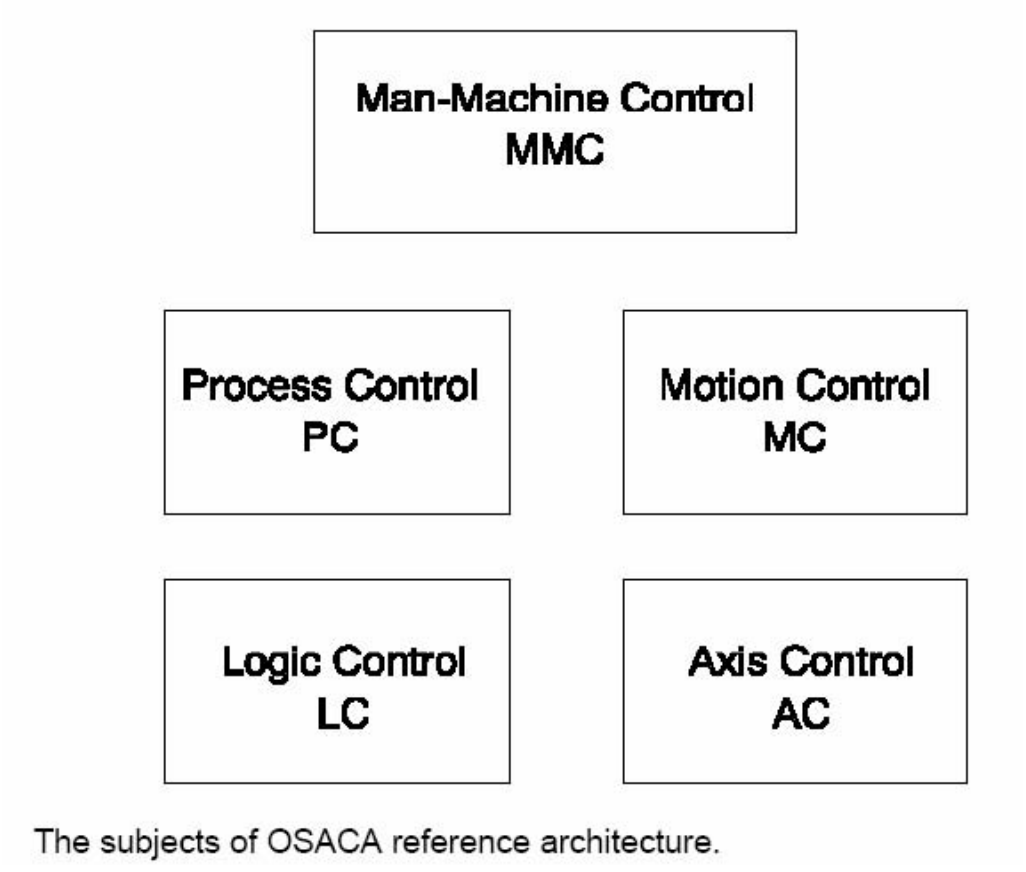


Figura 14: Arquitetura de Referência **OSACA** (ESPRIT, 1996).

A seguir será apresentado o significado de cada agrupamento de módulos do **OSACA** (ESPRIT, 1996).

MMC (*Man-Machine Control* - Controle Homem-máquina): Representa a máquina ou parte dela em relação a entidades externas, tais como o operador, sistema de supervisão sobre a rede, sistemas CAD/CAM, etc. Assim, o **MMC** permite a estas entidades externas controlar a operação da máquina. Ele pode também fornecer utilitários para preparação de tarefas para a máquina bem como para utilização geral e gerenciamento da máquina e de seus recursos computacionais.

MC (*Motion Control* - Controle de Movimento): Possibilita a máquina produzir movimento relativo com independentes graus de liberdade. Deste modo, emite comandos apropriados ao controle de eixos correspondente ao grupo de eixos da máquina. O movimento pode ser preciso, contínuo, sincronizado ou confinado.

LC (*Logic Controls* – Controles Lógicos): São responsáveis pela operação dos atuadores e sensores da máquina. Eles podem também assegurar a consistência das operações e a captura dos dados.

AC (*Axis Control* – Controle de Eixos): Incluem todos os meios necessários para ativação dos eixos a fim de executar comandos de movimentação, incluindo suas definidas restrições. Os comandos são muitas vezes cíclicos.

PC (*Process Controls* – Controles de Processos): Representa, quando necessário, os sistemas auxiliares da máquina. Eles são também responsáveis pelo gerenciamento de dados e processamento dos sistemas auxiliares que representam.

A arquitetura básica para todos os sistemas de controle modulares consiste em uma plataforma que contêm *hardware* e *software*, como o sistema operacional e um conjunto de módulos de software da aplicação contendo as funcionalidades específicas do sistema de controle representadas através de **AO**`s exibidos na figura 15.

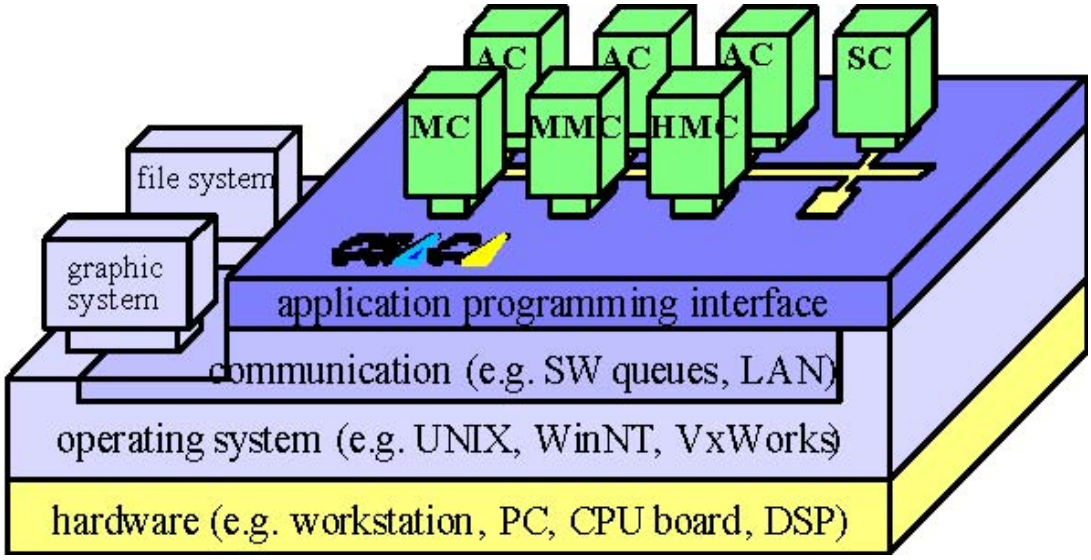


Figura 15: Arquitetura para Sistemas de Controle Aberto (SPERLING, 1997a).

A fim de aliviar programadores da aplicação de resolver problemas específicos de comunicação, uma camada uniforme de software foi introduzida para controlar os objetos de comunicação do **AO**. Este gerente de objeto de comunicação (*Communication Object Manager* - **COM**), figura 16, mantém uma lista dos objetos de comunicação disponíveis e executa pedidos de serviços sob eles.

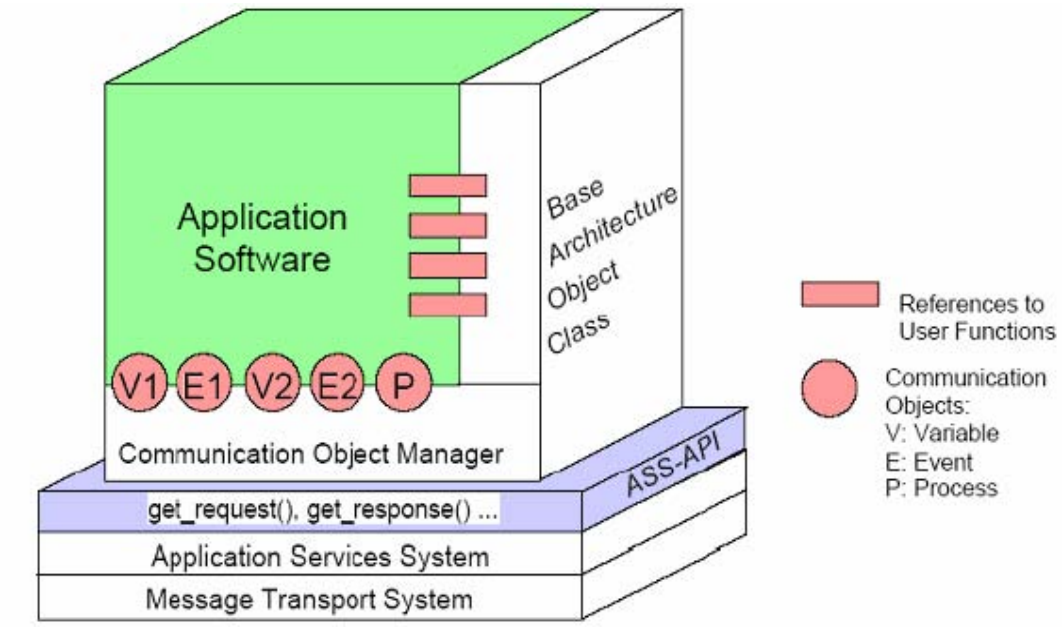


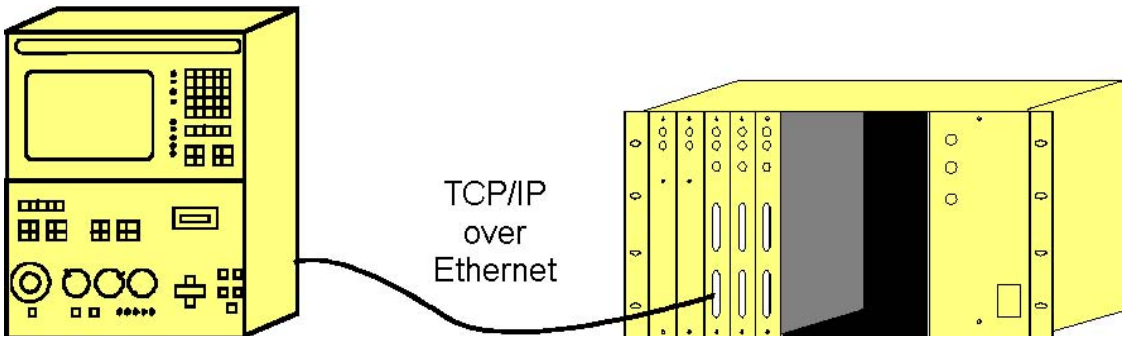
Figura 16: Arquitetura Básica para *Software* de Aplicação (SPERLING, 1997a).

API

A interface para programação de aplicações, **API,** é o *front-end* para o programador, conceitualmente semelhantemente a uma caixa preta. Por exemplo, a função matemática "*cos(x)*" especifica o nome da função, a seqüência de chamada e o parâmetro de retorno, e não como o *cosseno* é implementado, seja ele consultando uma tabela de valores ou por aproximação da série de Taylor. O que realmente importa na especificação da **API** é a função "assinatura" e suas chamada, e a seqüência de retorno. Portanto, a **API** permite também que os fornecedores de sistemas de controle implementem soluções adequadas e otimizadas, sem violar os critérios de sistemas abertos. É importante salientar que as **API**`s têm que ser independentes do fornecedor, a fim de viabilizar a portabilidade dos módulos de aplicação em sistemas de diferentes fornecedores.

Aplicações

A seguir serão descritos os principais resultados alcançados pela proposta OSACA através de exemplos, demonstrações e aplicações na indústria usando a mesma**A**. Uma configuração típica para sistemas de controle é o uso de **PC`s** que hospedam o painel do operador, componentes baseados em barramento **VME** (*VersaModule Eurocard*) para controle de movimento e dispositivos de entrada e saída com representados na figura 17. Ambas as partes são conectadas usando um barramento serial, neste caso *Ethernet* com **TCP/IP**. Esta configuração permite soluções eficientes usando componentes comercialmente disponíveis de *hardware* e de *software* (SPERLING, 1997b).



PC-based **VME**-bus based operator's panel motion control

Figura 17: Exemplo de Configuração: **HMI** e *Motion Control* (SPERLING, 1997b).

O sistema de controle aberto resultante consiste em plataformas divididas em duas partes: a parte baseada em **PC** no Windows 95 e a outra parte baseada em barramento **VME** usando uma placa processadora e o VxWorks como sistema operacional tempo-real. O sistema de comunicação permite que os **AO's** se comuniquem de maneira transparente, não importando se eles estão localizados na mesma ou em diferentes placas, tal como na figura 18.

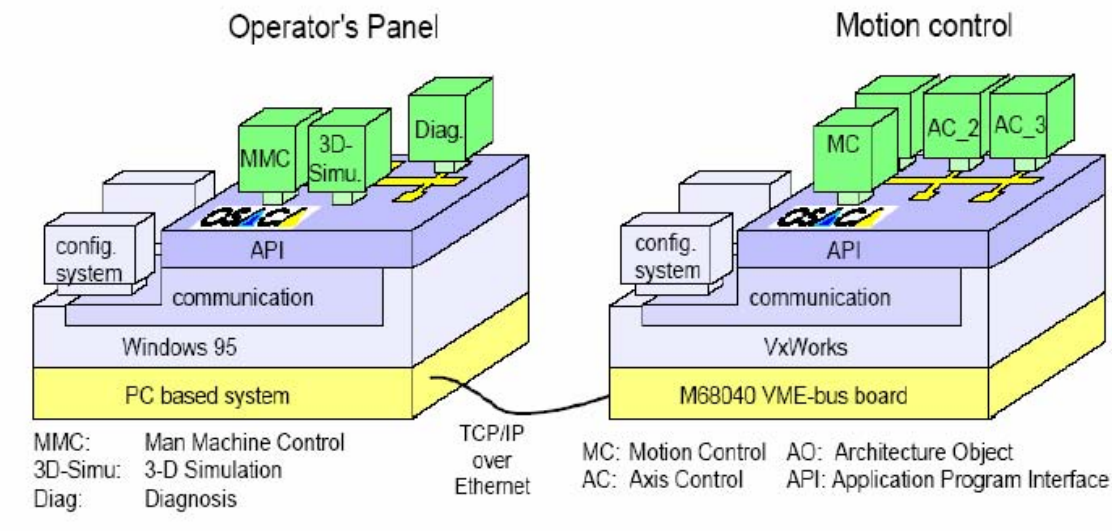


Figura 18: Exemplo de Plataforma (SPERLING, 1997b).

Diversas aplicações de propósito geral desenvolvidas originalmente para finalidades de demonstração e verificação foram melhoradas e podem ser aplicadas por usuários do **OSACA**. Os exemplos de tais aplicações são:

- Interface homem-máquina (**HMI**) que pode ser portável para diferentes ambientes e conectadas a diferentes fornecedores de sistemas de controle.
- Sistema de Diagnóstico que pode gerir diferentes mensagens de erro.
- Simulador 3D que permite visualizar os movimentos de uma máquina real.

Diferentes ferramentas foram implementadas a fim de suportar o processo de desenvolvimento para a arquitetura **OSACA**:

- Uma ferramenta de monitoração de dados que permite visualizar e manipular dados específicos de controle e ações relacionadas com teste de conformidade.
- Um protótipo de um sistema de configuração gráfica.

As primeiras demonstrações destas aplicações foram apresentadas durante o “dia aberto” do **OSACA**, que ocorreu em abril 1996 em Stuttgart, figura 19, e mais um exemplo em Aachen, na Alemanha, figuras 20 e 21(LUTZ, 1998).

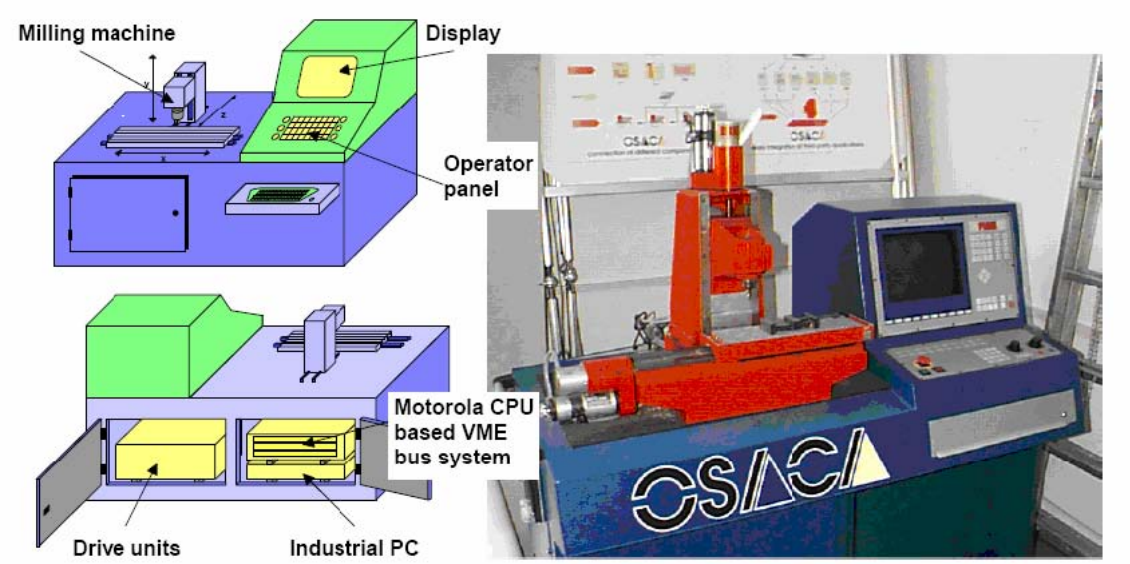


Figura 19: Primeira implementação **OSACA** no **ISW**, Stuttgart (LUTZ, 1998).

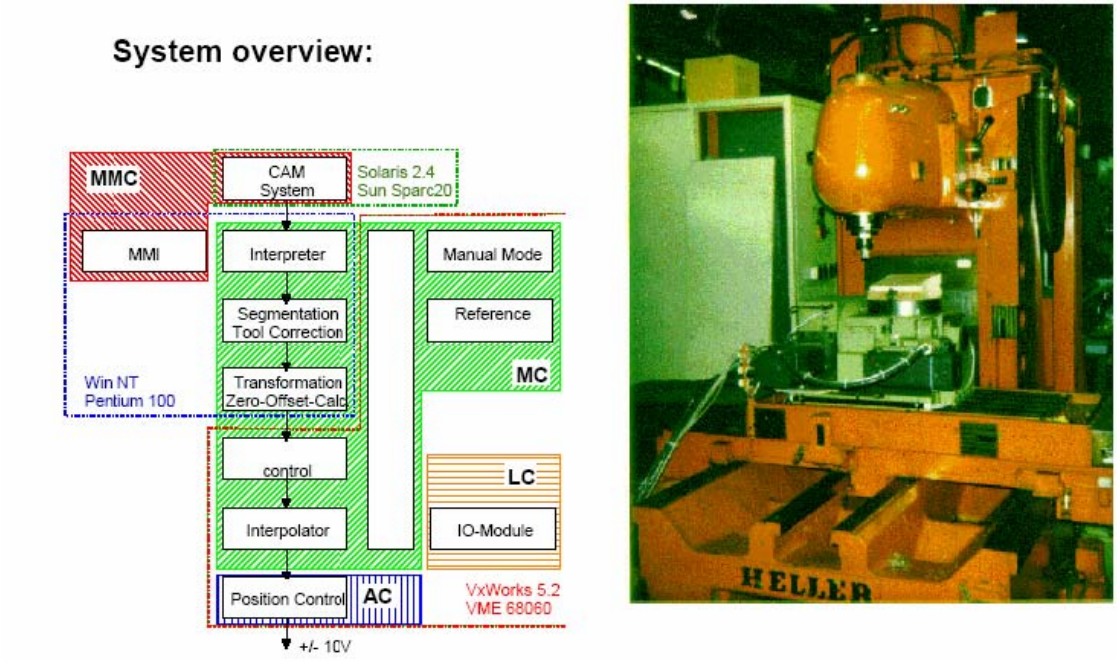


Figura 20: A primeira máquina ferramenta baseada no **OSACA** completamente funcional no **WZL**, Aachen (LUTZ, 1998).

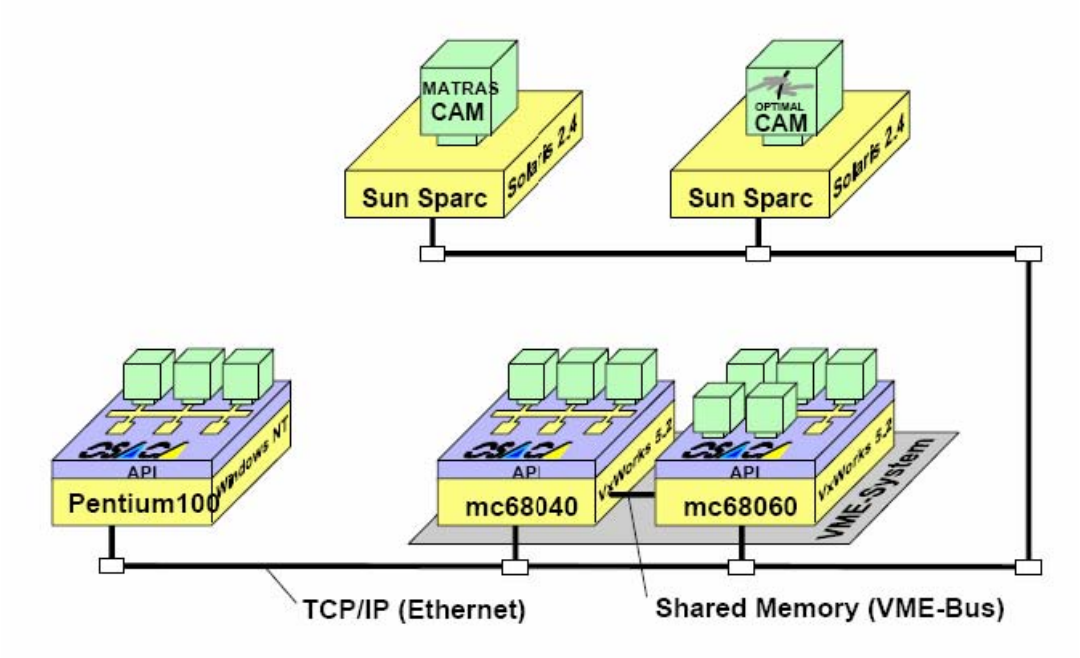


Figura 21: Plataforma **OSACA** no **WZL** (LUTZ, 1998).

Outra apresentação foi na EMO, em setembro de 1997, em Hannover. A partir do começo de 1998 o projeto HÜMNOS foi criado com o objetivo de aplicar os resultados obtidos com o **OSACA**. Como resultados finais do HÜMNOS foram concebidos e apresentados dois sistemas na BMW e *Daimler-Benz*. Estes pilotos estão sendo usados para testar os resultados em um ambiente industrial (PRITSCHOW, 1997).

Os sucessos das demonstrações e aplicações nas indústrias atestam a validade do **OSACA** (NACSA, et al., 1997). Além disso, a partir dos resultados obtidos no **OSACA,** foram concebidos os projetos HÜMNOS, OCEAN (BAUER, et al., 2001) e PC-ORC (HONG, 2001)**,** que são a continuidade do **OSACA**.

2.3.2 – OMAC

Devido ao grande interesse da indústria nas tecnologias de arquitetura de controladores abertos e modulares, a partir do trabalho intitulado “As necessidades de um controlador de arquitetura aberta e modular para aplicações na indústria automotiva” (versão 1.0 datada de 15 de agosto de 1994, publicado pelas companhias automotivas americanas) (YEN, et al., 1994), formou-se o grupo de usuários de controladores de arquitetura aberta e modular denominado **OMAC**.

O objetivo da especificação **OMAC** é permitir que fabricantes de controladores forneçam componentes padronizados para que fabricantes de máquinas possam facilmente configurá-las e integrá-los. Os usuários finais também devem fazer uso desta facilidade e facilmente re-configurar seus sistemas baseados em exigências de desenvolvimento da manufatura. Portanto, foram empreendidos esforços para definir uma especificação de arquitetura de um controlador aberto e **API**.

A visão de construir um sistema de controle escrevendo o código-fonte estaria sendo substituída pela construção de um sistema de controle a partir da montagem e integração de módulos ou componentes de *software* previamente existentes. Do mesmo modo, a noção de modificar um controlador reescrevendo o código-fonte seria substituída pela re-configuração ou substituição de seus módulos dentro do sistema de controle. Nesta visão baseada em componentes, um integrador de sistema seleciona um conjunto apropriado de componentes configurando-os e os integrado à aplicação (BIRLA, et al., 2001). Para realizar tal tarefa, baseando-se em componentes, foi necessária a definição de uma **API** de integração formal para estabelecer os relacionamentos e as colaborações entre componentes.

O projeto **OMAC** nasceu no Estados Unidos, mais especificamente na Universidade de Michigan, através dos Professores Dr. Sushil Birla e Dr. Yoram Koren, no **NIST** representada pelo Prof. Dr. John Michaloski, e na empresa GM, com o trabalho do Eng. Jerry C. Yen juntamente com as companhias Ford, Chrysler, Boeing, etc.

Definições

Os sistemas de controle não devem somente ser abertos e modulares, mas também escaláveis, econômicos, confiáveis, e manuteníveis (**OMAC** *Users Group*, 2002). Portanto, cada uma destas características é fundamental para o futuro na área de sistemas de controle abertos. Segue as definições propostas pelo padrão **OMAC**.

ABERTO

A incorporação de padrões, as disponibilidades de componentes de *hardware* e *software* são exigências importantes que devem ser consideradas para que um sistema ou controlador seja considerado aberto. Os sistemas abertos devem ser construídos usando a idéia dos sistemas de controle modulares abertos disponíveis no mercado. Estes padrões devem ser definidos rigorosamente em uma especificação pública, a fim de serem considerados componentes abertos. A utilização de interfaces padrão para conseguir abertura, modularidade e escalabilidade (*openness, modularidade* e *scalability)* é uma característica distintiva de sistemas baseados no **OMAC**. É exigido das interfaces padronizadas que sejam estendidas, além de usar placas com barramento de computador comercialmente comuns tais como **ISA** (*Industry Standard Architecture*), **PCI** (*Peripheral Component Interconnect*), etc. As interfaces padrão requeridas em um sistema **OMAC** incluem as **API**`s comuns para os módulos do *software* de controle, redes no nível de dispositivo, interfaces digitais, interfaces de rede em um nível mais elevado, e outros elementos contidos na arquitetura.

MODULAR

A exigência da modularidade refere-se à necessidade de recolocação fácil de componentes ou módulos do sistema de controle. Poder-se-á, simplesmente “plugando” um novo módulo, começar a operar com um mínimo esforço, similar ao carregamento de um novo *driver* de *software*. O “*Plug and Play*” neste sentido envolve a integração e a validação do sistema de controle sem a necessidade de esforço significativo. Este esforço de engenharia indesejável é freqüentemente necessário quando as mudanças em um processo de manufatura precisam sofrer modificação no sistema de controle ou quando os componentes de um antigo sistema se transformam obsoletos e necessitam ser substituídos.

As mudanças no processo podem ser implementadas na planta somente depois de uma validação completa e rigorosa das mudanças no sistema de controle.

ESCALÁVEL

A *scalability ou* escalabilidade do sistema de controle permite que os módulos sejam configurados, adicionados, e/ou removidos do sistema de controle a fim fornecer a potencialidade do controle requerida por cada aplicação. Isto permite que o sistema de controle acomode uma grande faixa de aplicações que podem possuir poucas ou muitas funcionalidades e assim se adapte a uma enorme variação de necessidades de um sistema de manufatura.

ECONÔMICO

Um aspecto importante na implementação de sistemas **OMAC** é a redução dos custos totais associados com um sistema de controle. Entretanto, tão importante quanto minimizar o custo inicial da aquisição, é reduzir o custo total sobre o ciclo de vida do sistema de controle. Os custos iniciais de implementação diminuirão como a adoção de tecnologias associadas com a indústria de computadores pessoais. Os sistemas modulares abertos permitem melhoramentos incrementais e a integração de componente de maneira fácil, reduzindo assim o custo na modificação do sistema de controle.

CONFIÁVEL E MANUTENÍVEL

Os sistemas de manufatura baseados no **OMAC** têm as mesmas exigências de *reliability* ou confiabilidade que um controlador proprietário. Os sistemas **OMAC** devem ser facilmente *maintainable* ou manuteníveis e ter recursos para maximizar o *uptime* e minimizar o *downtime* tal que satisfaça as necessidades do ambiente da planta. Além disso, necessitam também ser mantidos por indivíduos com habilidades e treinamento iguais ou mais baixos do que aqueles requeridos para a manutenção de sistemas fechados.

Infra-estrutura

A infra-estrutura trata primeiramente do ambiente computacional, incluindo os serviços da plataforma, o sistema operacional e as ferramentas de programação. Os serviços da plataforma incluem itens como temporizadores, gerenciadores de interrupção e comunicações interprocesso. O sistema operacional inclui a coleção dos serviços de *software* e *hardware* que controlam a execução dos programas de computador e fornecem serviços tais como alocação e controle de recursos, a entrada e saída de dispositivos, e a gerência de arquivos. As extensões de sistemas operacionais tempo real podem ser consideradas serviços da plataforma, já que são necessários para implementação de semáforos, escalonamentos preemptivos com prioridade, bem como comunicação interprocesso local e distribuída. As ferramentas de programação incluem compiladores, *linkers* e *debuggers* (BIRLA, et al., 1997). O **OMAC** não especifica uma infra-estrutura porque muitas delas acabam fora do domínio do controlador. Além disso, entende-se que este seria um esforço adicional desnecessário, já que elas poderiam ser mais bem especificadas por especialistas. Existem muitas competições entre tecnologias de infra-estrutura. Por isso, a **OMAC** resolveu deixar que o mercado decidisse qual irá utilizar.

Arquitetura de Referência

O **OMAC** não tem uma arquitetura explícita de referência. Ao invés disso, ele adota a tecnologia baseada em componentes a fim de suportar seus módulos. Em um nível mais elevado, o conjunto dos módulos do **OMAC** em um sistema requer uma arquitetura de integração ou referência. O **OMAC** prefere fornecer uma **API** para cada módulo e uma estratégia de montagem e conexão destes módulos (MICHALOSKI, et al., 1996).

O **OMAC** supõe um conjunto de módulos descrito por esta hierarquia de abstração:

- *Foundation Classes* ou classes fundamentais;
- *Framework Components* ou componentes;
- *Core Modules* ou módulos núcleo;
- *Integration or Reference Architecture* ou arquitetura de referência;
- *Application Architecture* ou arquitetura de aplicação;

As classes fundamentais são blocos que podem ser encontrados em múltiplos módulos. Por exemplo, a definição da classe *point* (pontos de acesso) seria encontrada na maioria dos módulos.

Os *frameworks* de componentes são instâncias das classes fundamentais que podem ser integradas dentro dos módulos. Por exemplo: *line*, *helix*, e NURB são componentes do *framework* para a classe *motion plan* (planejamento de movimento).

Os módulos do núcleo têm a funcionalidade que será apresentada a seguir. Uma arquitetura de integração ou de referência descreve a metodologia de configuração à topologia de componentes, sincronismo e dos protocolos de comunicação entre componentes. O resultado da arquitetura de integração é a arquitetura da aplicação. Com a arquitetura da aplicação, os usuários podem desenvolver e executar programas. Por exemplo, alguns candidatos a arquiteturas de referência distribuídas incluem o seguinte: **DCOM** *(Distributed Component Object Model)*, **CORBA** (*Common Object Request Broker Architecture)*, **OSACA** ou **EMC**.

A figura 22 esboça o exemplo do relacionamento entre um sistema de controle da aplicação, seus módulos e componentes, baseados na especificação **API OMAC**.

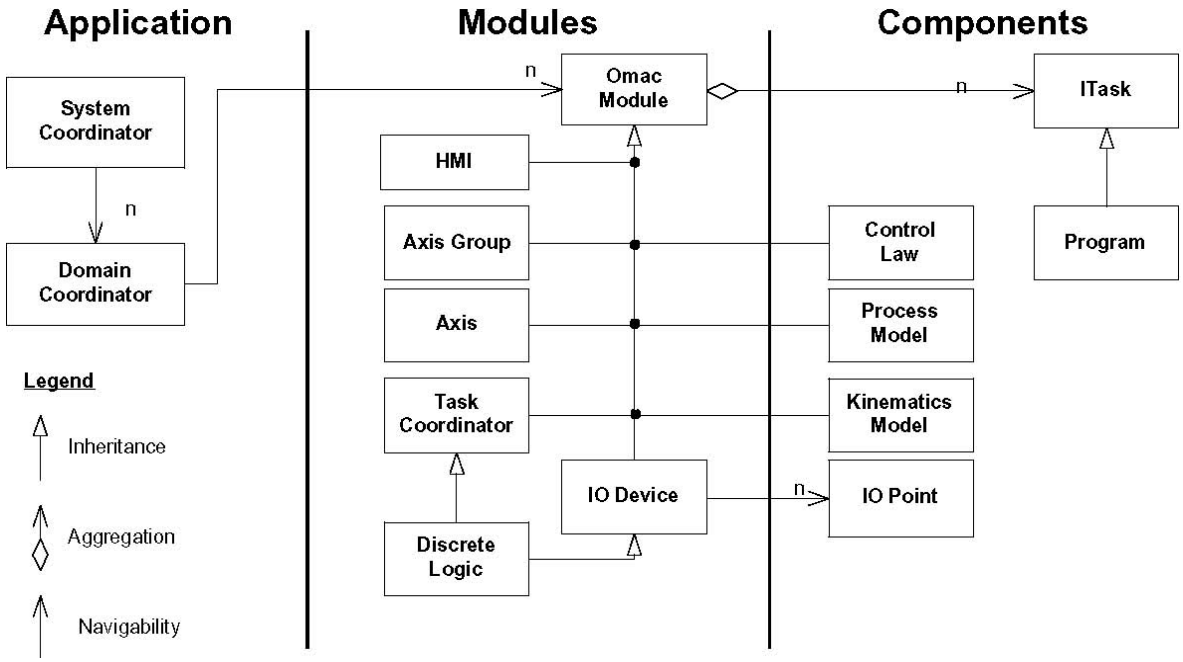


Figura 22: Relações das Aplicações, Módulos e Componentes do **OMAC** (BIRLA, et al., 2001).

Assim, uma aplicação é um sistema de controle computacional construído através da montagem e integração de componentes de *software*. O domínio da aplicação da **API OMAC** é o controle de movimento coordenado de múltiplos eixos, tipicamente encontrado em máquinas **CNC** ou robôs. As aplicações representativas do controlador incluem: cortar, manipular e usinar. Assim, o controlador é complexo para atender uma faixa de aplicações de complexidade variável como, por exemplo, braços múltiplos robóticos a simples eixos usados na indústria de empacotamento ou estação de transferência.

A aplicação pode ser distribuída em uma ou mais plataformas, de modo que um mestre, *System Coordinator*, seja responsável por reunir todos os módulos distribuídos em cada plataforma do controlador, e um *Domain Coordinator* seja responsável por cada plataforma.

A especificação da **API OMAC** define uma série de módulos como foi mostrado na figura 22, com será descrito a seguir, incluindo **Axis**, **Axis Group**, **Task Coordinator**, **Discrete Logic, HMI e IO Device**.

O módulo **Axis** é responsável pelo servo controle do movimento dos eixos, transformando entradas de movimento em referências para os atuadores correspondentes.

O módulo **Axis Group** é responsável por coordenar os movimentos dos eixos individuais, transformando uma especificação de movimento em uma seqüência de referências igualmente espaçadas no tempo para cada eixo coordenado.

O módulo **Task Coordinator** é responsável por seqüenciar as operações e coordenar os vários módulos do sistema baseado nas tarefas programáveis. O **Task Coordinator** pode ser considerado a máquina de estado finito de nível mais elevado no controlador.

O módulo **Input-Output (IO) Device** é responsável por controlar a comunicação entre o dispositivo de *hardware* físico e o *software* de entrada/saída, além de controlar um grupo de pontos de entrada/saída.

O módulo **Human Machine Interfac**e (ou **HMI**) é responsável pela interação humana com o controlador, incluindo apresentação de dados, gerenciamento de comandos, e a monitoração de eventos.

O módulo **Discrete Logic** serve como um mecanismo de propósito geral para representar o *software* de controle de **IO**, permitindo a programação da lógica discreta e de entrada/saída usando tarefas (**Tasks**).

A especificação da **API OMAC** define os **componentes,** como foi mostrado na figura 22, que incluem **Control Law, IO Point**, **Kinematics**, **Process Model** e **Task**.

O componente **Control Law** é responsável pelos cálculos da lei de controle a fim de alcançar as referências especificadas.

O componente **IO Point** é responsável pela leitura de dispositivos de entrada e pela escrita de dispositivos de saída através de uma interface baseada em ler/escrever. Os **IO Point** relacionados logicamente são aglomerados dentro de um módulo **IO Device** ou dentro de um módulo **Discrete Logic**.

O componente do modelo do **Kinematics** é responsável pelas propriedades geométricas do movimento implementando o modelo cinemático do manipulador.

Um componente **Process Model** contem um modelo dinâmico dos dados para que sejam integrados com as funcionalidades de controle.

As **Tasks** são componentes que encapsulam o comportamento funcional de um sistema. A interface da **Task** define um modelo de máquina de estado finito (*Finite State Machine* - **FSM***)* que inclui métodos para começar, parar, reiniciar, parada emergencial, e continuar uma tarefa.

O componente **Program** controla uma série de tarefas (**Task**), com habilidade de reiniciar e gerir.

Os *frameworks* formalizam o ciclo de vida, colaboração e outros aspectos para cooperação de componentes. Exemplos de *frameworks* incluem o **DCOM** e **CORBA**. A utilização de uma arquitetura de controlador baseado em *framework* de componentes é vantajosa tanto para os fabricantes quanto para os consumidores. Com um *framework* padrão para desenvolver componentes, os fornecedores de controlador podem dedicar-se a melhorias especificas da aplicação em vez de reinventar soluções. O grupo **OMAC** tinha o objetivo de especificar um *framework* independente de plataforma, utilizando a linguagem de

especificação **UML** (*Unified Modeling Language*) (**OMG** *Object Management Group*, 2001) (RAPOSO, 2002). Para realizar o desenvolvimento e a validação da **API OMAC** foi escolhida a *framework* **COM**. A **API OMAC** produziu uma documentação de referência da **API** baseando-se em **MIDL** (*Microsoft Interface Definition Language*).

Historicamente, a dificuldade na integração dos componentes tem sido uma barreira à reutilização de código-fonte em grande escala. Com a dificuldade encontrada na integração, cada sistema novo demanda grande esforço, como se ele tivesse sido criado a partir do zero, sendo assumidos todos os riscos envolvidos em uma nova concepção. Para aliviar este problema, o **OMAC** definiu um processo formal de integração a fim estabelecer os relacionamentos e as colaborações necessárias para os componentes nas aplicações de sistema de controle. O **OMAC** definiu uma integração ao nível de componentes de modo que estes compreendessem como colaborar, anunciar sua funcionalidade, onde e sob que *framework* operar. O processo de integração **OMAC** define uma interface **IOmac**, que descreve os serviços básicos para a integração e a distribuição de componentes de uma maneira consistente (PROCTOR, et al., 2000). Usando a notação **UML**, a figura 23 ilustra o modelo de componente alto nível da **API** para a integração.

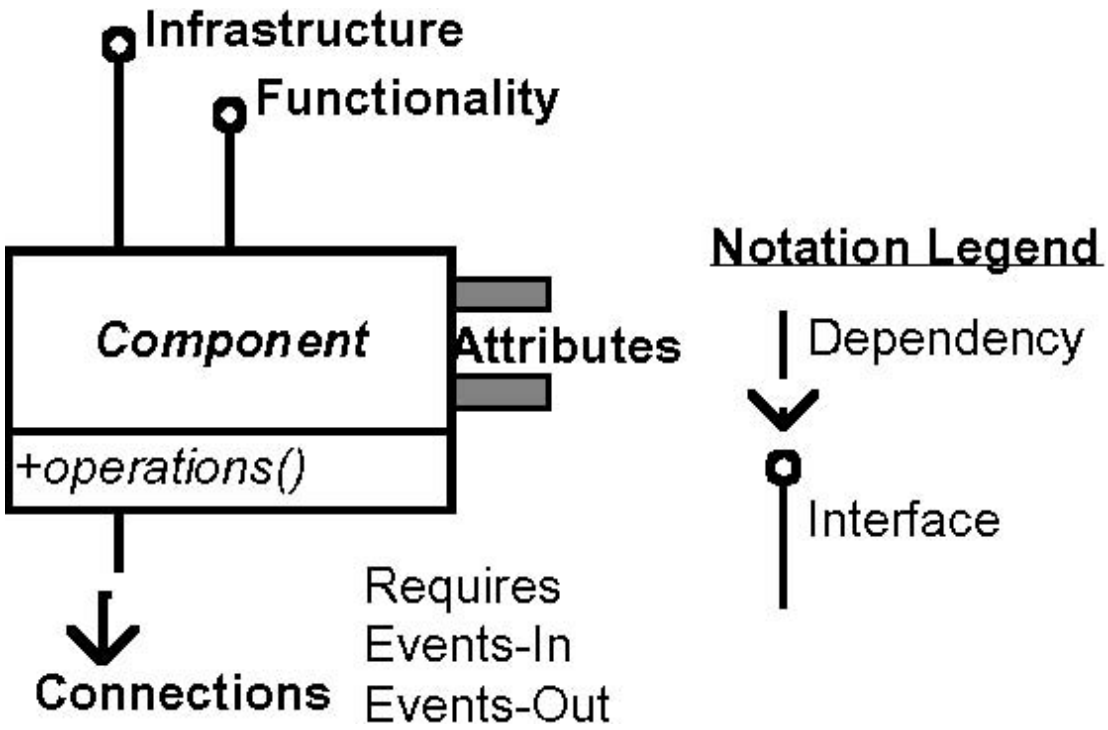


Figura 23: Visão geral do Componente IOMAC (PROCTOR, et al., 2000).

- Interface ***Functionality:*** define a capacidade de controle de um componente. Esta interface inclui métodos para o comportamento, estado e parâmetro de manipulação.
- Interface ***Infrastructure:*** fornece o suporte ao *framework* de aplicação. A interface de infra-estrutura gerencia a identificação e o registro dos componentes, ou seja, é usado para o componente anunciar o que faz, onde está e como opera.
- Interface ***Connection*****:** anuncia as dependências dos módulos tais como as interfaces que requerem e os eventos de entrada e saída que suporta, respondendo desse modo à pergunta sobre o que o módulo necessita.
- ***Attributes:*** parametriza as características de um componente que podem ser customizadas.

O **OMAC** foi pioneiro na busca de aumentar a reusabilidade e customização usando a idéia de componentes plugáveis e permitindo que módulos específicos do controlador possam ser substituídos ou adicionados buscando maximizar a re-configurabilidade dos sistemas(BIRLA, et al., 2001). Por exemplo, o componente *Command Output* dentro do módulo *Axis* e os demais componentes plugáveis, como mostrados na figura 24, podem ser desenvolvidos para comunicar-se com uma rede **SERCOS** (***SE****rial* ***R****eal-time* ***CO****mmunications* ***S****ystem*) ou **CAN** (*Controller Area Network*). O sucesso desta distribuição "*pluggable*" se deve ao fato de não coloca nenhuma carga extra de programação ao usuário final. Os fornecedores de controle ofereceriam um conjunto padrão de componentes previamente conectados de modo que os usuários necessitariam modificar somente se for necessário.

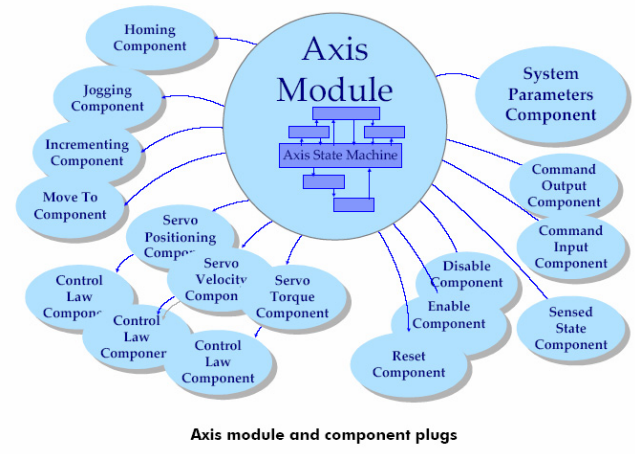


Figura 24: Módulo *Axis* e *components* "plugados" (PROCTOR, et al., 2000).

Um controlador **OMAC** é construído com a combinação de conexões de componentes, figura 25. Alguns módulos contêm pré-conexões de componentes, tais como plugues para *Commanded Input, Commanded Output e Sensed State*. Outras conexões plugáveis de componentes são realizadas questionando ao componente que plugues ele necessita. Os componentes "anunciam" quais outros componentes eles necessitam e um integrador de sistema escolhe um plugue que combina. Então, liga os componentes juntos. Um *I/O point* é um exemplo de um componente ligado que é parte do *I/O Module*.

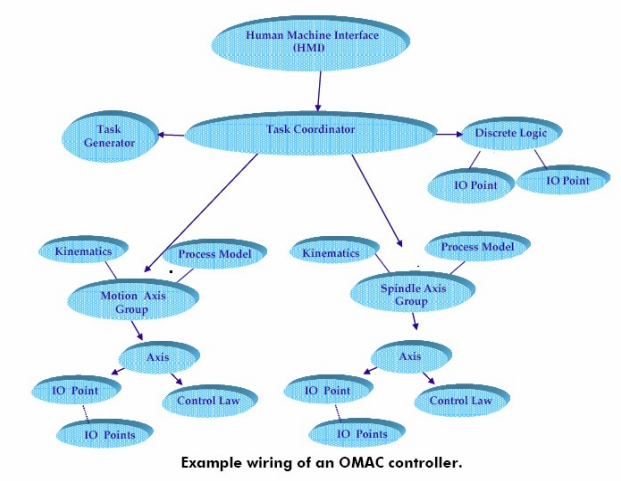


Figura 25: Exemplo de um Controlador **OMAC** (**OMAC** *Users Group*, 2002).

API

Para satisfazer a especificação da arquitetura aberta **OMAC**, um **API** padrão para cada um dos módulos do núcleo foi definida. Conseqüentemente, o objetivo preliminar do projeto foi definir a **API** padrão para os módulos do núcleo pertencentes a arquitetura. Esta seção refinará o conceito de "**API**" e descreverá a metodologia usada para especificação da mesma. A metodologia da especificação da **API** aplica os seguintes princípios:

- Estabelecer a especificação no nível de **API** usando **IDL**;
- Não especificar uma infra-estrutura;
- Usar os paradigmas da Orientação a Objetos;
- Usar modelo o Cliente-Servidor juntamente com máquina de estado;
- Usar Agentes para ocultar a comunicação distribuída;
- Utilizar **FSM** para modelar os dados e os fluxos de controle do sistema;
- Definir Classes fundamentais para fomentar a reutilização;
- Representar por espelhamento o sistema de objetos na **HMI**.

No **OMAC**, o comportamento é um elemento explícito dentro da definição da **API** e segue um modelo definido usando transição de estado. A **API** padrão é útil porque a complexidade de programação é reduzida quando existem diversas alternativas. Esta é uma padronização pequena, mas significativa à arquitetura de controlador aberto.

Existe um problema na seleção da linguagem de especificação usada para representar uma **API**. A linguagem de especificação deve ser flexível o bastante para suportar uma variedade de linguagens de implementação e plataformas. O **OMAC** escolheu a **IDL** como sua linguagem de especificação. **IDL** é uma tecnologia de sintaxe independente utilizada para descrever interfaces. Em **IDL**, as interfaces têm atributos (dados) e assinaturas de operação (métodos). **IDL** suporta a maioria de conceitos da orientação a objeto incluindo a herança. **IDL** pode ser traduzida em linguagens orientadas a objetos tais como C++ e Java, e pode também ser convertida em linguagens que não são orientadas a objetos como o C. As especificações **IDL** são compiladas em arquivos cabeçalhos e programas *stub* para o uso direto por desenvolvedores de aplicação. Por exemplo, para esclarecer o problema de unificação da especificação, considere o mapeamento da **IDL OMAC** em três *testbeds* diferentes para validação. Para o ICON, a **API** padrão em **IDL** tem que ser mapeada em Java. Na universidade Michigan foi usada a ferramenta *Rational Rose* para projetar seu controlador. O *Rose* também aceita arquivos cabeçalho através do processo de engenharia reversa. No *testbed* do **NIST**, o **IDL** é traduzido em arquivos cabeçalhos C++ usados no **EMC** juntamente com sua infra-estrutura. Para as três implementações, a especificação **IDL** pode ser mapeada em todas as linguagens necessárias para suportar as aplicações (KOREN, et al., 1996).

O **OMAC** definiu um processo formal de integração para estabelecer os relacionamentos e as colaborações entre componentes. O **OMAC** contém um módulo **API**, que descreve os serviços básicos para integração e distribuição de componentes de uma maneira consistente. A documentação dos módulos **OMAC** tem uma descrição da distribuição do ciclo de vida dos componentes **OMAC**. Cada módulo no **OMAC** define uma máquina de estado finito que permita que os componentes colaborem de maneira conhecida. Como parte dos serviços fundamentais dos módulos **OMAC**, uma **API** de conexão é definida a fim de permitir aos componentes auxiliar na resolução de dependências de outros.

Aplicações

A seguir serão descritos os principais resultados através de experimentos usando o **OMAC**. A figura 26 mostra um controlador de uma máquina perfuratriz onde a sincronização do eixo *spindle* com o eixo Z é possível de ser realizada para perfurar peças (**OMAC API** *Work Group*, 2001).

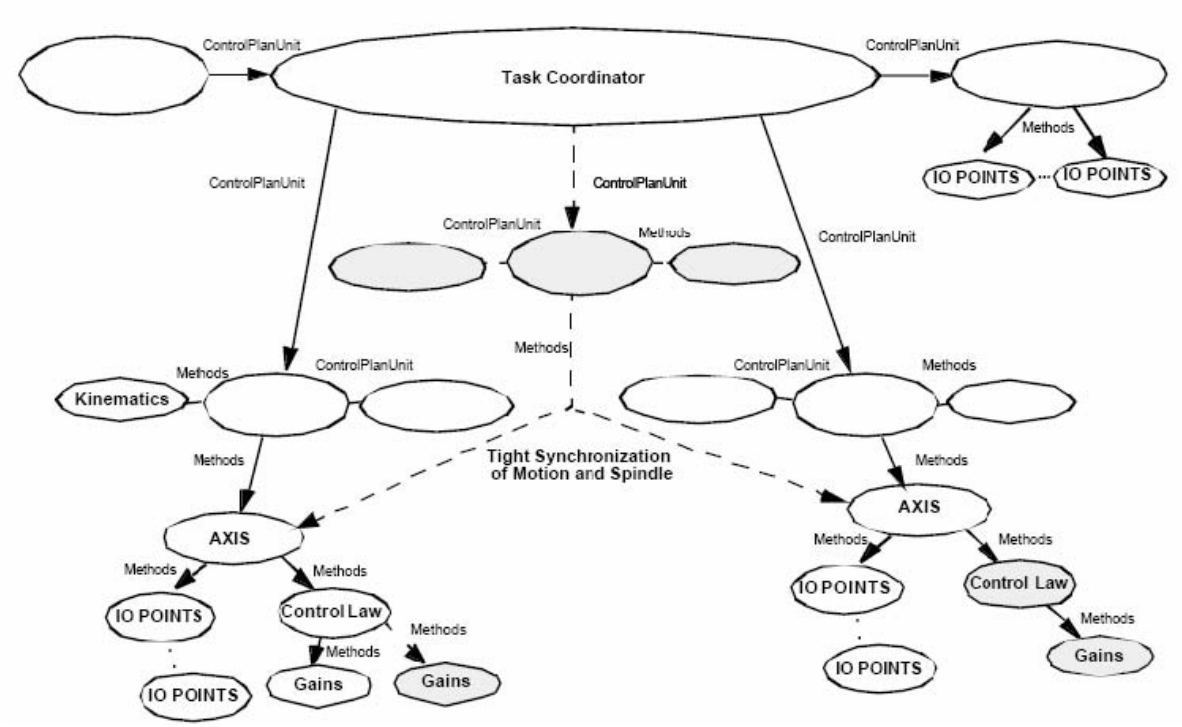


Figura 26: Controlador baseado no **OMAC** de uma máquina perfuratriz (**OMAC API** *Work Group*, 2001).

Resultados foram mostrados a partir da implementação de uma extensão de um controlador de máquina ferramenta usando a linguagem de programação Java e o **OMAC** (WEINERT, 2000).

Além disso, foi implementada a integração de ferramentas usando as potencialidades do **OMAC** tais como: **API** de componentes, geração do código da aplicação e sua verificação para consistência do sistema. A partir deste trabalho podem-se verificar diferentes tipos de teste do controlador bem como suas várias possibilidades de extensões.

2.3.3 – OSEC

O projeto **OSEC** começou no final de 1994, nascido no Japão, e representada pelo Prof. Dr. Ryuji Wada do departamento de engenharia industrial e sistemas da Universidade Setsunan, pelo Dr. Chihiro Sawada da IBM japonesa, juntamente com as empresas Toshiba, Mitsubishi e outras. Sua origem se deu a partir da união de diferentes e importantes fornecedores de sistemas de controle, que juntos formaram um grupo com o objetivo de pesquisar e desenvolver um controlador aberto japonês. Em 1996, o **FAOP** (*Japan Factory Automation Open Systems Promotion Group*) foi fundado em paralelo ao **OSEC** com o mesmo objetivo, mas aplicado a automação de fábrica. Assim, as iniciativas se uniram na tentativa de unificar seus trabalhos e passaram a ser reconhecidas sob a sigla **JOP** (*Japan Open Promotion*). O objetivo do **OSEC** foi construir um modelo de arquitetura, definir funções de controle e interfaces neste modelo, estabelecer algumas regras de inter-relações entre elas e propor uma nova linguagem de descrição para comando numérico denominado **OSEL** (*Open System Enviroment Language)*. Além disso, a validação deste desenvolvimento foi realizada através da criação de protótipos (SAWADA, 1997).

Definições

O **OSEC** foi concebido com base nas necessidades e especificações dos usuários das máquinas ferramentas, como apresentado na tabela 4 (FUJITA, et al., 1996).

Tabela 4: Necessidades **OAC** para Máquina Ferramenta.

Itens	Necessidades
Reconfiguração	Funções podem ser incluídas ou retiradas
Escalabilidade	*Hardware* e *software* necessitam de independência funcional e reconfigurabilidade.
Linguagem **NC**	Concepção de uma nova linguagem de programação para **NC** (*Numerical Control*).
Novas funções	Criação de novas funcionalidades tais como interpolação de curvas e condições de corte.
Inteligência	Compatibilidade entre o programa e o tratamento da ferramenta utilizando uma única linguagem de programação
Aberto	Capacidade de integração de diferentes ferramentais, tais como: CAD/CAM, supervisório, redes, etc.

Infra-estrutura

A infra-estrutura utilizada na abordagem **OSEC** é focalizada somente na plataforma **PC** e Windows. No **OSEC** não é possível fazer o controle distribuído de seus componentes (NACSA, 2001).

Arquitetura de Referência

No **OSEC** existe a definição de um modelo de referência composto de 7 camadas baseado nas camadas do **OSI**(SAWADA, 1997). Assim, nesta arquitetura é dada grande importância para as interfaces de programação entre as camadas, denominada de *Message Coordination Field*, como apresentado na figura 27. Na mesma figura, são citados os módulos que compõe a arquitetura com suas respectivas funcionalidades: *Resource Management* (Gerenciamento de recursos), *Motion Generation* (Geração de movimento), *Machine Control* (Controle da máquina) e *Device Control* (Controle de dispositivos). As implementações das funções pertencentes ao **OSEC** utilizam a linguagem de programação C.

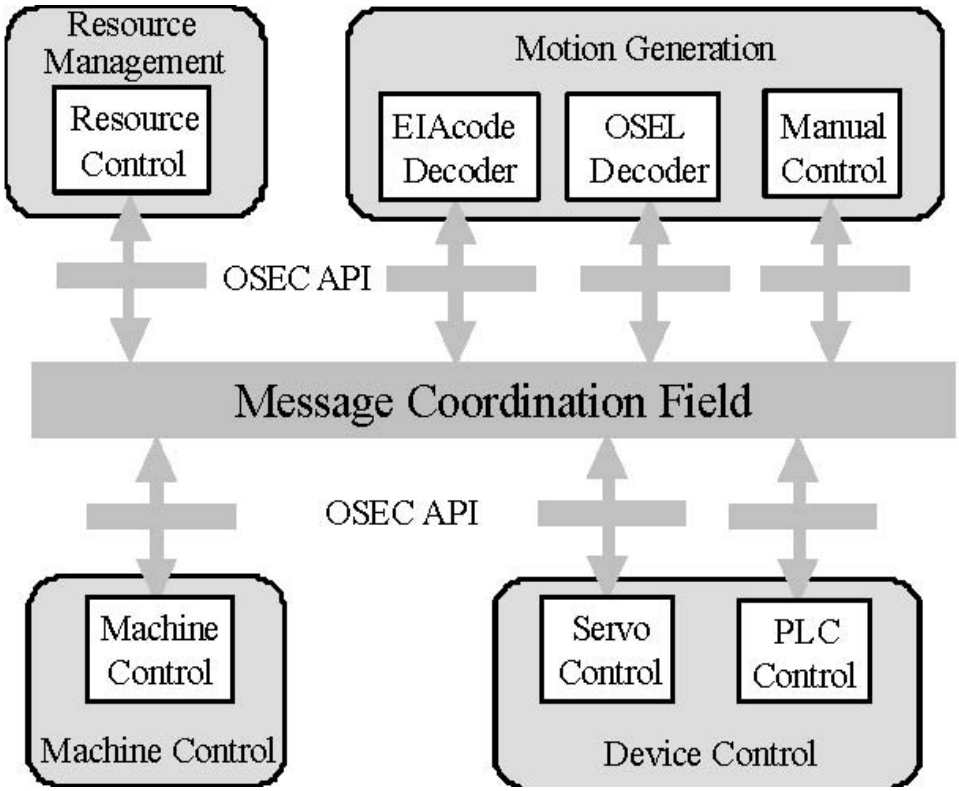


Figura 27: Arquitetura de Referência **OSEC** (SAWADA, 1997).

API

Infelizmente, não houve harmonia na junção das **API´**s do **OSEC** e **FAOP**. Assim, o **JOP** publicou uma nova **API** entre o núcleo **NC** e a interface **HMI** que é chamada PAPI e utiliza a linguagem de programação C.

Aplicações

A seguir serão descritos os resultados experimentais utilizando o **OSEC**. Módulos de *software* correspondentes às funções baseadas no **OSEC** foram desenvolvidos e verificados nos sistemas protótipos.

Na figura 28, é mostrada a comparação entre os três sistemas (SAWADA, 1997).

Figura 28: Estação de *Machining* (SAWADA, 1997).

No modelo arquitetura **CNC**, figura 29, os blocos grandes descrevendo as estações A, B e C mostram a parte **CNC**. E as linhas horizontais cinzentas indicam o grupo de funções usando um protocolo comum entre cada uma das estações. O bloco acinzentado significa que as funções correspondentes são implementadas por placas **NC** adicionais ao sistema.

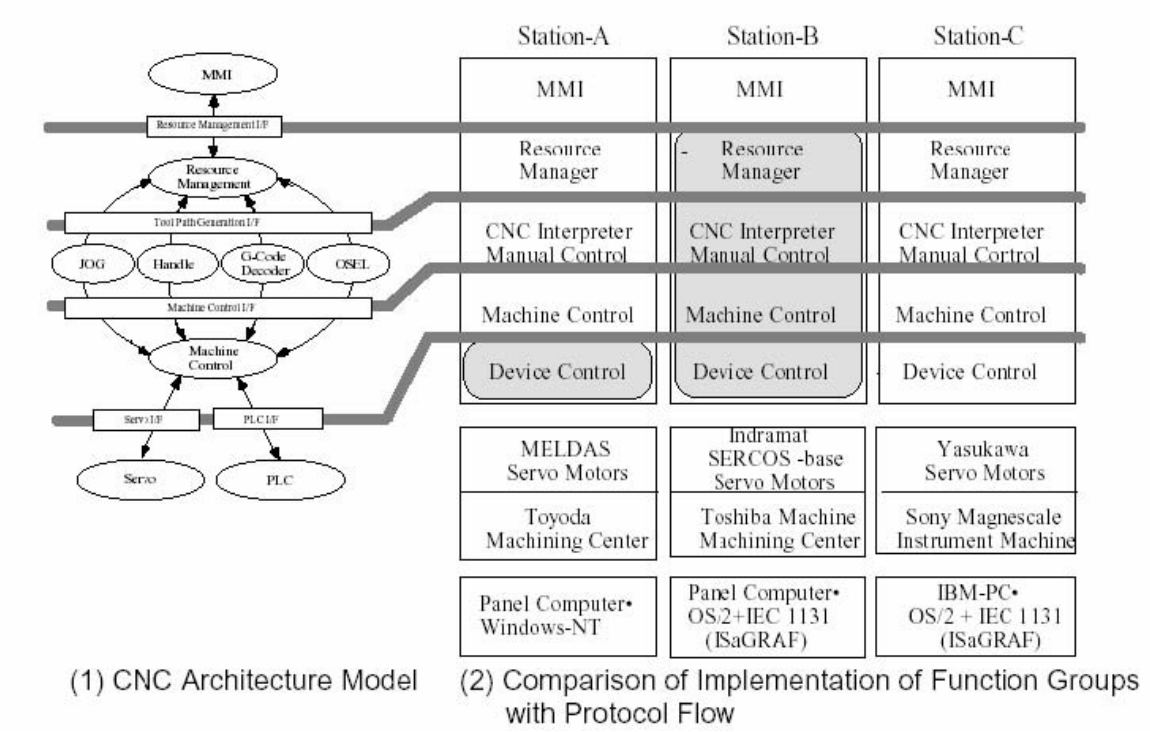


Figura 29: Modelo Arquitetural **CNC** e Comparação de Implementações Funcionais (SAWADA, 1997).

O modelo arquitetura da máquina é subdividido em:

A estação A implementa o grupo de funções de controle de dispositivo com a placa **NC** mas também implementa outros blocos de funções com controle de *software* baseado em **PC**.

A estação B é um **CNC** baseado em **PC** e usa um *software* no **PLC** (*Programmable Logic Controller*) para monitoração. Além disso, ela tem *servo-drives* com especificação **SERCOS**.

A estação C é um *software* **CNC** e usa o *software* no **PLC** para controle de seqüência.

2.4 - Pontos fortes e fracos dos OAC`s

A partir dos estudos realizados das arquiteturas dos controladores abertos **OSACA**, **OMAC** e **OSEC** bem como do estudo dos trabalhos de Porto (2002), Kremer (2001), Yonglin (2005) e Nacsa (2001) pôde-se elaborar uma análise comparativa que norteie a concepção de sistemas abertos para automação em relação a Infra-estrutura, Arquitetura de Referência e **API**. A seguir, serão apresentados os resultados conclusivos e as comparações das abordagens, comentários e conclusões a respeito dos respectivos trabalhos.

Porto (2002) publicou um artigo que sintetiza qual o principal enfoque tratado pelas abordagens de controladores abertos. Neste estudo é relatado que a arquitetura **OSACA** dá maior ênfase à área de *software*. E assim, define conceitos, infra-estrutura, arquitetura de referência e **API**. O **OMAC** tem abrangência nas aplicações industriais, preocupando-se em definir uma visão geral da arquitetura de controlador aberto a partir de diferentes usuários, tal que compartilhem experiências de maneira padronizada. Por fim, o **OSEC**, que vê o controlador como um recurso inserido em um sistema de automação utilizado nas plantas industriais.

Kremer (2001) apresentou um estudo feito com o objetivo de avaliar especificamente o **OSACA**, **OMAC** e **OSEC**. Uma comparação inicial do **OSACA** e do **OMAC** indica que há muitas características comuns, por exemplo, a metodologia orientada a objetos em ambas as aproximações, mas não examinou os detalhes das especificações. Além disso, neste estudo ficou clara a necessidade das abordagens tratarem as características tempo-real das aplicações nas quais são utilizadas arquiteturas abertas. Outra observação a partir deste estudo foi a necessidade de conceber mais aplicações utilizando as abordagens citadas, para que se possa melhor avaliá-las.

Recentemente, Yonglin (2005), propôs uma forma inédita de avaliação de controladores de arquitetura aberta. Os principais problemas desta área são a aplicação e a avaliação para aceitação destes tipos de sistemas. Assim, esta metodologia poderá guiar tanto a escolha de qual abordagem utilizar, como também como projetar uma arquitetura de controlador aberta em função das características levantadas a partir da aplicação em questão. Além disso, em seus estudos comparativos, Yonglin considerou o **OMAC** a arquitetura mais aberta em relação ao **OSACA** e o **OSEC**.

O trabalho de Nacsa (2001) analisou o **OSACA**, **OMAC** e **OSEC** do ponto de vista da implementação. Entretanto, o seu objetivo era descobrir quais os controladores abertos existentes, compará-los e unificá-los tal que servissem de padrão para implementação de aplicações. Em síntese, não é possível unificar **OSACA**, **OMAC** e **OSEC**. Deve-se escolher somente uma das iniciativas que satisfaça as necessidades da aplicação.

As pesquisas mostram que o **OMAC** foi o mais indicado para suporte a aplicações de natureza aberta. Porém, sua **API** é complicada em virtude de sua abrangência de funcionalidades. A seguir serão mostrados as tabelas e comentários extraídos a partir do trabalho de Nacsa (2001).

2.4.1 - Comparação de Infra-estrutura

OMAC é a única abordagem aberta do ponto de vista de infra-estrutura, como mostra a tabela 5.

Tabela 5: Características de *hardware* e *software*.

	OSACA	**OSEC**	**OMAC**
Plataforma	Específica	Específica	Geral
PC - Intel	Também outras	Somente	Geral
Windows	Também outras	Somente	Geral
Linguagem de Programação	C/C++	C/C++	C/C++/Java
Distribuída	Sim	Não	Sim
DCOM, **Corba**, etc.	Não	Não	Sim
Modernidade	Desatualizado	Desatualizado	Atualizado

No caso do **OSACA**, existe a necessidade de uma plataforma específica. Se houver necessidade de portabilidade de plataforma, isto inviabiliza sua utilização. Por fim, o **OSEC**, que trabalha somente em **PC** utilizando o sistema operacional Windows.

2.4.2 - Comparação de Arquiteturas

Apesar do mesmo propósito tratado nas diferentes abordagens, a abstração realizada em cada uma delas é colocada em módulos diferentes. Conseqüentemente, torna-se inviável a substituição de módulos entre abordagens, principalmente, em virtude das diferenças de especificação. A tabela 6 apresenta os resultados a partir da análise feita nas arquiteturas de referência de cada um dos modelos e, posteriormente, é apresentada a conclusão a respeito.

Tabela 6: Especificação da Arquitetura de Referência.

	OSACA	**OSEC**	**OMAC**
Arquitetura de Referência	Existe	Existe	Modular
Numero de Módulos	9	7	14
Módulos Especificados	4 de 9	Todos	Todos
Faixa de aplicações potenciais	Aceitável	Aceitável	Largamente
Descrição interna do módulo	Indefinida	Indefinida	Especificada com **FSM**

O **OSACA** e o **OSEC** não apresentam o mesmo nível de detalhamento, tal que uma representação abstrata de um módulo possa ser realmente alternada entre estas arquiteturas.

2.4.3 - Comparação entre API`s

As características da **API** são mostradas na tabela 7.

Tabela 7: Propriedades das **API**`s.

	OSACA	**OSEC**	**OMAC**
Lógica	Orientada a Objetos	Funções	Componentes
Definição	C++	C	**IDL**
Granularidade	Ideal	Grande	Pequena
Tamanho	~100 itens (variáveis, eventos, etc.)	~ 150 funções	Milhares de funções e muitos tipos
Vantagem	Tamanho da granularidade	Fácil para entender	Completa
Desvantagem	Variáveis ao invés de funções	Não é orientada a objetos	Complexa

Infelizmente, as idéias fundamentais e os níveis de abstração são diferentes. O detalhamento da proposta **OMAC** é maior em relação as abordagens **OSACA** e **OSEC**. Classificando-se os **OAC**`s a partir dos estudos realizados e na observação das unanimidades dos trabalhos apresentados anteriormente, pode-se dizer que o **OAC** mais aberto é o **OMAC**, seguido pelo **OSACA** e, por fim, pelo **OSEC**. A seguir, serão confrontados os pontos fortes e fracos dos **OAC**`s de maior relevância, **OSACA** e **OMAC**, apresentados nas tabelas 8 e 9, com destaque para definições, infra-estrutura, arquitetura de referência, **API**, e aplicações.

Tabela 8: Pontos fortes e fracos do **OSACA**.

	Pontos Fortes	**Pontos Fracos**
Definições	Descrição baseada em Padrões	Atualmente incompletas
Infra-estrutura	Linguagem de programação **ANSI** C e C++	Específica (**PC** Intel, Windows 95 e VxWorks) Barramento **VME**
Arquitetura referência	Número de módulos	Descrição incompleta
API	Orientação a objeto **ANSI** C++	Especificação incompleta Uso de variáveis ao invés de funções
Aplicações	Domínio máquina ferramenta	Domínio robôs

Os principais benefícios do **OSACA** são suas definições e descrições do que seria um controlador aberto baseado em padrões internacionais; a idéia de um sistema de configuração de módulos e objetos que diga qual a topologia necessária para suportar a aplicação; uso do padrão **POSIX** (**IEEE** and *The Open Group*, 2004) para implementação tempo real; e, por fim, sua consistência de resultados das aplicações implementadas.

Tabela 9: Pontos fortes e fracos do **OMAC**.

	Pontos Fortes	**Pontos Fracos**
Definições	Atualizadas	Padronização incompleta
Infra-estrutura	Todas	Nem todas foram testadas
Arquitetura referência	Descrição dos módulos Completa Especificação dos módulos usando **FSM**	Número de módulos
API	Componentes **IDL** Completa	Quantidade de funções Complicada
Aplicações	Domínio máquina ferramenta	Domínio robôs

Os principais benefícios do **OMAC** são a concepção com enfoque no modelo baseado no conceito de componentes; os módulos especificados de maneira formal através de **FSM**; oferece metodologia para integração de componentes; o sistema de comunicação é externo à arquitetura de referência, fornecendo o suporte de comunicação necessário aos componentes do sistema; e, por fim, preocupação com o determinismo de *hardware* e *software* das tarefas tempo real do sistema.

2.5 - *GAP* dos OAC`s

A seguir será relacionado de maneira sucinta o que falta ser feito nos **OAC`s** do ponto de vista deste trabalho e, posteriormente, respectivos comentários:

- Definição das características de **OAC** focados em robôs;
- Definição de uma Arquitetura de Referência ou Modelo para robôs;
- Concepção com enfoque ao modelo independente de infra-estrutura;
- Criação de **API** a partir do modelo;
- Especificação de infra-estrutura baseada em **PC**;
- Aplicação de **OAC`s** no domínio da robótica.

A conclusão resultante do estudo dos **OAC**`s é que existe a necessidade de uniformidade internacional através de um padrão que atenda as exigências do que seria um controlador aberto para robô. Entretanto, não é possível fazer uma convergência entre as três abordagens relatadas em uma única solução. O motivo fundamental é a não uniformidade de conceitos, pois não existe consenso do que seria um sistema aberto para robôs. Daí a importância de se adotar as características que formalizam um sistema aberto estritamente para robôs. O **OSACA**, por exemplo, trabalha com cinco definições que caracterizam um **OAC**, e o **OMAC** define seis características do que é um controlador aberto. Entretanto, somente uma delas é comum a ambas as abordagens de sistemas abertos para automação e sendo diferentemente definidas.

Definir e criar uma arquitetura de referência ou modelo para robôs baseados em normas internacionais, esclarecendo quais componentes fundamentalmente fazem parte da estrutura, seu comportamento, relações e organizações. Não existe, por exemplo, uma definição clara do que é uma arquitetura de referência para robôs abertos. Isto irá simplificar a criação de um **OAC** para robôs, principalmente usando conceitos baseados em normas internacionalmente aceitas pelas comunidades acadêmicas e industriais.

O uso de especificação formal deve ser empregado na concepção do **OAC**. Assim, podem-se evitar ambigüidades e minimizar erros utilizando ferramentas de verificação. Por exemplo, eliminação de *deadlock* com a modelagem das tarefas do **OMAC** usando **FSM**.

O ciclo de concepção de um **OAC** é formado pelas etapas de definição, arquitetura de referência, **API** e infra-estrutura. Não se deve buscar a flexibilidade desejada olhando-se para o resultado final do processo de criação do **OAC**, no caso, a aplicação, mas sim, para as etapas iniciais, que são os componentes do modelo. À medida que se avança em cada etapa, sendo cada uma delas conseqüência da outra, se na etapa anterior não for garantida a flexibilidade aos componentes, ficará cada vez mais difícil obtê-la na aplicação. Por exemplo, o **OSACA** enfoca suas características de controlador aberto em relação à plataforma de aplicação. Já o **OMAC** aborda os componentes de sua arquitetura de referência, daí o motivo de obter melhores resultados.

Seria desejável definir e criar uma **API** para robôs baseando-se nos conceitos de orientação a objetos, bem como na implementação desta utilizando os padrões **ANSI** C++ e **ANSI** C. Por exemplo, tanto o **OSACA** como o **OMAC** usam normas. O problema é que suas **API`s** são incompletas em virtude da abrangência de domínio dos **OAC**.

É fato que o uso de **OAC** baseado em **PC** traz enorme flexibilidade. Então, deve-se buscar maximizar sua utilização bem como todo ferramental de suporte de desenvolvimento e implementação de sistemas **OAC**`s. A definição ou especificação do que é uma infra-estrutura básica para robôs também é necessária, principalmente sua flexibilidade em relação tanto aos aspectos de *hardware* como de *software*. Por exemplo, no caso do **OSACA**, a infra-estrutura é tão específica que o torna limitado em sua aplicabilidade. A Interface gráfica é suportada pelo Windows 95 com placa **PC** Intel sob barramento **PCI** e o Controlador de Movimento utiliza VxWorks com placa M60040 Motorola sob barramento **VME**. Já o **OMAC** não tem nenhuma especificação de infra-estrutura básica, o que, o torna inconsistente. Portanto, devem-se utilizar sistemas operacionais comuns e padronizados ao invés de criar controladores com sistemas operacionais específicos.

Além disso, devem-se utilizar linguagens de programação de alto nível como C e C++ para controlar as tarefas de um robô ao invés de criar uma linguagem de programação específica e restrita a robôs. Seguindo esta idéia, deve-se usar suporte tempo real e/ou distribuído que existe à disposição sem a necessidade de incorporá-los aos **OAC**`s ou até mesmo criá-los com este propósito específico. O **OSACA**, ao contrário, tem suporte de comunicação próprio, módulos internos usam modelo cliente/servidor e em relação ao meio externo, a comunicação é baseada no modelo **OSI**.

A partir das observações do estudo realizado, existe uma abrangência no sentido de automação. As três concepções são realizadas no domínio de máquinas ferramenta, **PLC**`s e robôs. Assim, torna-se difícil definir, especificar e aplicar de maneira geral as respectivas arquiteturas a fim de avaliá-las em sua totalidade. Aplicar **OAC**`s no domínio da robótica é uma necessidade a fim de validar a consistência dos resultados tal que se possa difundir e estimular à utilização de robôs abertos em diferentes áreas de aplicação. Por exemplo, historicamente, a dificuldade na integração dos componentes tem sido uma barreira à reutilização de código-fonte em grande escala. Com a dificuldade encontrada na integração, cada sistema novo transforma-se freqüentemente em um novo grande esforço, com se não houvesse nada, assumindo todos os riscos envolvidos em uma nova concepção. Porém, ambas arquiteturas demonstram seus resultados a partir de suas respectivas aplicações apesar das limitações em relação à robótica. Além disso, de posse do modelo aberto, pode-se avaliar se a aplicação de um robô é factível. Por fim, deve-se demonstrar a viabilidade e validade da proposta através de aplicações e assim comprovar de fato a flexibilidade dos **OAC`s** em relação a: **HMI**; estratégia de controle; geração de trajetória; diferentes modelagens cinemáticas; técnicas de calibração; tipos de manipuladores; robótica com enfoque tempo real; uso de diferentes componentes que fazem parte de um robô.

Capítulo 3

Um Modelo Aberto para Robôs Manipuladores de Serviço

O estudo do estado da arte em arquitetura de controladores permitiu evidenciar as metas a serem ultrapassadas para alcançar os objetivos da tese. Assim, neste capítulo, é descrita a sistemática de como foi concebido o modelo do sistema, a implementação computacional e a aplicação do modelo no decorrer da tese.

Em termos gerais, este trabalho pretende definir um modelo aberto para robôs que atenda às especificações rígidas de sistema aberto, respeite padrões internacionais de *software* e *hardware*, flexibilize as aplicações da área de robótica e permita, dessa forma, disseminar a utilização de robôs em áreas até então pouco exploradas.

Em relação à abertura a ser alcançada pelo modelo, ela pretende ser a mais completa. Conforme abordado anteriormente no capítulo dois, existem três níveis de abertura de um sistema: o nível que abre somente a interface gráfica com o usuário, que é o mais básico; o nível que abre também a estratégia de controle e, por fim; o mais completo, que abre os componentes, tais com: ferramentas, controlador, manipulador, entre outros. Realizou-se a implementação do modelo definido com o objetivo de validá-lo, obter consistência e realimentar as pesquisas com os resultados obtidos. A implementação pode ser a modificação do processo de soldagem, a troca de manipuladores, integração de diferentes ferramentas, etc. Além disso, no escopo deste trabalho, estão às aplicações das implementações escolhidas no âmbito da robótica, que foi numa área atípica como a do setor energético.

Atualmente, as iniciativas **OSACA** e **OMAC** ainda não chegaram a um robô aberto, como mostra esta tese. O **OSACA** desde 1992 até a presente data persegue este objetivo inserido agora no projeto denominado OCEAN, lançado em 2001. As aplicações do **OSACA** demonstram somente a existência de uma arquitetura de controlador aberta usado em máquinas ferramentas. No caso do projeto **OMAC** que tem o mesmo objetivo desde seu surgimento em 1994 até os dias atuais, as aplicações são menos expressivas quando comparadas com o **OSACA**. Essas aplicações são direcionadas à área de máquinas ferramenta e **PLC**`s, o que dificulta a conclusão com respeito à robótica. Deste modo, ao final deste trabalho é apresentado um modelo aberto para robôs, implementado e aplicado para demonstração do seu ineditismo.

3.1 – Modelagem do Sistema

No desenvolvimento e implementação de um **OAC** foram identificadas as etapas de definição, arquitetura de referência, **API**, infra-estrutura e aplicações. Nesta tese é feita uma nova sistemática que compreende a modelagem do sistema, implementação computacional, e, por fim, aplicações do modelo buscando automatizar o processo de concepção de sistemas de controle aberto.

Um modelo é uma representação da realidade, por isso não existe uma razão fundamental para construir um modelo computacional. Entretanto, é criado um modelo para que se possa entender o sistema a ser desenvolvido e implementado. Segundo Booch, Jacobson e Rumbaugh (1999), a importância do processo de criação de um modelo está relacionada aos seguintes fatores:

- Modelos ajudam a visualizar como é ou será o sistema a ser concebido;
- Modelos permitem especificar a estrutura e o comportamento do sistema;
- Modelos são uma referência que guia a construção do sistema;
- Modelos servem para documentar as decisões no decorrer do processo de desenvolvimento e implementação do sistema.

Os princípios da modelagem são (BOOCH, 1999):

- A escolha de quais modelos criar tem uma profunda influência sobre como um problema é abordado e como a solução é formada;
- Cada modelo pode ser expresso em diferentes níveis de precisão;
- Os melhores modelos são conectados com a realidade;
- Um único modelo não é suficiente.

Nesta tese, os requisitos da aplicação são enquadrados nas características de sistemas abertos para robôs contemplados na modelagem do sistema. Posteriormente, a modelagem do sistema é equivalente a arquitetura de referência, consistindo dos aspectos estruturais acrescentados dos aspectos comportamentais, representados de maneira formal, utilizando uma linguagem de especificação padrão unificada para representação de modelos denominados de diagramas **UML,** desta maneira garantindo a independência de infra-estrutura entre as etapas de desenvolvimento e implementação de um sistema.

A arquitetura de referência do **OSACA** possui um número de módulos ideal, suficiente para modelar um sistema. Com relação ao **OMAC**, sua arquitetura de referência é completa, pois todos os seus módulos são descritos formalmente utilizando máquina de estados. Estas peculiaridades trazem enormes contribuições aos **OAC**`s. Porém, ambas as abordagens, **OSACA** e **OMAC**, estão baseados em definições não consensuais e não especializadas para robótica. Em virtude de não se ter um sistema na área de robótica, um modelo aberto para robôs denominado **ORM** (*Open Robot Model*), fundamentado em padrões de sistemas abertos e completamente especificado, é introduzido nesta tese.

De maneira geral, tanto o **OSACA** quanto **OMAC** buscam resolver a problemática dos sistemas abertos para máquinas ferramenta, **PLC**´s e robôs. Ao se representar um **OAC**, tem-se um modelo maior e mais complexo em relação a modelagem somente no domínio de robôs. Então, é feita a redução no escopo do modelo à robótica, com a finalidade de simplificar tanto o entendimento e a utilização de um modelo aberto para robôs. Desta maneira, tem-se um modelo concluído, implementado e, por fim, aplicado ao domínio de robôs.

O **OSACA** foi pioneiro em levantar as necessidades associadas à arquitetura de controladores abertos e conseguiu, a partir delas, definir suas características baseando-se nos conceitos padronizados de sistema abertos. O **OMAC** definiu novas características para controladores abertos, mas não utilizou para isso qualquer padrão aceito internacionalmente. Sendo assim, é feita a definição de cada uma dessas características buscando mapeá-las nos padrões **IEEE** de sistemas aberto.

Conceber um sistema aberto para robôs fazendo uma previsão de seus componentes, como é feito no **OMAC**, é uma tarefa de grandes proporções, haja vista a infinidade de aplicações para que um robô possa vir a ser utilizado. Isto pode levar ao comprometimento do resultado. Por outro lado, criar um sistema aberto para robôs com enfoque baseado na aplicação, como no **OSACA**, aumenta as chances de chegar a um resultado, porém, pode torná-lo limitado, atendendo somente a uma situação específica. Portanto, é feita uma abordagem *top/down* do sistema ao invés de *button/up* como são feitas tanto no **OSACA** como no **OMAC**. E a construção do sistema é baseada no modelo computacional abstraindo-se da aplicação e de seus componentes garantindo a independência da infra-estrutura utilizada para suportar o sistema.

O **OMAC** utiliza uma linguagem de especificação formal para a representação de seus componentes em sua arquitetura de referência e utiliza máquina de estados para expressar o comportamento de seus componentes. Neste trabalho, utilizar-se **UML** na representação do modelo, extrapolando a utilização de especificação formal em componentes feitas pelo **OMAC**. Desta maneira, através da engenharia direta e reversa, pode-se gerar automaticamente a partir do modelo do sistema sua **API** correspondente e vice-versa. E desta forma, elimina-se inconsistência entre o modelo computacional e implementação do sistema.

William Ford (1994) vislumbrou o que seria um **OAC** para robô citando quais as características que um robô aberto deveria possuir, entretanto, não as definiu. Sendo assim, a seguir serão citadas estas características e definidas, neste trabalho, cada uma delas a partir de padrões internacionalmente aceitos tal que exista uma uniformidade no que é um controlador de arquitetura aberta para robô.

1. **Portabilidade** *(Portability)*: É a facilidade que um sistema ou componente pode ter de ser transferido de um ambiente de *hardware* ou *software* para outros (**IEEE**, 1990) (**OSACA**, 2007).

2. **Escalabilidade** (*Scalability*): É a habilidade para prover funcionalidades, incremental ou decremental, a uma série gradual de plataformas de aplicação que diferem em velocidade e capacidade (**IEEE**, 1995) (**OSACA**, 2007).

3. **Extensibilidade** (*Extendability* or *expandability* or *extensibility*): É a facilidade que um sistema ou componente de poder ser modificado para aumentar sua capacidade de armazenamento ou funcional (**IEEE**, 1990) (**OSACA**, 2007).

4. **Interoperabilidade** (*Interoperability*): É a habilidade de dois ou mais sistemas ou componentes para trocar informações e usá-las (**IEEE**, 1990) (**OSACA**, 2007).

5. **Intercambialidade** (*Interchangeability* or *exchangeable*): É a capacidade de um módulo ser substituído por outro módulo com comparáveis funcionalidades (**OSACA**, 2007).

6. **Reusabilidade** (*Reusability*): É o grau que um módulo de *software* ou produto pode ser usado em outros programas ou sistemas computacionais (**IEEE**, 1990) (**OMAC**, 2007).

7. **Reconfigurabilidade** (*Reconfigurability*): É a capacidade de um módulo ou componente ser customizável tal que possa ser reutilizável (**OMAC**, 2007).

8. **Confiabilidade** (*Reliability* or *dependability*): É a habilidade do sistema ou componente manter sua integridade em caso de anormalidades (**IEEE**, 1990) (**OMAC**, 2007).

9. **Flexibilidade** ou **adaptabilidade** (*Flexibility* or *Adaptability*): É a facilidade que um sistema ou componente pode ser modificado para ser utilizado em aplicações ou ambientes diferentes do qual foi concebido (**IEEE**, 1990).

A partir das características identificadas e definidas rigorosamente, não se tem mais imprecisão do que é um **OAC** pra robô. De tal modo, a especificação do modelo é feita utilizando-se a **UML** que, por fornecer a visualização gráfica do sistema, facilita em muito o entendimento do mesmo, já que cada visualização é apresentada em forma de diagramas a partir de pontos de vista estrutural e comportamental. Adicionalmente, a **UML** define um padrão de notação, viabilizando que pessoas de áreas distintas possam interagir sem equívocos ou ambigüidades em relação à notação usada para entendimento do modelo. E outro benefício que se encaixa às aplicações na área de robótica, é que **UML** permite englobar aspectos de *software* e *hardware* num mesmo modelo.

Na figura 30 é apresentado o diagrama de caso de uso do modelo. Na **UML**, os diagramas de caso de uso demonstram o comportamento de como o sistema interage como o meio externo. Este diagrama apresenta a idéia fundamental do modelo **ORM** aplicado no âmbito dos projetos Roboturb e Robofurnas. Nele, o sistema é modelado por um robô (***Robot***); ferramentas que o robô irá utilizar (***Tool***); gerador de trajetória (***Trajectory Generation***), tal qual o robô irá executar; e a interface com o usuário (***User Interface – UI***), estabelecendo as configurações e o controle dos componentes necessárias ao robô tal que este realize sua tarefa de acordo com sua programação de trajetória especificada.

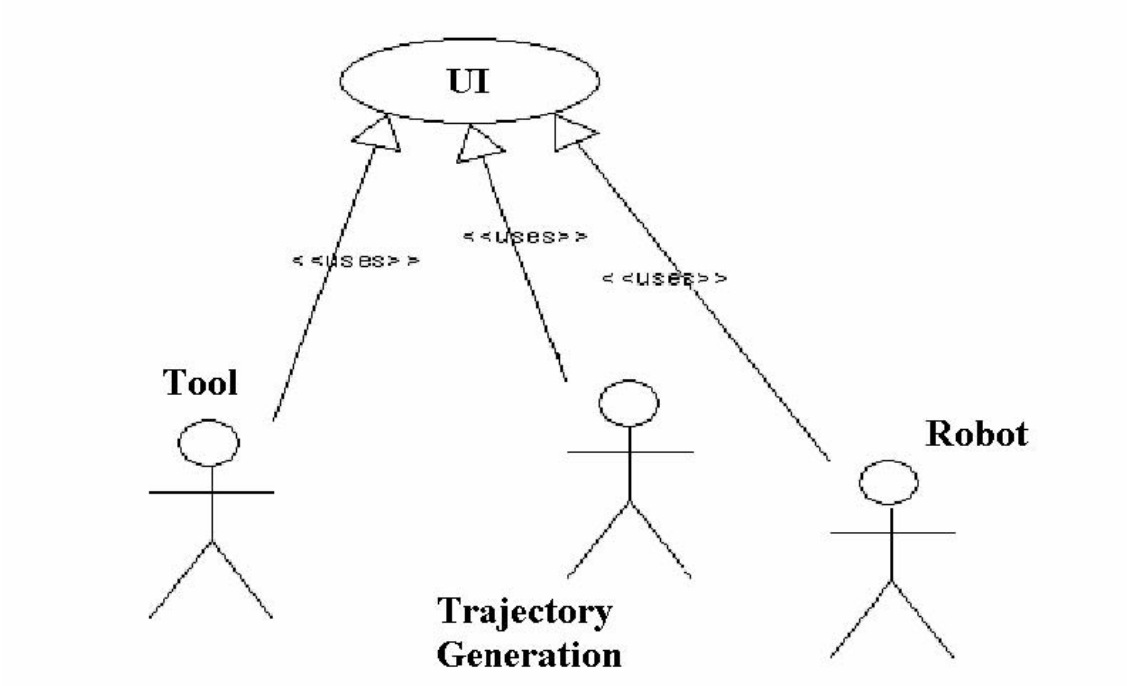


Figura 30: Diagrama de Caso de Uso do **ORM**.

O diagrama de distribuição mostra a estrutura física do sistema, ou seja, de que forma os elementos de *hardware* e *software* estão distribuídos e como interagem entre si. A figura 31 mostra o diagrama de distribuição do modelo **ORM**, que é dividido em três camadas onde cada uma delas representa um nível de abstração. Considerando a camada de mais baixo nível, próximo ao robô, a camada #1, denominada de camada física. Acima desta, tem-se a camada lógica e, por fim, a camada #3, camada de aplicação.

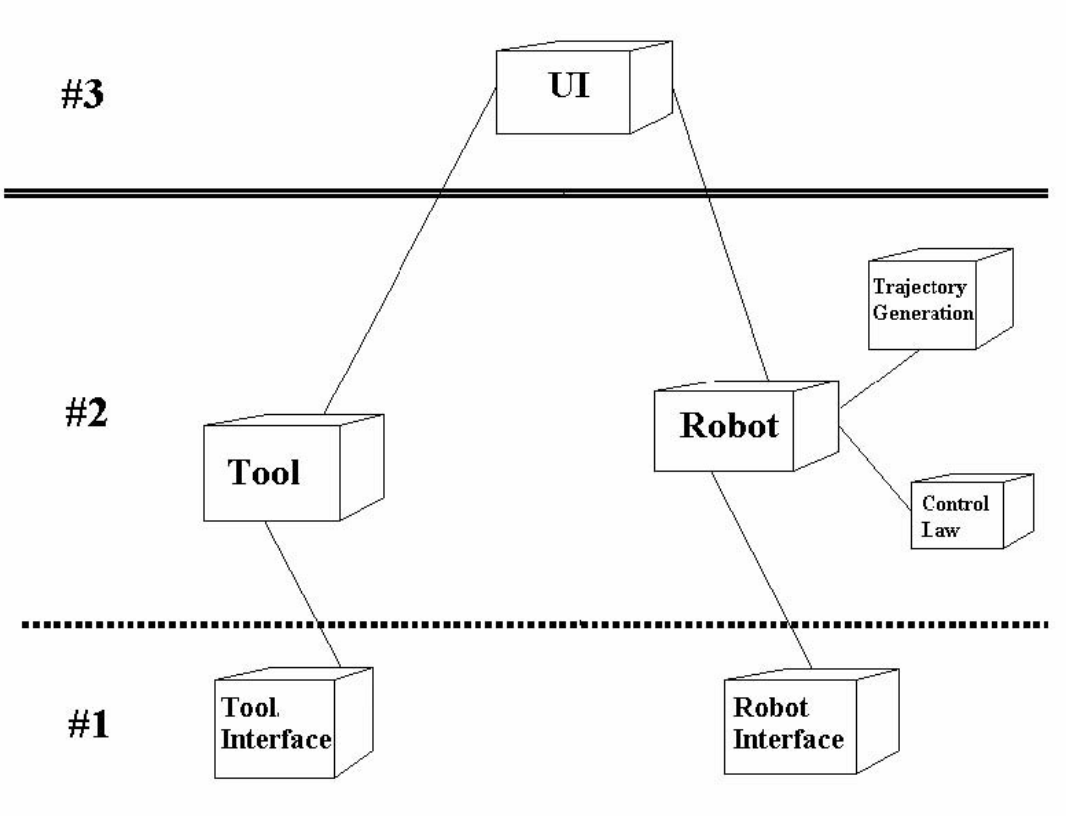


Figura 31: Diagrama de Distribuição do **ORM**.

A seguir, a descrição de cada pacote do diagrama de distribuição:

- **UI**: Responsável pela interface com o usuário.
- ***Robot***: Pacote que trata os aspectos dos modelos cinemáticos/dinâmicos do robô.
- ***Trajectory Generation***: Pacote que implementa o gerador de trajetória.
- ***Control Law***: Pacote que implementa a lei de controle associada ao robô.
- ***Tool***: Pacote que representa as ferramentas, efetuadores, de trabalho do robô.
- ***Tool Interface***: Biblioteca às placas de interface das ferramentas.
- ***Robot Interface***: Biblioteca às placas de controle de movimento.

Dando continuidade à especificação do modelo, será apresentado outro diagrama com o intuito de refinar o sistema modelado. Este diagrama fornecerá informações adicionais para a etapa de implementação do modelo.

A figura 32 apresenta o diagrama de componentes. Na **UML**, o diagrama de componentes mostra as organizações e as dependências entre componentes, que podem ser: um arquivo fonte, uma biblioteca, ou um arquivo executável. Assim, pode-se observar, por exemplo, que o pacote *robot*, encapsulado na biblioteca denominada *robot.dll*, é formado por dois arquivos: *robot.h* (**API**) e *robot.cpp (*Métodos*).*

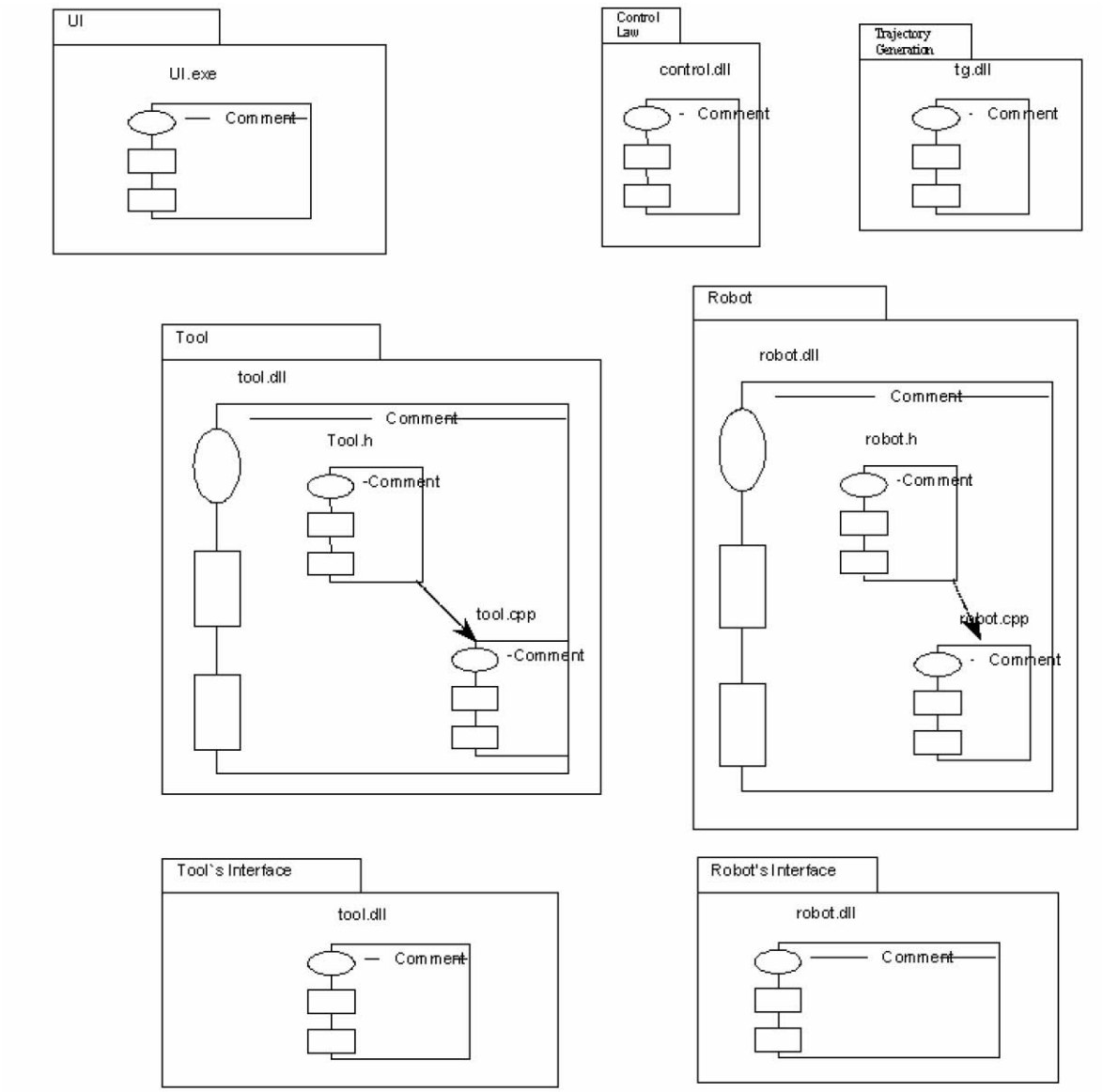


Figura 32: Diagrama de Componentes do **ORM**.

3.2 - Implementação Computacional

A implementação computacional segue gerando a **API** a partir do modelo **UML**, escolhendo-se qual a linguagem de programação computacional mais adequada para representá-la e seus respectivo **IDE**. Desta maneira, é dado início a especificação da infra-estrutura de apoio ao modelo.

O **OSACA** define uma infra-estrutura e a utiliza em pelo menos duas aplicações demonstrando sua viabilidade. O **OMAC**, afirma que todas as infra-estruturas são candidatas ao suporte de suas respectivas arquiteturas. Entretanto, a infra-estrutura do **OSACA** é específica, tornando-a conseqüentemente limitada e o **OMAC** não indica nenhuma recomendação tal que permita concluir algo a respeito. A estratégia adotada na tese foi a criação de uma infra-estrutura para robôs baseados em **PC** obedecendo aos padrões internacionais de *hardware* e *software* garantido suporte de sistemas abertos. A infra-estrutura baseada em PC especificada é complementada para dar suporte à **API** resultante do modelo. E garantindo flexibilidade, como por exemplo, ao sistema operacional, suporte de comunicação, sistema tempo-real, suporte ao processamento distribuído, ambiente de desenvolvimento e linguagem para codificação do sistema.

A **API** do **OMAC** é completa. Entretanto, complexa, com foi mostrado no estado da arte. Nesta tese, a **API** é criada a partir do modelo, então, para garantir que ela seja completa basta garantir que o modelo seja completo e fundamentado nos conceitos de orientação a objetos.

O **OMAC** usa um ambiente de desenvolvimento para dar suporte à implementação de sua arquitetura de referência resultando em sua **API**. Este trabalho também utiliza o **IDE** para implementação computacional e estende o seu uso ao longo do ciclo de desenvolvimento do sistema para integração e testes do sistema.

O **OMAC** emprega métodos formais de especificação em alguns de seus componentes e tem a preocupação de utilizar uma linguagem de especificação padrão para a descrição dos mesmos. Entretanto, a utilização de uma linguagem formal padronizada tanto na etapa de concepção do modelo quanto no momento da criação de sua **API** é algo novo na área de **OAC**`s. Isto nos dá a possibilidade de aplicar a engenharia direta e reversa tal que se obtenha sempre fidelidade bilateral entre o modelo e sua implementação computacional.

O suporte baseado em **PC** não é algo novo na área de **OAC**`s. Ele foi empregado a partir da década de 80, como foi mostrado no estado da arte desta tese, e consagrado no **OSACA** e no **OMAC**. O caráter inovador apresentado aqui é a extensão da utilização do **PC** ao longo da implementação computacional tal que sirva de suporte a aplicação, mas também ao desenvolvimento do sistema, às etapas de teste, integração e avaliação. Ou seja, foi realizada neste trabalho a especificação de uma infra-estrutura aberta baseada em **PC** não somente servindo de suporte ao **ORM**, mas também ao ciclo de desenvolvimento do sistema.

O **OMAC** preocupa-se em estabelecer uma metodologia para criação de um **OAC** através de uma **API**. A sistemática sugerida como forma de metodologia para o alcance do desenvolvimento e implementação de um sistema aberto para robôs é uma das conseqüências deste trabalho. A partir desta, pode-se estabelecer como ciclos de criação de um **ORM**:

- Definições das características abertas para um robô;
- Composição do modelo do sistema em substituição a arquitetura de referência;
- Geração da **API** a partir do modelo;
- Especificação da infra-estrutura de suporte a **API**;
- Testes de unidade na aplicação e
- Avaliações integradas em sua totalidade na aplicação.

3.3 - Aplicação do ORM

Por fim, a etapa final, aplicação do modelo computacional para resolução do problema. A utilização de robôs se dá mais comumente na indústria automobilística, suas principais tarefas neste setor são solda, pintura e montagem/desmontagem. Neste trabalho, a idéia foi à aplicação do robô em outras áreas.

A aplicação oportunamente escolhida para utilizar um modelo aberto para robôs também é de caráter inovador mundialmente. Não existe nenhum robô aberto aplicado à área de energia. Esta aplicação foi também pioneira neste sentido. Um modelo aberto para robôs baseado em **PC** foi criado para possibilitar adaptações dos robôs nas áreas de atuação, permitindo:

- Realimentar as Pesquisas e o Desenvolvimento do modelo; ou refinar o modelo;

- Demonstrar as vantagens proporcionadas por um robô aberto, viabilizar seu uso em diferentes áreas de aplicação;

O **OSACA** apresenta pelo menos duas aplicações de **OAC** no domínio de máquinas ferramentas. Assim sendo, os seus resultados são demonstrados desde sua arquitetura de referência, sua **API** implementada representando esta arquitetura e, por fim, as aplicações que comprovam suas afirmações e conclusões no decorrer de mais de uma década de trabalho. A quantidade de aplicações do **OMAC** é menor quando comparadas ao **OSACA**. Mas também consegue demonstrar seus resultados, o que motiva a continuação de suas pesquisas e desenvolvimento nesta área. Apesar disso, não se tem aplicações nem do **OSACA** nem do **OMAC** no domínio da robótica. Nesta tese são demonstradas, posteriormente, pelo menos três aplicações de um sistema aberto no domínio da robótica como forma de avaliação experimental do **ORM**.

Capítulo 4

Avaliações Experimentais

A seguir serão descritas as avaliações experimentais da tese que foram realizadas em três diferentes sistemas no âmbito dos projetos citados. Em cada um deles serão apresentados o modelo do sistema, a infra-estrutura de suporte ao modelo implementado e as aplicações realizadas a partir da utilização do **ORM**.

4.1 – Reis

O protótipo do manipulador Roboturb descrito no capítulo introdutório não estava concluído e era necessário dar continuidade ao desenvolvimento do sistema. Esse problema foi resolvido com a utilização de um robô que estava há algum tempo sem ser utilizado na **UFSC**. O robô, da marca REIS, viabilizou testes com os equipamentos e com o *software* do sistema. Através da plataforma de teste foi possível realizar ensaios dos algoritmos que realizam o controle do robô; do algoritmo de geração de trajetória; modelo cinemático e dinâmico; integração fonte solda e integração do sistema em sua totalidade (CHACONAS, et al., 1990) (ISAAK, 2005) (RAPOSO, 2007). O REIS modelo RV-15 é um robô com seis graus de liberdade, sendo estes originados por seis juntas rotativas. O acionamento é elétrico, através de motores de corrente contínua sem escovas. Para a medição da posição angular das juntas dispõe-se de codificadores de posição, *encoders* ópticos incrementais, montados em seus respectivos atuadores. Este manipulador é típico das aplicações da indústria. Ele é composto de um braço antropomórfico e um punho esférico. Em seu controlador encontram-se os servo-amplificadores e seu sistema computacional. O servo-amplificador do REIS é formado por seis placas com amplificadores de tensão. Em cada placa amplificadora se dispões de uma estratégia de controle, baseada em PID, que a partir de seus potenciômetros pode-se estabelecer e regular qual melhor esquema em função da aplicação. O sistema computacional é dedicado, utiliza barramento de comunicação **VME**, processador e sistema operacional dedicados.

Os problemas anteriormente expostos não puderam ser resolvidos utilizando o controlador original do robô, pois havia necessidade acrescentar/retirar ou atualizar algoritmos de estratégia de controle, modelo cinemático, integração de ferramentas e executivo tempo real. Assim, foi projetado um novo controlador, um controlador de arquitetura aberta baseado em computador pessoal denominado OpenReis.

4.1.1 - Modelo

O diagrama de distribuição especificado em **UML**, figura 33, mostra o modelo da estrutura do OpenReis, ou seja, de que maneira seus elementos, destacados na cor vermelha, estão distribuídos e como interagem entre si. O modelo é composto de três camadas na qual cada uma delas representa um nível de abstração. Considerando a camada de mais baixo nível, próximo ao robô, a camada #1, denominada de camada física. Acima desta, tem-se a camada lógica e, por fim, a camada #3, camada de aplicação.

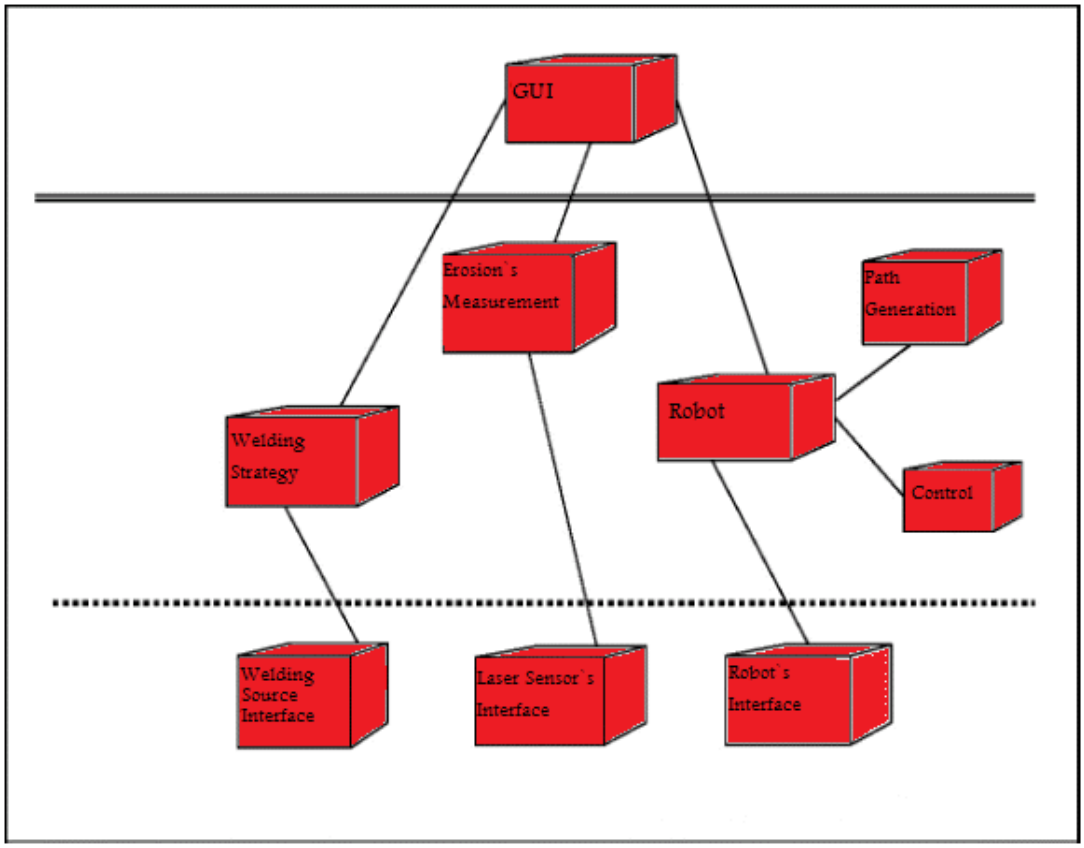


Figura 33: Diagrama de Distribuição do **ORM** aplicado ao OpenReis.

Neste caso, pode-se observar a forma como o modelo permite tratar o processamento de eventos concorrentes e distribuídos em cada camada. Ou seja, observe que o processo de soldagem e acionamento do robô são tarefas que podem ser executadas de modo concorrente. A seguir, a descrição de cada pacote do diagrama:

- ***GUI** (Graphical User Interface)*: responsável pela interface com usuário.
- *Robot*: pacote que trata dos aspectos de acionamento do robô.
- *Path Generator*: pacote responsável por implementar o gerador de trajetória.
- *Control*: pacote que implementa o controlador do robô.
- *Welding Strategy*: pacote que define a melhor estratégia para soldagem.
- *Erosion's Measurement*: pacote que tem como objetivo a medição dos pontos de erosão.
- *Weld Source Interface*: biblioteca dinâmica para a interface com a fonte de solda.
- *Laser Sensor's Interface*: biblioteca dinâmica para a interface com o sensor laser.
- *Robot's Interface*: biblioteca dinâmica para a interface com o manipulador.

O diagrama da figura 34 é o diagrama de caso de uso do pacote ***GUI***, expandindo em mais detalhes. Nele foram mapeados sete casos de uso referente a interface Homem-Máquina relacionado a camada de aplicação do sistema.

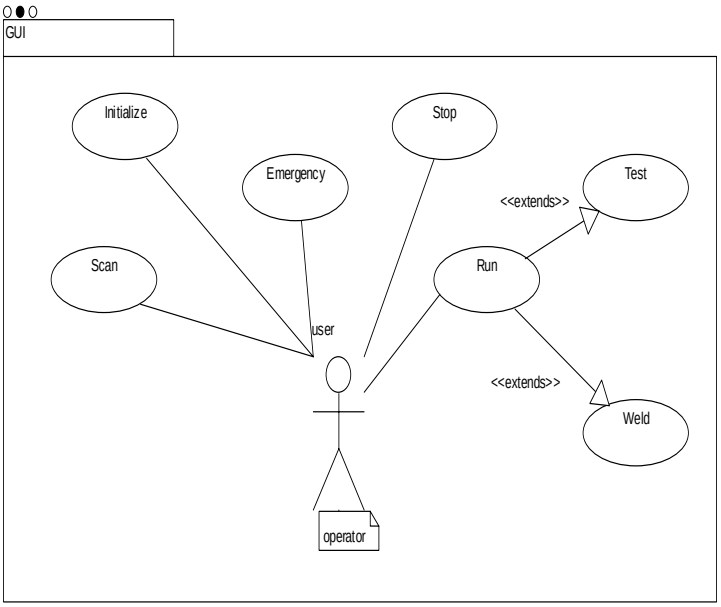


Figura 34: Diagrama de Caso de Uso do Pacote **GUI** do OpenReis.

A seguir, serão comentadas as funcionalidades de cada caso de uso:

- *Initialize*: captura o processo de inicialização do manipulador.
- *Scan*: realiza a varredura da superfície a ser soldada.
- *Run*: estágio em que o robô está pronto para execução.
- *Test*: processo de simulação da trajetória a ser feita durante a soldagem.
- *Weld*: realiza o processo de soldagem.
- *Stop*: parada do sistema.
- Emergency: parada de emergência do sistema por uma situação de falta.

Na figura 35 é apresentado o diagrama de transição de estado. Este diagrama mostra a evolução da operação do sistema mapeando os casos de uso em estados.

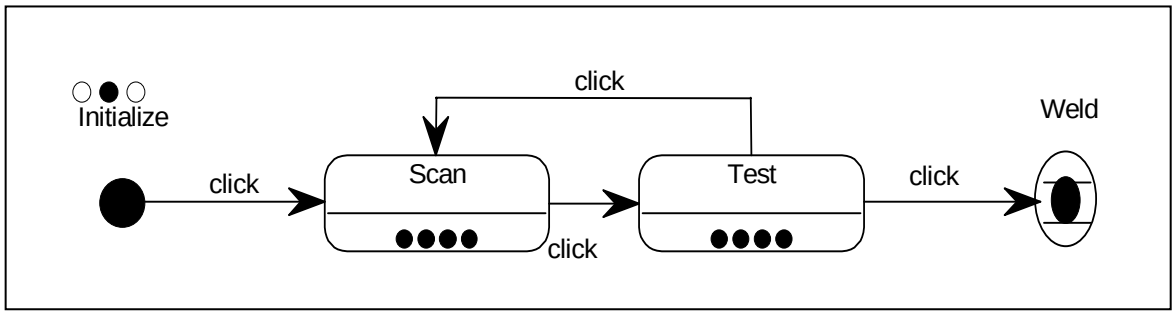


Figura 35: Diagrama de Estado do Sistema OpenReis.

Assim, os estados que corresponde a operação do sistema são:

- *Initialize*: É o estado inicial do sistema. Nele, o robô foi inicializado e está pronto para operação. Este estado compreende o processo de referenciamento do manipulador em relação ao seu sistema de coordenadas, carregamento dos parâmetros da lei controle, compensação do trilho, e, por fim, a habilitação de todas as juntas do manipulador.

- *Scan:* Neste estado são realizadas as etapas de medição manual para delimitação da região erodida por cavitação bem como o processo de medição automática a partir da ferramenta denominada sensor laser também conhecido como cabeçote de medição a cratera que irá receber a solda é digitalizada.

- *Test:* Este estado representa a verificação cinemática das trajetórias dos processos de medição e soldagem automática juntamente com suas respectivas ferramentas tal que seja possível encontrarem uma solução que possa operar no espaço de trabalho do robô sem haver colisão.

- *Weld:* Este é o estado final do sistema, representa o processo de deposição de cordões de solda na cratera que foi previamente digitalizada. Nele acontece o controle da máquina de solda e da tocha para efetivamente ocorrer a solda propriamente dita.

Neste caso da modelagem do sistema, são destacados os eventos, exemplificando o *click* de um operador, que causam uma transição de um estado para outro e as ações que resultam dessas mudanças. Sendo assim, pode-se afirmar que este diagrama expressa o comportamento dinâmico do robô.

A partir da especificação do modelo hierárquico do robô foi realizada a codificação do sistema. Foram utilizadas duas ferramentas: System Architect, versão 4.0, e o Cbuilder Enterprise, versão 4.0. A partir do modelo **UML**, com o System Architect foi gerado o esqueleto da hierarquia de programação em C++, bem como as respectivas DLL's, de acordo com o diagrama de componentes. De posse deste esqueleto, utilizou-se o Cbuilder para implementar os pacotes. Como exemplos, as **API**`s *Robot, Scan* e *Weld* são respectivamente mostradas nos apêndices deste trabalho.

4.1.2 - Infra-estrutura

Assim, posteriormente a implementação, foi projetado um novo controlador para suportar o modelo do sistema. Um controlador de arquitetura aberta baseado em **PC** utilizando barramento de comunicação padrão **PCI** e sistema operacional Windows NT 4.0, aproveitando-se do elemento de manipulação e amplificador de potência do REIS. A figura 36 apresenta a infra-estrutura do OpenReis, explicitando posteriormente os aspectos de *software* e *hardware*.

Figura 36: OpenReis.

Software

- **UML:** System Architect 4.0
- **IDE:** C++Builder 4.0
- **OS**: Windows NT
- *Driver*: Flexmotion 5.1

Hardware

- **PC** ou compatível: PC AMD 1GHz
- Controle de Movimento: MC 7344 PCI
- Interface Universal: UMI 7764
- Captura de Imagem: IMAQ1408 PCI

4.1.3 - Aplicação

Após toda concepção do Robô Aberto, cerca de 40 ensaios de soldagem foram realizados a fim de validar o sistema. Eles foram feitos inicialmente com chapas planas e depois evoluíram para chapas fresadas que simulavam uma superfície cavitada. Todos os testes foram feitos com o corpo de prova posicionado com uma inclinação de 45º sobre-cabeça, situação de pior caso para soldagem. A bancada utilizada consiste do OpenReis incluindo fonte de soldagem microprocessada, modulo Plasma, alimentador de arame e suporte à chapa ou corpo de prova, como mostra a figura 37.

Figura 37: Bancada Experimental OpenReis.

A partir do OpenReis foi possível fixar o sensor óptico, cabeçote de medição, para realizar a medição inicial da superfície do corpo de prova de maneira concorrente com o controle de movimento das juntas do manipulador, figura 38.

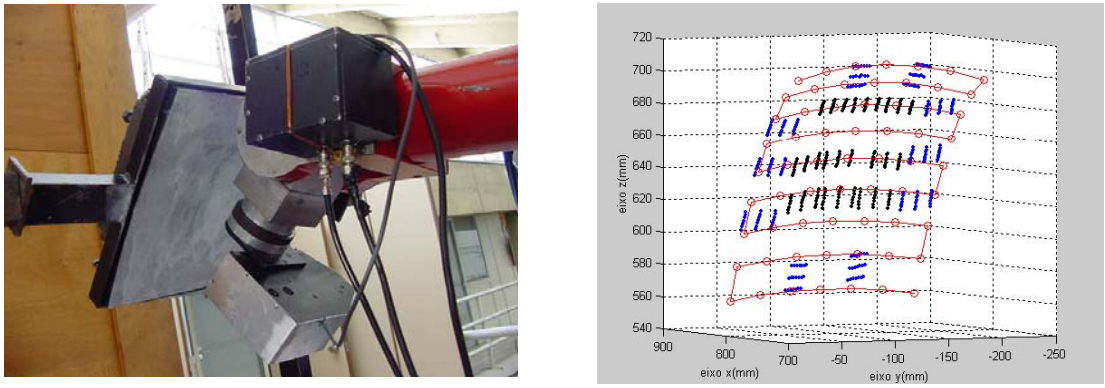


Figura 38: Medição automatizada com OpenReis.

Os pontos em azul, pontos externos, definem a região não danificada que envolve a cavitação. Os pontos em preto, pontos internos, caracterizam a forma da região danificada. A trajetória em vermelho do tipo zig-zag representa a medição automatizada da cavidade do corpo de prova. O resultado do processo de medição pode ser visto na figura 39.

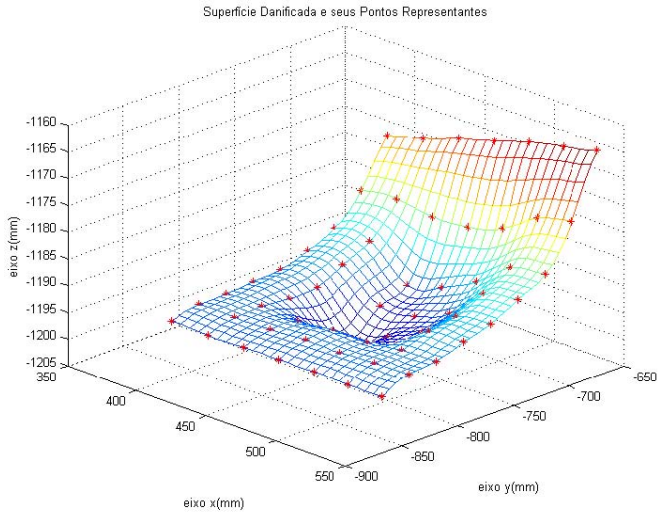


Figura 39: Cratera digitalizada com OpenReis.

A partir do processo de medição, algoritmos matemáticos e de interpolação reconstroem as superfícies danificada e original. A superfície original é o modelo que se deseja alcançar com o processo de soldagem de preenchimento. A máxima profundidade existente, calculada a partir da diferença entre a superfície original reconstruída e a superfície danificada.

A figura 40 mostra como exemplo a superfície danificada e a localização dos pontos, em preto, referentes às trajetórias dos cordões de solda da primeira camada de soldagem.

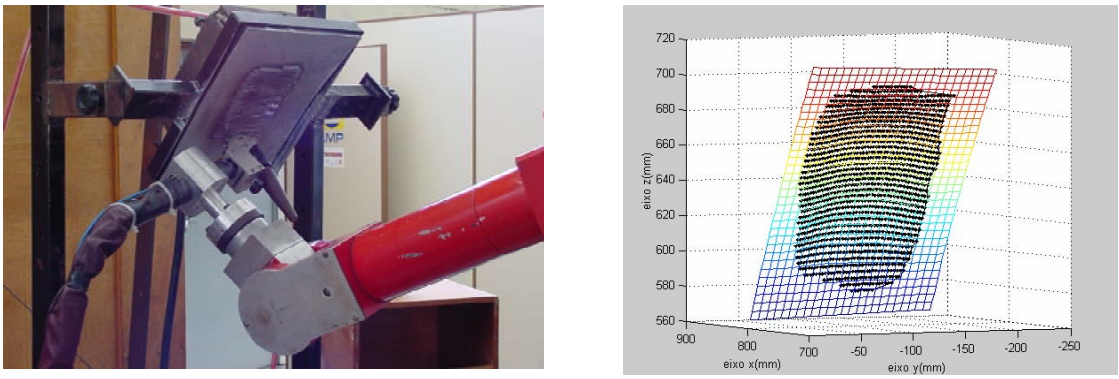


Figura 40: Soldagem automatizada com OpenReis.

A partir do OpenReis foi possível fixar a tocha de soldagem Plasma. Desta maneira, foi realizada a adição de material de acordo com a trajetória indicada pela figura 40 e já convertidas para o espaço operacional, controlando concorrentemente os movimentos e o disparo da fonte de solda juntamente com o alimentador automático de arame. O resultado final do processo de soldagem é mostrado na figura 41.

Figura 41: Primeira recuperação completa de uma cratera realizada com o OpenReis.

4.2 – Roboturb

Com a conclusão do projeto mecânico do Roboturb e a sua montagem, iniciou-se uma etapa de transição no projeto do *software*. O objetivo era aproveitar tudo aquilo que já se tinha alcançado usando o OpenReis, adaptando ao Roboturb conforme a demanda (RAPOSO, et al., 2003). As adaptações que foram necessárias ao Roboturb viabilizadas pelo **ORM** são:

- Criação de um *Teach Pendant* (#1)
- Novo Manipulador (Roboturb)
- Integração de Trilhos flexíveis com ventosas e sapatas magnéticas
- Novo Cabeçote de medição (#2) e nova Tocha (Carijó)
- Criações de amplificadores de potência #1 e #2
- Sintonia do PID
- Substituição do PC por um PC industrial (Jabuti) com barramento PXI

4.2.1 - Modelo

Na figura 42 que representa o modelo aberto do Roboturb são destacados na cor laranja os elementos que refletem as adaptações. Os demais elementos são reutilizados a partir do Openreis.

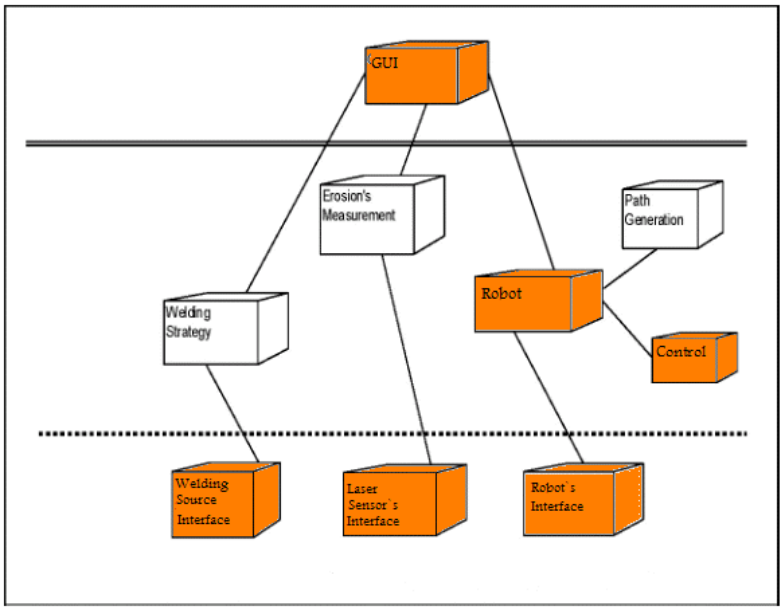


Figura 42: Diagrama de Distribuição do **ORM** aplicado ao Roboturb.

Com o intuito de tornar a operação do robô ergonômica foi criado um *Teach Pendant* (TP), um painel de controle, para servir de interface entre o operador do sistema e o manipulador. Portanto, criou-se uma classe, figura 43, que encapsula o seu funcionamento. Para o operador, os componentes do TP podem ser divididos em três grupos distintos: visor, teclado e manoplas. Seguindo esta divisão física, dividiu-se também a estrutura da classe *RTeachPendant*. Esta classe trata apenas dos métodos de funcionamento geral do TP, ou que requeiram a interação entre dois grupos de componentes. Cada grupo de componentes possui uma classe própria, com um objeto instanciado dentro da classe *RTeachPendant*. A Figura 43 nos mostra o diagrama de classes do *Teach Pendant*.

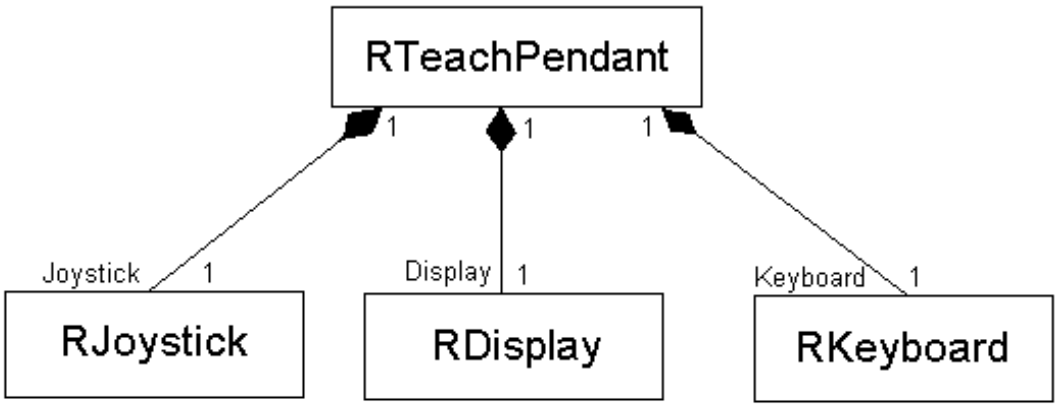


Figura 43: Diagrama de classes do *Teach Pendant.*

A classe *RJoystick* trata de todas as funcionalidades relacionadas às manoplas, *track-whell* e chaves de modo de operação. Sua operação é baseada no método *Refresh*, que promove a atualização do estado destes recursos. Uma vez atualizado o estado, podem ser chamados os métodos para leitura do valor destes recursos. Existem também métodos de calibração, que são utilizados para um ajuste por *software* dos limites de operação das manoplas. O visor está associado à classe *RDisplay*. Esta classe permite ao operador enviar textos e gráficos ao visor, assim como fazer ajustes de contraste e *BackLight*. Todos os botões do teclado numérico, teclas de funcionalidades e o botão da manopla estão associados ao método *RKeyboard*. Esta classe possui métodos que permitem a leitura destas teclas caso tenham sido pressionadas. Dentre os métodos que compõem a classe *RTeachPendant*, deve-se destacar o método *EditNumber*, que abre no display um campo para que possa ser editado um número. Este método encontra-se nesta classe porque utiliza recursos de dois dos grupos de recursos: o teclado e o visor. Deste modo a classe *RTeachPendant* encapsula as demais classes, fornecendo acesso ao TP a partir de uma instanciação.

4.2.2 - Infra-estrutura

A infra-estrutura especificada para suportar as necessidades do modelo do Roboturb tal que satisfaça os requisitos da aplicação é mostrada a seguir. Assim, na figura 44, é apresentado o controlador para o Roboturb e juntamente com seu *teach pendant* utilizando o **ORM**. Posteriormente, são explicitados os aspectos de *software* e *hardware*.

Figura 44: **ORM** aplicado ao Roboturb.

Software

- **UML:** Rational Rose
- **IDE:** C++Builder 4.0
- **OS**: Windows NT Embedded
- *Driver*: Flexmotion 5.1

Hardware

- **PC** ou compatível: PC *Pentium* 700 MHz
- Controle de Movimento: MC 7344 PXI
- Interface Universal: UMI 7764
- Captura de Imagem: IMAQ1408 PXI

4.2.3 - Aplicação

Os primeiros resultados da aplicação do **ORM** no Roboturb tornaram possível a operação do robô seguindo trajetórias programadas, ponto a ponto, pelo operador e disponível para usar o processo de soldagem MIG como foi apresentado na feira Internacional de Mecânica em 2002, figuras 45 e 46.

Figura 45: Roboturb na Feira Internacional de Mecânica em 2002.

Figura 46: Resultados do Processo de Soldagem MIG com Roboturb.

A partir dos trabalhos de definição da estratégia de medição e soldagem usando o processo plasma (BONACORSO, 2004) a bancada experimental mostrada na figura 47 foi adaptada aos novos ensaios para validação utilizando o manipulador e o **ORM** do Roboturb, obtendo os resultados como mostra a figura 48.

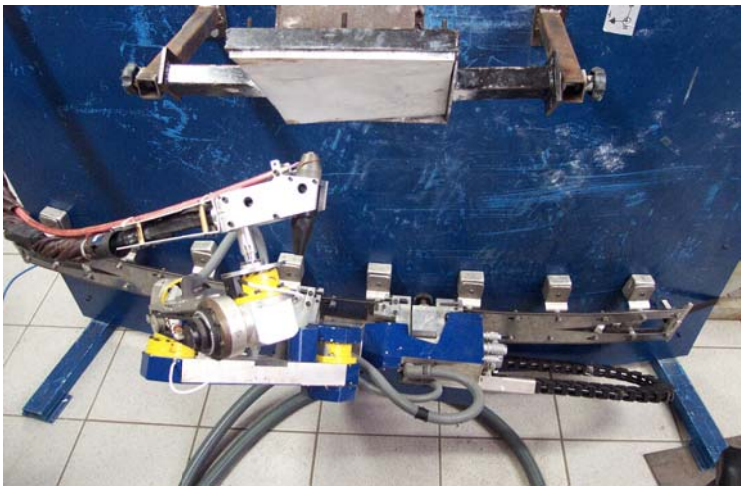


Figura 47: Bancada Experimental para processo PLASMA.

Figura 48: Resultado da Recuperação de uma cratera com o Roboturb.

A visita a campo do projeto a usina hidrelétrica de Governador Bento Munhoz (GBM) para aplicação do Roboturb. Nesta última visita, as etapas de medição e soldagem do processo de recuperação foram realizadas com sucesso utilizando o **ORM** Roboturb, conforme figuras 49 e 50, resultando na primeira cavidade recuperada completamente em campo, com deposição de quatro camadas. A recuperação da cratera foi realizada em paralelo aos reparos manuais de trincas encontradas na turbina, simulando novamente a condição pretendida que é a dê ter quatro frentes de trabalho simultâneo na turbina.

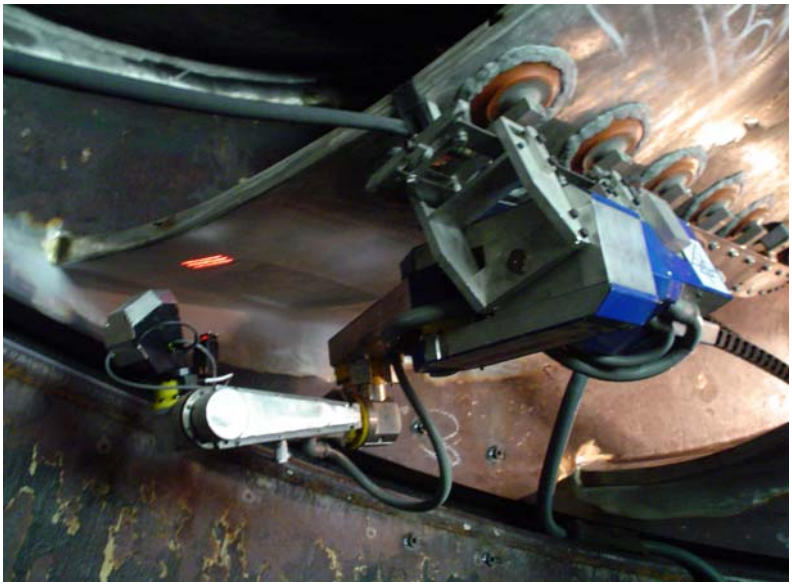


Figura 49: Medição Automática com Roboturb.

Figura 50: Soldagem Automática com Roboturb.

Com esta recuperação completa realizada em campo, foi demonstrada a viabilidade técnica do projeto para realizar a deposição automatizada de material por soldagem, para recuperação de crateras ocasionadas por cavitação, como mostra a figura 51.

.

Figura 51: Resultado da Recuperação Automática com o Sistema Roboturb na Usina Hidrelétrica de **GBM**.

4.3 – Robofurnas

As idas a campo proporcionaram a elaboração de novos requisitos o que refletiu em adaptações, tais como:

- Novo *Teach Pendant* (DISPLAY)
- Novo Trilho flexível + trilho de estacionamento
- Novo manipulador com 8 eixos de apoio no carro e com giroscópios integrados
- Nova máquina de solda com módulo plasma e alimentador de arame integrados
- Criação de um amplificador de potência #2
- Sintonia do PID

4.3.1 - Modelo

O diagrama de distribuição do **ORM**, figura 52, é destaca na cor azul as adaptações que foram necessárias ao Robofurnas. Os demais elementos são reutilizados a partir do Roboturb.

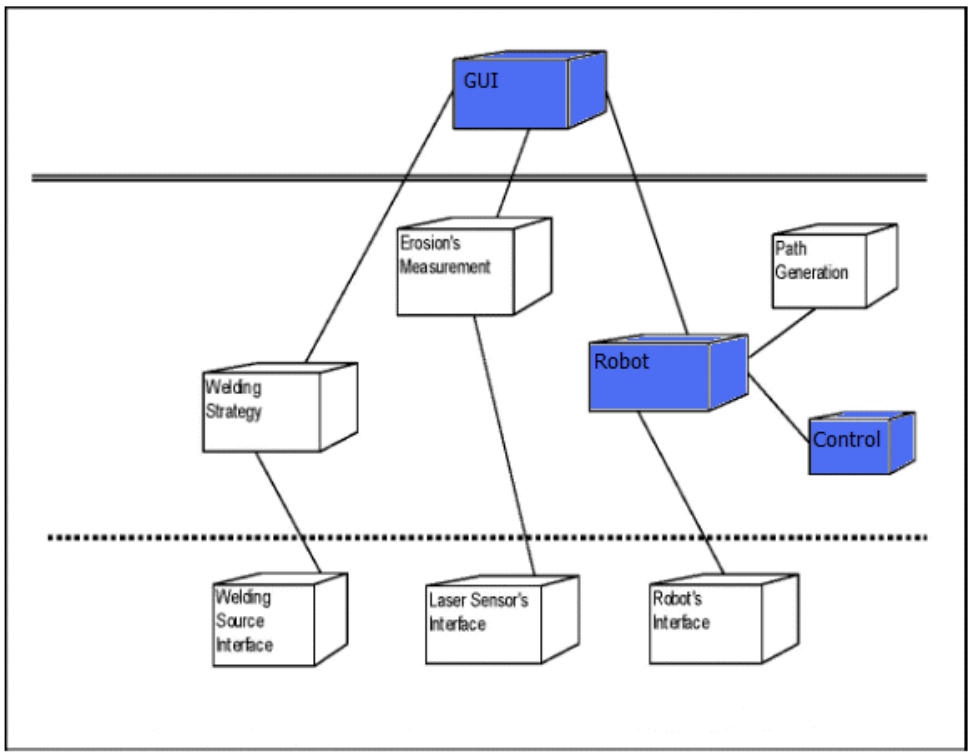


Figura 52: Diagrama de Distribuição do **ORM** aplicado ao Robofurnas.

4.3.2 - Infra-estrutura

A infra-estrutura especificada para suportar as necessidades do modelo do Robofurnas tal que satisfaça os requisitos da aplicação é mostrada a seguir. Assim, na figura 53, é apresentado o Robofurnas juntamente com seu *teach pendant* utilizando o **ORM**. Posteriormente, são explicitados os aspectos de *software* e *hardware.*

Figura 53: **ORM** do Robofurnas.

Software

- **UML:** Rational Rose
- **IDE:** C++Builder 4.0
- **OS**: Windows 2000 Embedded
- *Driver*: Flexmotion 5.1

Hardware

- **PC** ou compatível: PC *Pentium* 700 MHz
- Controle de Movimento: MC 7344 PXI
- Interface Universal: UMI 7764
- Captura de Imagem: IMAQ1408 PXI

4.3.3 - Aplicação

A visita a campo do projeto Roboturb a usina de Governador Bento Munhoz para aplicação do manipulador juntamente com o **ORM** Roboturb. O teste do ponto de vista do desenvolvimento do projeto foi muito positivo porque foi possível testar os diversos módulos que compõem o sistema em condições reais de aplicação, inclusive com serviços paralelos típicos da manutenção, como esmerilhamento e outras pás sendo soldadas. Em certos momentos simulou-se a condição pretendida que é a dê ter quatro frentes de trabalho simultâneo na turbina. Porém este ambiente também atrapalhou o teste e ainda ao final retirou recurso do robô que levaram a conclusão ainda prematura do teste. Por fim foram superadas todas as etapas do processo, mesmo que com dificuldades. Também esta foi a maior camada já depositada em condições reais de turbina pelo robô, pois nas duas outras oportunidades não se conseguiu ir tão longe. No primeiro teste (COPEL/**GBM** 06/2004) não foi possível passar pela etapa de medição da cratera, e no segundo teste (FURNAS/Estreito 10/2004) foi possível chegar até a etapa de soldagem, onde foi depositado 14 cordões, mas estes tinham uma qualidade muito pior do que agora, quando foi realizado 45 cordões de boa qualidade, muito próxima do que foi obtido em laboratório. Entretanto, não foi concluída a recuperação da cratera em campo mais na bancada experimental utilizando o Robofurnas, figura 54.

Figura 54: Bancada experimental Robofurnas.

A cratera encontrada na usina de estreito foi digitalizada pelo processo de medição automática e usinada em aço para que em laboratório fosse completada a recuperação e validação do Robofurnas, como demonstra a figura 55.

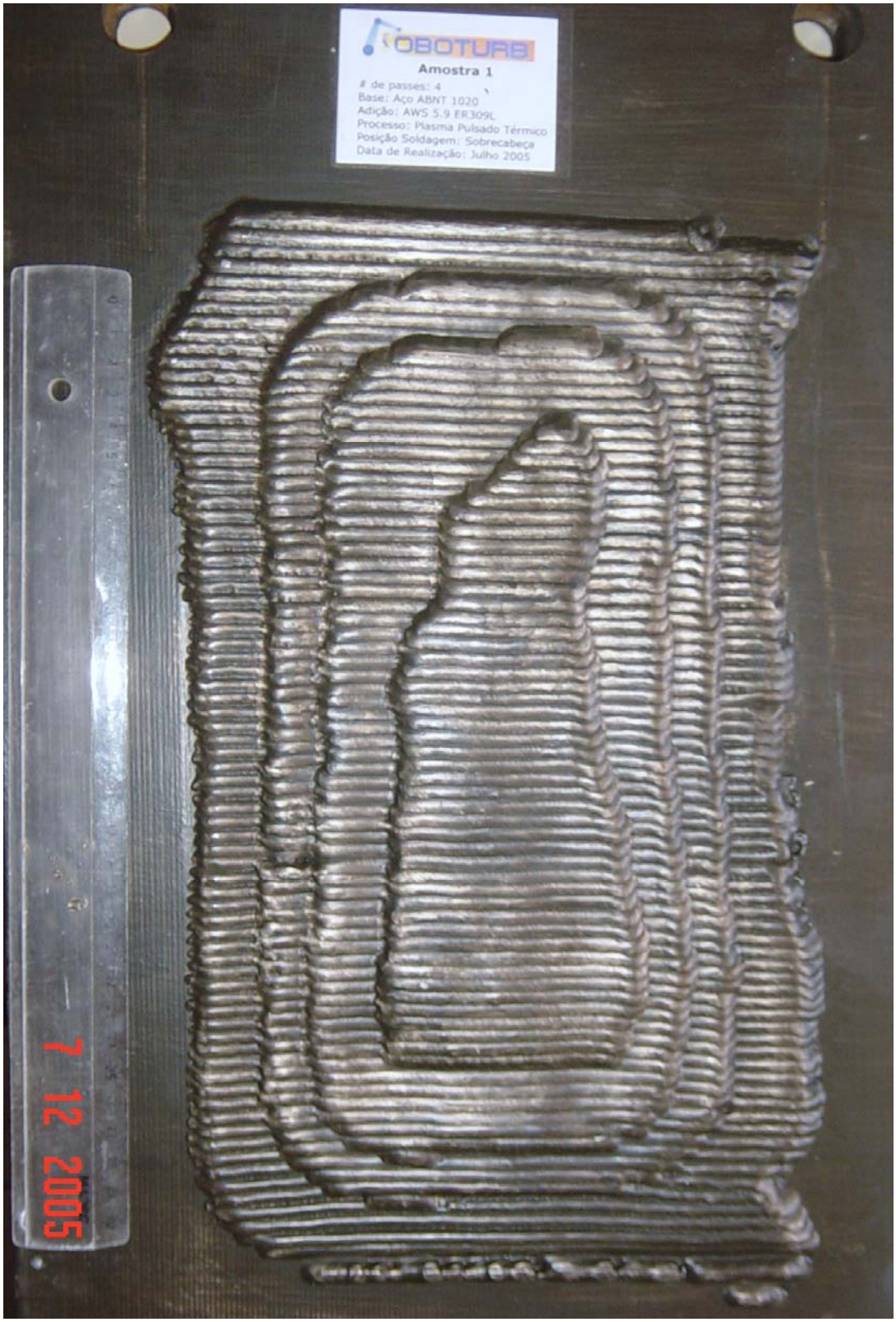


Figura 55: Primeira recuperação completa de uma cratera realizada com o Robofurnas.

Conclusões

Em termos gerais, neste trabalho é proposto um modelo aberto para robôs que atenda a especificação formal de sistema aberto, respeite padrões internacionais de *software* e *hardware*, flexibilize as aplicações na área de robótica e permita, dessa forma, disseminar a utilização de robôs em áreas até então pouco exploradas. Para tal, é definida uma especificação rigorosa de sistema aberto, de modo a fornecer suporte para a implementação de sistemas de controle de robôs usando a filosofia de sistemas abertos. Ao final do trabalho é apresentado um modelo aberto para robôs independente de infra-estrutura, implementado usando orientação a objetos a partir dos diagramas **UML** e aplicado em uma área atípica para demonstração do seu ineditismo.

Em relação a abertura alcançada pelo modelo, ela é completa. Conforme abordado anteriormente, existem três níveis de abertura de um sistema: o nível que abre somente a interface gráfica com o usuário, que é o mais básico, o nível que abre também a estratégia de controle e, por fim, o mais completo, que abre os componentes: ferramentas, manipulador, controlador. A implementação do modelo proposto foi realizada com o objetivo de validá-lo, obter consistência e realimentar a pesquisa com os resultados obtidos.

A compilação do estado da arte em controladores de arquitetura aberta para robôs mostra que não existe consenso no que seria um **OAC** para Robôs. Neste trabalho, as definições das características de **OAC** para robôs são realizadas com base nos padrões **IEEE** de sistemas abertos e engenharia de *software*. E assim, é mostrado que é possível haver conformidade do que seria um sistema de controle aberto para robôs a fim de tornar plausível a portabilidade, escalabilidade, extendibilidade, interoperabilidade, intercambialidade, reusabilidade, reconfigurabilidade, flexibilidade e confiabilidade de componentes robóticos.

O estudo do estado da arte em **OAC** para robôs não estabelece nenhuma taxonomia diferentemente do que foi observado no caso dos **OAC** para automação: **OSACA**, **OMAC** e **OSEC**. Esta taxonomia compreende as definições que caracterizam um **OAC**; a arquitetura de referência; **API**; infra-estrutura; aplicações. Neste trabalho é mostrado que é possível simplificar a taxonomia através das etapas de modelagem computacional, implementação do sistema e aplicação.

Ao concretizar o estado da arte foi constatada que a maior parte das abordagens de **OAC** utiliza os paradigmas da orientação a objetos em suas arquiteturas. Entretanto, estas arquiteturas ou modelo de **OAC** são especificados de maneira incompleta, representados sem nenhuma padronização, dependentes da infra-estrutura, não apresentam aplicações no domínio da robótica o que ocasiona o comprometimento na obtenção de resultados.

Durante os últimos 20 anos vem se estudando maneiras de criar sistemas flexíveis através dos conceitos de sistemas abertos. A proposta **OSACA**, dentre as características de um sistema aberto para robô, contempla a portabilidade, extendibilidade, escalabilidade, interoperabilidade e intercambiabilidade usando camadas de abstração baseado no modelo **OSI** bem como a utilização do paradigma de orientação a objetos. Já no caso do **OMAC,** as características tratadas são a reusabilidade e re-configurabilidade abalizada no conceito de componentes.

Nesta tese é apresentado o modelo **ORM**, que engloba estrutura e comportamento, completamente especificado, mostrando quais os pacotes que encapsulam os componentes de um robô. Ele é expresso de maneira padronizada, unificada e fundamentado em orientação a objetos. Este modelo, para garantir a característica da flexibilidade, é concebido de maneira *top/down*. E assim, tornando-se independente de qualquer infra-estrutura. Já para contemplar a característica da confiabilidade aos componentes do modelo, fez-se uso da especificação formal através da **UML**.

A sistemática apresentada durante a tese para construir um controlador para robô compreende a modelagem do sistema, sua implementação gerada a partir dos diagramas **UML** que expressa o modelo, e por fim, a aplicação como forma de avaliação do modelo do controlador. Esta maneira viabiliza a automatização do processo de desenvolvimento e implementação de controladores de arquitetura aberta para robôs.

A avaliação experimental do **ORM** mostra que um robô industrial pode ser aberto para atender um problema específico para o qual não foi projetado. Após o robô ser aberto, foi possível acrescentar **HMI**, PID, TG; Processos de medição e soldagem. Posteriormente, o sistema aberto para robô, OpenReis, foi validados através da recuperação de corpos de prova que representam as regiões erodidas das pás ocasionadas por cavitação utilizando processos de soldagem MIG e PLASMA,

Os componentes criados no OpenReis, tais como: **HMI**, TG e Tools com seus respectivos processos são reusados tanto no Roboturb como Robofurnas. Isto permite que se desenvolva e implemente robôs com maior rapidez para diferentes áreas de aplicação.

O Roboturb é um robô de serviço, pois é um sistema específico para recuperação de pás de turbinas. Para se conseguir desenvolvê-lo foi necessário realizar simultaneamente o desenvolvimento e a implementação do manipulador, ferramentas/processos e controlador. Assim, após a concepção do manipulador pode-se reusar o controlador do OpenReis bem como as ferramentas/processos em função das características de sistemas abertos.

No caso do Robofurnas, este é uma replica do Roboturb, mas com o enfoque a robustez levantada a partir das aplicações em campo do Roboturb. Aspectos tais como: **HMI**, Controlador, Máquina de Solda, Manipulador, Cabeçote de Medição tiveram que ser adotados em função dos requisitos impostos pela agressividade do trabalho em campo. Assim, graças ao sistema aberto podem-se viabilizar as adaptações dos componentes para garantir a confiabilidade exigida pela aplicação.

Em virtude do **ORM** foi possível reutilizar os componentes Tools e TG. Além disso, por exemplo, pode-se flexibilizar:

- Interface com usuário **escalável** a pelo menos dois diferentes tipos de *Teach Pendant.*
- Estratégia de controle **re-configurável** a pelo menos dois diferentes tipos de manipuladores e efetuadores;
- Gerador de trajetórias **interoperável** com o restante do sistema tal que permite a programação *offline* da trajetória utilizando como padronização as coordenadas cartesianas sem a necessidade de uma linguagem de programação de robôs específica.
- Modelo cinemática **extendível** em função das variações dos manipuladores e necessidades de calibrações;
- Ferramentas e processos **intercambiáveis** entre os sistemas. Utilizando pelo menos dois diferentes tipos de processos e por conseqüência suas ferramentas;
- Manipuladores **portáveis** entre controlador em função das necessidades de uso de diferentes morfologias;
- Sistema tempo real **adaptável** para ser adequado aos requisitos da aplicação;
- Robôs aplicados **confiáveis** a partir de sua avaliação de desempenho;

Por fim, conclui-se, a partir das avaliações experimentais, que tanto os robôs industriais como os robôs de serviço necessitam serem abertos motivados pelas atualizações das evoluções científicas e/ou tecnológicas em robótica considerando a aplicação ou o seu ambiente. O robô industrial por definição é multipropósito isto o torna genérico para atende diversificado escopo bem conhecido de aplicações. E quanto mais genérico menor as chances de alcançar o desempenho desejado à aplicação. No caso do robô de serviço, a aplicação é específica. O que aumenta a probabilidade de sucesso na resolução do problema, mas sua especificidade pode ser afetada por causa da variabilidade de ambientes nos quais este robô pode ser aplicado. Logo, os robôs industriais e de serviço deverão ser abertos tal que sejam flexíveis o suficiente para serem ajustados às aplicações ou ambientes. E para tal, deverão empregar a teoria de modelo aberto para robôs.

ANEXO 1 – API *ROBOT*

Tabela 10: **API** *Robot*

```
//---------------------------------------------------------------------------
#ifndef RobotTypeH
#define RobotTypeH

#include <Flexmotn.h>
#include <Flexcomp.h>
#include <Image_Kinematics.h>

#define AXISINFO_NO_BUFFER 0
// Valores válidos de buffer: 1 - 255 (0x01 - 0xFF).
// 0 significa nenhum buffer configurado.

//---------------------------------------------------------------------------
typedef struct AxisInfo
{
  u8 Position;        // posição do eixo na placa de aquisição
  int Board;          // indica qual das placas de aquisição o eixo encontra-se
  u8 Buffer;          // Endereço do buffer de posições correspondente ao eixo.
  unsigned long int BufferSize;      // Tamanho do buffer em no. de posições.

  i32 DefVelocity;     // a velocidade padrão do eixo
  i32 DefAcceleration; // a aceleração padrão do eixo

  i32 ENCPosition;     // as posições do eixo em pulsos de encoder
  double RADPosition;  // as posições do eixo em radianos
} AxisInfo;

//---------------------------------------------------------------------------
class RobotType
{
  public:   AxisInfo Axis[7];

        Input_Direct_Kinematics DirectKin;
        u16 AllAxesBoard1, AllAxesBoard2;
        int nAxes;
        bool isRoboturb;       // Booleano que define se é Roboturb
        u16 RobotStatus;       // Dá o status das operações em hardware

        // Methods

        RobotType();
        void SetRobot(bool);
        bool GetAxisPosition(i32 *PositionEnc, int AxisNum); // Grava em PositionEnc a posição do eixo AxisNum
        int  GetAxesPositions(i32* PositionsEnc); // Grava em PositionsEnc as posições de todos os eixos (0 -> N)
};

//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
#endif   //RobotTypeH
```

ANEXO 2 - API *SCAN*

Tabela 3: **API** *Scan*

```
/*
* Projeto RoboTurb/RoboFurnas
* Rimage.h
*
* Copyright (C) 2003 Labmetro
*
* Autor: Diego Carvalho dic@labmetro.ufsc.br
*        Luan Perdomo lpy@labmetro.ufsc.br
*        Roger Schipmann
*        Tiago Costa tlp@labmetro.ufsc.br
* Data: 27/06/2003
*
* Histórico:
*
*/
//---------------------------------------------------------------------------
#ifndef CapturaH
#define _NIWIN WIN32
extern "C" {
#include "nivision.h"
#include "niimaq.h"
};
#include <registry.hpp>
#include "Main_Unit.h"


#define CapturaH
//---------------------------------------------------------------------------
/**
*
*  Classe Rimage
*
* Classe que funciona como camada extra para as funções NIvision
* Responsável por chamar as subrotinas da dll
* Criada com o intuito de separar as chamadas de biblioteca do escopo de outras
* classes
*
* @author Diego Carvalho dic@labmetro.ufsc.br
*         Luan Perdomo lpy@labmetro.ufsc.br
*         Roger Schipmann
*         Tiago Costa tlp@labmetro.ufsc.br
* @date Junho, 2003
* @version 1.0
*
*/
class Rimage
{

private:
   int erro;//contém o código do último erro gerado
   //caso não haja nenhum erro possui valor 0
   Image* image;
   Image* image2;
   void* arrayImage;
   int setupGrab();//operação necessária para inciar o 'grab'
   INTERFACE_ID interfaceID;
   SESSION_ID sessionID;


public:
   Rimage();
   ~Rimage();
   void capturaImagem();//Captura a imagem atual da câmera
   void grab(); //Captura imagens da camera continuamente
   int getErro(); //retorna o valor do último erro gerado
   void stopGrab(); //Pára a execução do grab
   Image& getImage();
   Image& getCopy();
   void copyImage();
   void salvaTiff(Image*); //Salva imagem com a extensão tiff
```

```
  void salvaBmp(Image*);  //Salva imagem com a extensão bmp
  void salvaBmp(char*);  //Salva imagem com a extensão bmp
  void salvaBmp();
  void salvaTiff2(char*); //Salva imagem com a extensão tiff
  void salvaTiff(char*); //Salva imagem com a extensão tiff
  //com um nome específico
  void salvaBmp2(char*);//Salva imagem com a extensão bmp
  void display(Image*);

  void imageToClipBoard(); //função não utilizada no momento

  void drawPoint(unsigned int, unsigned int, double grayScale); //Desenha um ponto na coordenada
  //especificada
  void drawLine(unsigned int, unsigned int, unsigned int, unsigned int);
  //traça uma ilnha da cordenada do ponto inicial a cordenada do ponto final
  Image& drawLine(unsigned int, unsigned int, unsigned int, unsigned int, Image* image);
  //traça uma ilnha da cordenada do ponto inicial a cordenada do ponto final
  //em uma imagem passada como parâmetro
  void setTriggerStatus(bool status);
  Image& subtractImages(Image* image1, Image* image2);
};
#endif


/*
* Projeto RoboTurb/RoboFurnas
* Measurement.h
*
* Copyright (C) 2003 Labmetro
*
* Autor: Diego Carvalho dic@labmetro.ufsc.br
*        Luan Perdomo lpy@labmetro.ufsc.br
*        Roger Schipmann
*        Tiago Costa tlp@labmetro.ufsc.br
* Data: 27/06/2003
*
* Histórico:
*
*/

//---------------------------------------------------------------------------
#ifndef MeasurementH
#define MeasurementH
#include <vcl.h>
#include <controls.hpp>
#include "PointsXY.h"
#include "matriz.h"
#include "ExcecaoMatriz.h"
#include "Math.hpp"
#include "Rimage.h"

typedef unsigned char uInt8;

/**
* Realiza as medições apartir de uma imagem contendo as linhas laser.
*
* Esta classe é responsável por todo o processo de medição: carrega os
* parâmetros dos arquivos, aplica filtros às imagens, encontra os pontos das
* linhas laser, passa esses pontos para o sistema de coordenadas desejado e
* salva os pontos em arquivo.
*
* @author Diego Carvalho dic@labmetro.ufsc.br
*         Luan Perdomo lpy@labmetro.ufsc.br
*         Roger Schipmann
*         Tiago Costa tlp@labmetro.ufsc.br
* @date Junho, 2003
* @version 1.0
*
*/
class Measurement
{

  struct parameters
  {
```

```
    int n; // número de pontos adquiridos por linha laser.
    int f; // filtro utilizado.
    float dist; // distância entre as linhas laser no plano z = zero.
    float tgangle; // tangente do ângulo entre os planos de laser pelo usuário
    float ycoef[3][3]; // coeficientes da calibração y.
    float zcoef[3][3]; // coeficientes da calibração z.
  };

public:
  // TYPES

  /** @name LIFECYCLE */
  /**@{*/

  Measurement();

  /**
  * Construtor padrão.
  */
  Measurement(Rimage* rimage_);

  /**
  * Construtor de cópia.
  *
  * @param memo ponteiro para o objeto TMemo, que eh onde sao mostrados os
  * resultados das mediçoes
  */
  Measurement(TMemo* memo,Rimage* rimage_);

  /**
  * Destrutor.
  */
  ~Measurement();

  /**@}*/
  // End of LIFECYCLE


  /** @name OPERATORS */
  /**@{*/

  /**@}*/
  // End of OPERATORS


  /** @name OPERATIONS */
  /**@{*/

  /**
  * Mostra os pontos no sistema escolhido.
  *
  * Chama a função getPoints, que calcula os pontos no sistema escolhido.
  * Mostra esses pontos na interface atraves das funções print.
  *
  * @param pointsXY Ponteiro que guarda as coordenadas dos pontos medidos.
  *
  * @return 0 em caso de sucesso e 1 em caso de falha.
  *
  * @pre A função initialize deve ser executada antes desta.
  *
  */
  int execute(PointsXY*);

  /**
  * Salva os pontos em Pixels.
  *
  * Salva os pontos obtidos diretamente do CCD(em pixels) no arquivo escolhido.
  * Se o arquivo não exisir, será criado um novo. Caso exista, os dados
  * serão acrecsentados no final do arquivo.
  *
  * @param filename Array de no máximo 30 caracteres com o nome do arquivo.
  *
  * @return 0 em caso de sucesso e 1 em caso de falha.
  *
```

```
* @pre A função execute para este sistema de coordenadas deve ser executada
* antes desta.
*
*/
int savePixels(char filename[30]);

/**
* Salva os pontos em SCS.
*
* Salva os pontos no sistema de coordenadas do sensor no arquivo escolhido.
* Se o arquivo não exisir, será criado um novo. Caso exista, os dados
* serão acrecsentados no final do arquivo.
*
* @param filename Array de no máximo 30 caracteres com o nome do arquivo.
*
* @return 0 em caso de sucesso e 1 em caso de falha.
*
* @pre A função execute para este sistema de coordenadas deve ser executada
* antes desta.
*
*/
int saveSCS(char filename[30]);

/**
* Salva os pontos em SCI.
*
* Salva os pontos no sistema de coordenadas da interface no arquivo escolhido.
* Se o arquivo não exisir, será criado um novo. Caso exista, os dados
* serão acrecsentados no final do arquivo.
*
* @param filename Array de no máximo 30 caracteres com o nome do arquivo.
*
* @return 0 em caso de sucesso e 1 em caso de falha.
*
* @pre A função execute para este sistema de coordenadas deve ser executada
* antes desta.
*
*/
int saveSCI(char filename[30]);

/**
* Salva os pontos em SCR.
*
* Salva os pontos no sistema de coordenadas do robo no arquivo escolhido.
* Se o arquivo não exisir, será criado um novo. Caso exista, os dados
* serão acrecsentados no final do arquivo.
*
* @param filename Array de no máximo 30 caracteres com o nome do arquivo.
*
* @return 0 em caso de sucesso e 1 em caso de falha.
*
* @pre A função execute para este sistema de coordenadas deve ser executada
* antes desta.
*
*/
int saveSCR(char filename[30]);

/**@}*/
// End of OPERATIONS

/** @name ACCESS */
/**@{*/

/**
* Obtém o n´´umero de linhas a serem analizadas na imagem.
*
* @return param.n.
*
*/
int getNumPoints();

/**
* Seta o n´´umero de linhas a serem analizadas na imagem.
*
```

```
   * @param numPoints.
   *
   */
   void setNumPoints(int);

   /**
   * Seta a refer^^encia.
   *
   * Refer^^encia é o sitema de coordenas escolhido.
   *
   * @param reference_.
   *
   */
   void setReference(int);

   /**@}*/
   // End of ACCESS

   //!!!IMPORTANTE!!!
  void initializeDKMatrix(double* dKMatrix_);
   //inicializa a matriz de
   //cinemática direta com os dados do vetor passado como parâmetro

   void setDifMeassurement(bool difMeasurement_);
protected:

   bool difMeasurement;

   /** contém os paramâmetros de inicialização carregados dos arquivos */
   struct parameters param;

   /** Guardas as coordenadas x, y e z dos pontos: [linha lazer] [linha
   * analizada] [coordenada]. A primeira linha lazer equivale ao valor 0. A
   * linha analizada corresponde à linha horizontal do CCD que está  sendo
   * analizada. A coordenada é 0 para x, 1 para y e 3 paraz. */
   int Points3d[3][572][3];

   /** Struct auxiliar que contém as coordenadas dos pontos encontrados, bem
   * como seus respectivos valores de intensidade */
   PointsXY* pointsXY;

   /** indica em qual sistema de coordenadas serão calculados os pontos:
   * 1 - pixel; 2 - SCS -> Sistema de coordenadas do sensor; 3 - SCI -> Sistema
   * de coordenadas da interface; 4 - SCR -> Sistema de coordenadas do robo. */
   int reference;

   /** Guarda as informações das operações realizadas para mostrar no form */
   TMemo *memo1;

private:

   Rimage* rimage;

   /** Matriz que guarda os pontos no sistema de coordenadas do sensor. As
   * dimenções são 3.n x 4, onde n é o numero de pontos analizados.
   * Nas primeiras n linhas estão os pontos da primeira linha lazer e nas
   * ultimas n linhas ficam os pontos da terceira linha lazer. */
   Matriz<double>* pointsSCS;

   /** Matriz que guarda os pontos no sistema de coordenadas da interface. As
   * dimenções são 3.n x 4. */
   Matriz<double>* pointsSCI;

   /** Matriz que guarda os pontos no sistema de coordenadas do robo. As
   * dimenções são 3.n x 4. */
   Matriz<double>* pointsSCR;

   /** Matriz constante da transformação do sistema de coordenadas sensor
   * para o sistema de coordenadas interface */
   Matriz<double>* kMatrix;

   /** Matriz da transformação da cinemática direta
   *  transformação do sistema de coordenadas da interface para o sistema de
   *  coordenadas do robo*/
```

```
Matriz<double>* dKMatrix;

/** Matriz constante
* para transformar o sistema de cordenadas do sensor para a notação
* específica do robo*/
Matriz<double>* cMatrix;

/**
* Filtro padrão: Linha média
*
* Esta função faz uma média de cinco pontos na vertical. Analisa dois pontos
* acima e dois abaixo da posição em análise.
*
* @param y Linha do CCD a ser analizada.
*
* @param buffer Dados brutos do buffer da imagem a ser analizada.
*
* @param UsedData Array de saída que guarda os dados da linha do CCD em
* questão depois de filtrados.
*
*/
int MediumLine(int, uInt8*, uInt8*);

/**
* Filtro 2, média ponderada.
*
* Faz uma média horizontal de cinco pixels, sendo que os pixels mais próximos
* recebem  pesos maiores no cálculo da média. Este filtro não trata os dados
* adequadamente.
*
* @param y Linha do CCD a ser analizada.
*
* @param buffer Buffer da imagem a ser analizada.
*
* @param UsedData Array de saída que guarda os dados da linha do CCD em
* questão depois de filtrados.
*
*/
int Average(int, uInt8*, uInt8*);

/**
* Filtro 1, Dados reais, sem tratamento.
*
* @param y Linha do CCD a ser analizada.
*
* @param buffer Buffer da imagem a ser analizada.
*
* @param UsedData Array de saída que guarda os dados da linha do CCD em
* questão depois de filtrados.
*
*/
int RealData(int, uInt8*, uInt8*);

/**
* Encontra a posição das linhas laser.
*
* Enconra o pico das três linhas laser para a linha do CCD em questão.
*
* @param y Linha do CCD a ser analizada.
*
* @param UsedData Dados filtradeos da linha em questão.
*
* @param buffer Buffer da imagem a ser analizada.
*
* @param lines Array de saída que contém a posição dos três picos das linhas
* laser para a linha em questão.
*
*/
int LinesFinder(int, uInt8*, uInt8*, float lines[3]);

/**
* Encontra os ponots das linhas laser.
*
* Encontra as coordenadas dos pontos das três linhas laser no sistema de
```

```
   * coordenadas escolhido.
   *
   */
   int GetPoints();

   /**
   * Inicializa o processo de medição.
   *
   * Carrega apartir de arquivos as informações necessárias para realizar as
   * medições: features.dat contém os parâmetros; ycoef.dat contém os
   * coeficientes da cordenada y; zcoef.dat contém os coeficientes da cordenada
   * z; scstosci.dat contém a matriz de transformação do SCS para o SCI;
   * cSCS.dat contém a matriz de transformação para o SCS;
   *
   */
   int Initialize();

   /**
   * Insere linha na interface.
   *
   * Insere um bloco de texto no TMemo da interface apontado pelo atributo
   * *memo1. Este bloco de texto deve ser do tipo AnsiString e pode contér
   * caracteres de formatação como '/n' e '/t'.
   *
   * @param memoString Variável do tipo AnsiString que contém o texto a ser
   * inserido.
   *
   */
   void setTexto(AnsiString memoString);

   void setLinha(AnsiString lineString);

   /**
   * Mostra os pontos no SCS.
   *
   * Mostra os pontos da medição realizada no sistema de coordenadas do sensor,
   * no TMemo da interface.
   *
   */
   void printSensorPoints();

   /**
   * Mostra os pontos em pixels.
   *
   * Mostra os pontos do CCD (em pixels) da medição realizada, no TMemo da
   * interface.
   *
   */
   void printPixels();

   /**
   * Mostra os pontos no SCI.
   *
   * Mostra os pontos da medição realizada no sistema de coordenadas da
   * interface , no TMemo da interface.
   *
   */
   void printPointsSCI();

   /**
   * Mostra os pontos no SCR.
   *
   * Mostra os pontos da medição realizada no sistema de coordenadas do robo,
   * no TMemo da interface.
   *
   */
   void printPointsSCR();


};
#endif
```

```
//---------------------------------------------------------------------------
#ifndef RScanH
#define RScanH
#include "..\Measurement\Measurement.h"
#include "..\RImage\RImage.h"
#include "..\Points\PointsXY.h"
#include "..\Excecoes\ExcecaoCampoInvalido.h"
//---------------------------------------------------------------------------
#endif

//---------------------------------------------------------------------------

class RScan
{

private:
        Rimage* rimage;
        Measurement* measurement;
        bool freezeImage;
        PointsXY* pointsXY;
        int numPoints;
        int reference;

protected:



public:
       RScan();
       RScan(Measurement* measurement_);
       ~RScan();
       bool getCBlinhaAuxiliar();
       double getEintervaloLinhas();
       int getCBganhoDAQ();
       double getEconfig1();
       double getEconfig2();
       double getEconfig3();
       double getReferencia();
       int getEpontosLinha();

};
```

ANEXO 3 - API *WELD*

Tabela 4: **API** *Weld*

```
/*
 * Projeto RoboTurb/RoboFurnas
 * Rweld.h
 *
 * Copyright (C) 2003 UFSC/LACTEC
 *
 * Autor: Cristiano Blum Weingartner - blum@labsolda.ufsc.br
 * Data: 15/07/03
 *
 * Histórico:
 *
 *  11/05/04: Carlos Augusto Vieira e Vieira - vieira@das.ufsc.br
 *    - Trocadas o tipo das constantes para int para se adequar ao retorno das
 *      respectivas funções.
 *
 *  18/05/04: Carlos Augusto Vieira e Vieira - vieira@das.ufsc.br
 *    - Alterado o método Send() para se adequar à nova versão do firmware da
 *      DIGITEC, que retorna o mesmo byte enviado ao invés de uma constante.
 *    - Adicionado o método WriteAndCheck(Byte), que envia um byte e verifica a
 *      resposta da DIGITEC. A constante MAX_READ_RETRIES declarada no método
 *      limita a quantidade de tentativas de checagem da escrita.
 *
 *  14/07/04: Carlos Augusto Vieira e Vieira - vieira@das.ufsc.br
 *    - Introduzida a constante SERIAL_W2R_DELAY, atraso em milisegundos entre a
 *      escrita e subseqüente leitura.
 *    - WriteAndCheck() agora conta com o atraso supracitado.
 *
 * Notas:
 *  13/07/04: Carlos Augusto Vieira e Vieira - vieira@das.ufsc.br
 *  - Atualmente compatível com o 'firmware' versão GU3.C com CTR_30.C, ambos de
 *    03.05.04.
 *
 */

//---------------------------------------------------------------------------
#ifndef RWeldH
#define RWeldH
//---------------------------------------------------------------------------
// Constantes indicadoras de sucesso e erro das funções.
const int ReadError = -1;
const int WriteSuccess = 2;
const int WriteError = -2;
const int SerialInitiateError = -3;
const int SerialInitiateSuccess = 3;
// Atraso em ms para esperar o controlador responder à uma escrita (deve
// ser PAR).
const unsigned long SERIAL_W2R_DELAY = 100;

//---------------------------------------------------------------------------

#include "SerialFunctions.h"

//---------------------------------------------------------------------------

class RWeld
{
  protected:
    int Write(const char data);
    int Read();

    /**
     * Escreve no canal serial e verifica se a mensagem foi recebida
     * corretamente.
     *
     * @param Byte Dado a ser enviado (um byte apenas, segundo método Write).
     * @return Verdadeiro se a mensagem foi recebida corretamente.
     */
    bool WriteAndCheck(const char Byte, bool IgnoreSerialReturn = false);
```

```
  public:
    SerialPort Serial;

    int SerialInitiate(char *com, int Baud);
    int Send(int type, int cmd, unsigned int data);
    bool ConnectMachine(void);
    void CloseMachine(void);
};

//---------------------------------------------------------------------------
#endif

/*
 * Projeto RoboTurb/RoboFurnas
 * RWeldPlasma.h
 *
 * Copyright (C) 2004 UFSC/LACTEC
 *
 * Autor: Carlos Augusto Vieira e Vieira - vieira@das.ufsc.br
 * Data: 11/05/04
 *
 * Histórico:
 *
 *  12/05/04: Carlos Augusto Vieira e Vieira - vieira@das.ufsc.br
 *    - Definida estratégia para acesso de variáveis de solda e protótipos das
 *      funções necessárias para tal, bem como o enumerador VariableLabel.
 *
 */
//---------------------------------------------------------------------------
#ifndef RWeldPlasma_H
#define RWeldPlasma_H
//---------------------------------------------------------------------------

//LOCAL
#include "RWeld.h"
#include "DigitecCodes.h"
//---------------------------------------------------------------------------

enum VariableLabel
{
  IPulse, IBase, TPulse, TBase, VwBase, VwPulse,
  IOpen, IEnd, TStart, TEnd,
  TUp, TDown
};

class RWeldPlasma : protected RWeld
{
  protected:
    //Identificador de processo ativo.
    bool IsWelding;

    //Parâmetros da solda em unidades do SI.
    float Ip, Ib, Tp, Tb, Vwb, Vwp, Io, Ie, Ts, Te, Tu, Td;

    //Parâmatros convertidos para comunicação com a Digitec.
    unsigned int IpConv, IbConv, TpConv, TbConv, VwbConv, VwpConv;
    unsigned int IoConv, IeConv, TsConv, TeConv, TuConv, TdConv;

    //Incrementos e decrementos das correntes de pulso e de base, convertidos
    //para a Digitec e divididos em parte alta (H) e baixa (L).
    int IB_inc_L, IB_inc_H, IP_inc_L, IP_inc_H;
    int IB_dec_L, IB_dec_H, IP_dec_L, IP_dec_H;

    //Flag interna que indica se a tocha está ativa.
    char flag_soldando;

    // Retorna o ponteiro de determinada variável. Uso interno.
    float* GetFloatVarPointer(VariableLabel Variable);
    unsigned int* GetUnsignedVarPointer(VariableLabel Variable);
```

```
    //Converte a variável para valores da DIGITEC.
    unsigned int Convert(VariableLabel Variable, float Data);

  public:
    RWeldPlasma();

    unsigned int GetData(VariableLabel Variable);
    void Set(VariableLabel Variable, float NewData);
    float Get(VariableLabel Variable);
};

#endif  //RWeldPlasma_H
```

ANEXO 4 – API *TEACH PENDANT*

Tabela 5: **API** *Teach Pendant*

```
//---------------------------------------------------------------------------
#ifndef RTeachPendantH
#define RTeachPendantH
//---------------------------------------------------------------------------
#include "SerialFunctions.h"
#include "RDisplay.h"
#include "RKeyboard.h"
#include "RJoystick.h"
//---------------------------------------------------------------------------
// Edit definitions
#define BackSpace       '*'
#define Enter           '#'
#define Cursor          '_'
#define MaxLength       17
#define tpInteger       0       // Edit Type Integer (default)
#define tpFloat         1       // Edit Type Float
#define tpNoErase       0       // Edit Mode 1 (rename later)
#define tpErase         1       // Edit Mode 2 (rename later)

class RTeachPendant
{
        SerialPort *DisplayPort;
        SerialPort *JoystickPort;
        bool Ready;
        int LastError;
        int EditType;
        int EditMode;
public:
        RKeyboard *Keyboard;
        RDisplay *Display;
        RJoystick *Joystick;

        RTeachPendant();
        ~RTeachPendant();
        void SetDisplayPort( String Port );
        void SetJoystickPort( String Port );
        String GetDisplayPort( void );
        String GetJoystickPort( void );
        bool Start();
        bool Stop();
        bool IsReady();
        int GetLastError();
        void SetLastError( int Error );
        String GetErrorString();
        String GetErrorString( int Error );
        void SetEditMode( int Mode );
        void SetEditType( int Type );
        double EditNumber( double Inicial );
        void LoadParameters( void );
        void SaveParameters( void );
        double EditNumberNonBlocking( double Inicial, bool *Finished );
};
//---------------------------------------------------------------------------
#endif
```

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUS, J.S., et al.; 1989. **NASREM** - The NASA/NBS Standard Reference Model for Telerobot Control System Architecture. In: INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ROBOTS (20. : Oct. 1989 : Tokyo, Japan) *Proceedings.* Tokyo.

ALBUS, J.S.; QUINTERO, R.; LUMIA, R.; 1994. Overview of **NASREM**: The Nasa/NBS Standard Reference Model for Telerobot Control System Architecture. In: SPACE PROGRAMS AND TECHNOLOGIES CONFERENCE (20. : Set. 1994 : Huntsville, AL) *Proceedings.* Alabama.

ANDERSON, R. J.; 1993**. SMART**: A Modular Architecture for Robotic and Teleoperation. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (May 1993 : Atlanta, GA) ***IEEE*** *Proceedings.* Georgia. p. 416-421.

BAUER, G.; KREMER, M.; **ISW**; 2001. *Open Controller Enabled by Advanced Real-Time Network (OCEAN).* Germany.

BIRLA, S., et al.; 2001. *Reconfigurable Machine Controller Using the* ***OMAC API***. College International pour la Recherche en Productique (CIRP).

BIRLA, S., et al.; 1997**.** [Online] 1997. http://citeseer.ist.psu.edu/580004.html.

BONACORSO, Nelso Gauze; 2004. *Automatização dos Processos de Medição de Superfícies e de Deposição por Soldagem Visando a Recuperação de Rotores de Turbinas Hidráulicas de Grande Porte*. Florianópolis. Tese (Doutorado em Engenharia Mecânica) - Centro Tecnologico, Universidade Federal de Santa Catarina.

BRUYNINCK, H.; 2001. Open Robot Control Software the **Orocos** Project. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (2001) ***IEEE*** *Proceedings.* p. 2523-2528.

BUTTAZZO, G.; 1997. Robot Control in Hard Real-Time Enviroment. In: INTERNATIONAL WORKSHOP ON REAL-TIME COMPUTING SYSTEMS AND APPLICATIONS (4.: Oct. 1997 : Taipei, Taiwan) ***IEEE*** *Proceedings*. Taiwan. p. 152-159.

CHACONAS, K.; KELMAR, L.; M., NASHMAN; **NIST**; 1990. *A **NASREM** Implementation of Position Determination from Motion.* Gaithersburg, Maryland.

DEROBERTIS, L.; MILANO, N.; 1991. A Man Machine Interface for **NASREM**-based Telerobots. In: INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (Nov. 1991 : Osaka, Japan) ***IEEE*** *Proceedings*. Osaka. p. 994-998.

DIAZ-CALDERON, A.; PAREDIS, C. J. J.; KHOSLA, P. K.; 1999. On the Synthesis of the System Graph for 3D Mechanics. In: AMERICAN CONTROL CONFERENCE (18. : Jun. 1999) *Proceedings*. vol. 5, p. 3137-3141.

DIXON, K. R.; KHOSLA, P. K.; 2003. Programming Complex Robot Tasks by Prediction. In: INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (Oct. 2003) ***IEEE*** *Proceedings*. p. 3150-3155.

ESPRIT; 1996. *Open System Architecture for Controls within Automation Systems.* ESPRIT III Project 6379/9115 Final Report.

FIALA, J.C.; LUMIA, R.; ALBUS, J.S.; 1987. Servo Level Algorithms for the **NASREM** Telerobot Control System Architecture. In: SPACE STATION AUTOMATION (3. : Oct. 2003) ***SPIE*** *Proceedings*. v. 851, p. 103.

FIEDLER, P.; SCHLIB, C.; 1998. Open Architecture Robot Controllers and Workcell Integration. *Robotics Today*, vol. 11, no. 4, p.1-4.

FORD, W. E.; 1994. What is an Open Architecture Robot Controller? In: INTERNATIONAL SYMPOSIUM ON INTELLIGENT CONTROL (Aug. 1994) ***IEEE*** *Proceedings*. p. 27-32.

FUJITA, S.; YOSHIDA, T.; 1996. **OSE**: Open System Environment for Controller. In: INTERNATIONAL MACHINE TOOL ENGINEERS CONFERENCE (7. : Nov. 1991) *Proceedings*. p. 234-243.

HAYESROTH, F.; ERMAN, L. D.; TERRY, A. et al.; 1992a. Distributed Intelligent Control and Management: Concepts, Methods and Tools for Developing **DICAM** Application. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING (4. : Jun. 1992 : Capri, Italy) ***IEEE*** *Proceedings*. Capri. p. 235-244.

HAYESROTH, F.; ERMAN, L. D.; TERRY, A. et al.; 1992b. Domain-specific Software Architectures: Distributed Intelligent Control and Management. In: SYMPOSIUM ON COMPUTER-AIDED CONTROL SYSTEM DESIGN (Mar. 1992) ***IEEE*** *Proceedings*. p. 117-128.

HISSAM, A. S.; KLEIN, M.; **CMU**/Software Engineering Institute; 2004. *A Model Problem for an Open Robotics Controller*. Technical Report TN 030.

HONG, K. et al.; 2001. A PC Based Open Robot Control System: PC-ORC. In: INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS (Jun. 1992) ***IEEE*** *Proceedings*. vol. 3, p. 1901-1906.

IEEE; 1990. *Standard Glossary of Software Engineering Terminology*, 610.12.

———; 1995. *Standard Guide to the **POSIX** Open System Environment (**OSE**)*, 1003.0.

IEEE; *The Open Group*; 2004. *Standard for Information Technology - Portable Operating System Interface (**POSIX**) - Base Definition*, 1003.1.

ISAAK, J.; 2005. **POSIX** - Inside: A Case Study. In: INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (38. : Jan. 2005 : Hawaii) ***IEEE*** *Proceedings*. Hawaii. p. 203-213.

ISO; 2004. http://www.iso.org/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeStandardsListPage .TechnicalCo mmitteeStandardsList?COMMID=4289. [Online] 2004.

KAHMEN, A.; **WZL**; 2004. *Open Controller Enabled by Advanced Real-Time Network (OCEAN),* Germany

KATZ, R.; MIN, B.; PASEK, Z.; 2000.

KOREN, Y.; 2005. What is a Reconfigurable Manufacturing Systems (RMS)? *CIRP*, local de publicacao, no., vol., no. do fasciculo (data), pagina inicial-final.

KOREN, Y.; PASEK, Z.J.; ULSOY, A.G.; et al.; 1996. Real-time Open Control Architectures and System Performance. *CIRP Annals—Manuf. Technol.*, local de publicacao, no., vol. 45, no. 1, p. 377-380.

LEAHY, M.B. Jr.; PETROSKI, S.B.; 1994. Unified Telerobotic Architecture Project Program Overview. In: INTERNATIONAL CONFERENCE ON ADVANCED ROBOTIC SYSTEMS AND THE REAL WORLD (Set. 1994 : Munich, Germany). ***IEEE*** *Proceedings*. Vol. 1, p. 12-16.

LEAKE, S. et al.; 1989. Hierarchical Ada Robot Programming System (**HARPS**): A Complete and Working Telerobot Control System Based on the **NASREM**. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (May. 1994 : Scottsdale, AZ). ***IEEE*** *Proceedings*. Arizona. Vol. 2, p. 1022-1028.

LEAL, Rafael Della Giustina; 2005. *Impactos Sociais e Econômicos da Robotização: Estudo de Caso do Projeto Roboturb*. Florianópolis. Dissertação (Mestrado em Engenharia Elétrica) - Centro Tecnologico, Universidade Federal de Santa Catarina.

LUMIA, R.; FIALA, J.; 1990. The **NASREM** Robot Control System and Tested. *International Journal of Robotics & Automation*, vol. 5, no. 1, p. 20-26.

LUMIA, R.; 1989. **NASREM**: A Functional Architecture for Control of the Flight Telerobotic Servicer. In: EUROPEAN IN-ORBIT OPERATIONS TECHNOLOGY SYMPOSIUM (2. : Sep. 1989 : Toulouse, France) *Proceedings*. Vol. 297, 1989. p. 361-??.

———; 1994. Using **NASREM** for Real-Time Sensory Interactive Robot Control. *Robotica*, vol. 12, n. 2 (Mar.), p. 127-135.

———; 1994. Using **NASREM** for Telerobot Control System Development. *Robotica*, vol. 12, n. 6, p. 505-512.

LUTZ, P.; **OSACA**; 1998. *Course Material*.

LUTZ, P.; SPERLING, W.; 1997. **OSACA**: The Vendor Neutral Control Architecture In: EUROPEAN CONFERENCE ON INTEGRATION IN MANUFACTURING (Dec. 1997 : Dresden, Germany) *Proceedings*, p. 247-256.

MARTIN MARIETTA; 1992. *Draft Volume I of Next Generation Workstation/Machine Controller (NGC) Specification for an Open System Architecture Standard (**SOSAS**)*. Document No. NGC-0001-13-000-SYS.

MATSUMOTO, A. et al.; 2000. Can Industrial Robots Be Connected with each Other? In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED SYSTEMS (7. : Oct. 2000 : Iwate, Japan). ***IEEE** Proceedings*. Iwate. p. 493-498.

METTALA, E.; GRAHAM, M. H.; **CMU**/Software Engineering Institute; 1992. *The Domain-Specific Software Architecture*. Technical Report SR009.

MICHALOSKI, J. et al.; 1996. The TEAM **API** Open Architecture Methodology.

MILLER, D.J.; Lennox, R.C.; 1991. An Object-oriented Environment for Robot System Architectures. In: INTERNATIONAL CONFERENCE ON ROBOTIC AND AUTOMATION (2. : May. 1989 : Cincinnati, OH) ***IEEE** Proceedings*. Vol. 1, 1991. OHIO. p. 352-361.

NACSA, J.; 2001. Comparison of Three Different Open Architecture Controllers. *IFAC MIM*, Prague , Aug. 2001, p. 2-4.

NACSA, J.; HAIDEGGER, G.; 1997. Built-in Intelligent Control Applications of Open CNCs. In: WORLD CONGRESS ON INTELLIGENT MANUFACTURING PROCESSES AND SYSTEMS (2.: Jun. 10-13 1997: Budapest, Hungary) *Springer Proceedings*, p. 388- 392.

NAKAMURA, A. et al.; 1986. Controller for Industrial Robots. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION. ***IEEE*** *Proceedings.* Vol. 3, p. 254-259.

OMAC API Work Group. 2001. [Online] 2001. http://www.arcweb.com/omac/.

OMAC Users Group. 2002. *Functional Requirements.* 2002. Version 1.0.

OMG Object Management Group. 2001. *OMG Unified Modeling Language Specifications.* s.l. : http://www.omg.org, 2001. version 1.4.

OROCOS. 2001. [Online] 2001. [Cited: Agosto 1, 2007.] http://www.orocos.org.

PARK, J.; PASEK, Z.; KOREN, Y. et al; 1995. An Open Architecture Testbed for Real-time Monitoring and Control of Machining Processes. In: AMERICAN CONTROL CONFERENCE (Jun. 1995 : Seattle, WA) ***IEEE*** *Proceedings*. Vol. 1. Washington. p. 200-204.

PI, J.; RED, E.; JENSEN, G.; 1998. Grind-free Tool Path Generation for Five-axis Surface Machining. *Computer Integrated Manufacturing Systems,* v. 11, n. 4 (Oct.), p. 337-350.

PRITSCHOW, G.; et al.; 1993. Open System Controllers - A Challenge for the Future of the Machine Industry. *CIRP Annals*, v. 42, n. 1 p. 449-452.

PRITSCHOW, G.; 1997. Trendwende in der Steuerungstechnik: : Herstellerübergreifend offene Systeme (**HÜMNOS**). EMO, Hannover.

PRITSCHOW, G.; 2005. Self-Adapting Control Systems for **RMS**. In: INTERNATIONAL CONFERENCE ON RECONFIGURABLE MANUFACTURING (3. : May. 2005 : Ann Arbor, USA) ***CIRP*** *Annals*.

PROCTOR, F. et al.; 2000. Analysis of Behavioral Requirements for Component-based Machine Controllers. In: INTERNATIONAL SYMPOSIUM ON INTELLIGENT SYSTEMS AND ADVANCED MANUFACTURING (Nov. 2000 : Boston, MA) ***SPIE*** *Proceeding*. Vol. 4191. Massachusetts. p. 10-18.

RAPOSO, Emerson Pereira; 1997. *Desenvolvimento de Um Sistema de Reconhecimento de Voz para Emissão de Comandos Verbais para Robôs.* Florianópolis. Dissertação (Mestrado em Engenharia Elétrica) - Centro Tecnologico, Universidade Federal de Santa Catarina.

RAPOSO, E. P.; STEMMER, M. R.; 1998. Um Sistema de Reconhecimento de Fala Aplicado à Interação com um Robô. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA (CBA) (12.: Uberlandia, MG) *Anais.*

RAPOSO, E. P.; STEMMER, M. R.; ALTHOFF, G. F.; 2002. Modelagem de Sistemas através da UML. *Revista de Automação e Tecnologia da Informação,* Florianópolis, Centro de Tecnologia em Automação e Informática (CTAI).

RAPOSO, E. P.; STEMMER, M. R.; 2003. Modelo do Controle Hierárquico de um Robô para Soldagem em Turbinas de Grande Porte. *Revista de Automação e Tecnologia da Informação*, CTAI.

RAPOSO, E. P.; STEMMER, M. R.; MAYTAIOUL, R.; 2007. Um Controlador de Arquitetura Aberta Aplicado a um Robo Reis RV-15 para Avaliação de Processos de Medição e Soldagem Robotizada. In: SOCIEDADE BRASILEIRA DE AUTOMAÇÃO INDUSTRIAL (Florianópolis, SC) ***SBAI*** *Annals*.

RESSA, B.; COSOLI, P; MORI, A.; 1991. A **NASREM** Based Architecture for Robot Control in the Sanitary Landfill Study Case. In: INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS (15. : Jun.. 1991 : Pisa, Italy) ***IEEE*** *Proceedings*. Vol. 2, 1991. Pisa. p. 1299-1304.

SAWADA, C.; AKIRA, O.; 1997. Open Controller Architecture **OSEC**-11: Architecture Overview and Prototype Systems. In: INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION (6. : Set. 1997) ***IEEE*** *Proceeding*, p. 543-550.

SPERLING W.; LUTZ, P.; FISW; 1995. *Enabling Open Control Systems: An Introduction to the Osaca System Platform.* ESPRIT III Project. Stuttgart, GmbH.

SPERLING W.; LUTZ, P.; 1997a. Enabling Open Control Systems - An Introduction to the **OSACA** System Platform. In: PUBLISHED IN ROBOTICS AND MANUFACTURING ISRAM (May. 1997 : Montpellier, France) vol. 6.

SPERLING W.; LUTZ, P.; 1997b. Designing Applications for an **OSACA** Control. In: INTERNATIONAL MECHANICAL ENGINEERING CONGRESS AND EXPOSITION (Nov. 1997 : Dalles, USA) ***ASME*** *Proceeding*, p. 16-21.

STEWART, D.B.; VOLPE, R.A.; KHOSLA, P.K.; 1992. Integration of Real-Time Software Modules for Reconfigurable Sensor-Based Control Systems. In: INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (Jul. 1992) ***IEEE*** *Proceeding*, p. 325-332.

———; 1997. Design of Dynamically Reconfigurable Real-Time Software Using Port-based. ***IEEE*** *Transactions Software Engineering*, vol. 23, Issue 12 (Dec.), p. 759-776.

TATE, A; 2001. Task, I-X and I-N-C-A: An Architecture and Related Ontology for Mixed-initiative Synthesis. http://citeseer.ist.psu.edu/tate01ix.html. set

TERRY, A.; HAYESROTH, F.; ERMAN, L.; 1994. Overview of Teknowledge´s Domain-Specific Software Architecture Program. ***ACM SIGSOFT*** *Software Engineering Notes*, v. 19, Issue 4 (Oct.), p. 68-76.

UNECE. 2004. 2004.

WEINERT, G. F.; 2000. Java Based Open Architecture Controller. In: WORLD AUTOMATION CONFERENCE, MAUI, (Jun. 11-16).

WEISEL, W.; 1997. The State of the Art in Robot Control Solutions. Talking Advantage of the PC Platform . *Robotics World*.

WHEELER, T.; 1994. *O Manual dos Sistemas Abertos*. Editora Campus.

YANG, Z.; RED, E.; 1997. On-Line Cartesian Trajectory Control Of Mechanisms Along Complex Curves. *Robotica*, New York, no. 3, vol. 15, (May.), p. 263-274.

YEN, J.; ALDERSON, G.; BAILO, C.; CHRYSLER, FORD, GM; 1994. *Requirements of Open, Modular Architecture Controllers for Applications in the Automotive Industry.* USA.

YONGLIN, C.; 2005. An Evaluation Space for Open Architecture Controllers. *International Jornal Advanced Manufecturing Technology,* Springer London, vol. 26, no. 4 (Ago.), p. 351-358.