UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

DALTON MATSUO TAVARES

ACID: Arquitetura de Controle para Integração de Dispositivos

São Carlos

2010

FOLHA DE JULGAMENTO

Candidato(a): Bacharel **DALTON MATSUO TAVARES.**

Tese defendida e julgada em 03/12/2010 perante a Comissão Julgadora:

PARTICIPAÇÃO POR VIDEOCONFERÊNCIA

Prof. Associado **GLAUCO AUGUSTO DE PAULA CAURIN** - (Orientador)
(Escola de Engenharia de São Carlos/USP)

APROVADO

Prof. Dr. **ADRIANO ALMEIDA GONÇALVES SIQUEIRA**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Prof. Dr. **PAULO MARCOS DE AGUIAR**
(Instituto Federal de São Paulo/IFSP)

APROVADO

Prof. Associado **WALTER FETTER LAGES**
(Universidade Federal do Rio Grande do Sul/UFRGS)

APROVADO

Prof. Dr. **DENIS FERNANDO WOLF**
(Instituto de Ciências Matemáticas e de Computação/ICMC)

Prof. Associado **JONAS DE CARVALHO**
Coordenador do Programa de Pós-Graduação em
Engenharia Mecânica

Prof. Associado **PAULO CÉSAR LIMA SEGANTINE**
Presidente da Comissão da Pós-Graduação da EESC

DALTON MATSUO TAVARES

ACID: Arquitetura de Controle para Integração de Dispositivos

Tese apresentada a Escola de Engenharia de São Carlos – EESC-USP, como parte dos requisitos para obtenção do título de Doutor em Engenharia Mecânica.

Área de Concentração: Dinâmica das Máquinas e Sistemas

Orientador: Prof. Dr. Glauco Augusto de Paula Caurin

São Carlos

2010

Dedicatória

“Dedicado a este trabalho a meus pais. Sem eles, nada disso seria possível”.

Agradecimentos

A Deus por ter me dado a curiosidade. Sem esta, não teria insistido tanto em aprender.

A meus pais, pelos direcionamentos na vida. Graças a eles, escolhi o caminho que trilhei.

A minha irmã por ter me incentivado sempre a continuar na luta, apesar de estarmos distantes.

A minha noiva, Stella Jacyszyn Bachega, pela companhia e pela paciência. Ela tem sido uma fonte de felicidades constantes em minha vida.

A meu orientador por ter me acompanhado em todos os passos dessa jornada árdua que foi meu doutorado. Agradeço sua paciência quando sofria para entender o sentido de outras áreas do conhecimento que não a Ciência de Computação.

A meus amigos José Martins Júnior, Rafael Vidal Aroca e aos companheiros do laboratório que em algum momento me auxiliaram com tanta paciência no decorrer deste processo.

A Roberto Gómez Bolaños, Ramón Gómez Valdez y Castilo, Carlos Villagrán "Pirolo" Eslava, Maria Antonieta de las Nieve Gómez Rodrigez, Florinda Meza García, Edgar Vivar, Rubén Aguirre Fuentes, Angelines Fernández Abad, Raul "Chato" Padilla e Horacio Gómez Bolaños pela inspiração nos momentos difíceis. Graças a eles consegui perseverar quando tudo mais parecia falhar.

Epígrafe

"Como dizia meu velho avô: Se quiser vir a ser alguma coisa, que devore os livros!".

Ramón Gómez Valdez y Castilo

Resumo

TAVARES, D. M. ACID: Arquitetura de Controle para Integração de Dispositivos. 2010. 164 f. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Paulo, 2010.

O objetivo desta tese é adaptar um modelo de integração industrial baseado no protocolo de comunicação XML Interface for Robots and Peripherals (XIRP), de modo que este pudesse ser aplicado à realidade de pequenas e médias empresas brasileiras (PMEs).

Para que isto fosse possível, é necessário estudar os ambientes industriais existentes e a complexidade envolvida na tarefa de integração de dispositivos para os mesmos. Observa-se que estes ambientes são orientados a fabricante com relação às soluções criadas em termos de equipamentos e softwares. Este é o exemplo de grandes empresas como ABB, KUKA, Reis Robotics etc. O fabricante detém a solução de integração e os dispositivos, e estes só podem ser integrados em plataformas deste fabricante.

O protocolo de comunicação industrial, no caso o protocolo XIRP, possibilita a comunicação entre dispositivos e sistemas controladores de maneira padronizada, permitindo que dispositivos de um fabricante se comuniquem com sistemas de outros fabricantes. Este protocolo é baseado na linguagem eXtensible Markup Language (XML) e fornece a especificação dos arquivos de esquema XML (XML schema), os quais permitem a criação de uma descrição de dispositivos e a composição de mensagens a serem trocadas entre eles. Para tanto, foi necessário criar a partir do padrão, toda a infra-estrutura de comunicação e sistema de integração de dispositivos.

Vale observar que o padrão XIRP foi criado levando em consideração dispositivos microprocessados. Como este padrão ainda está em desenvolvimento, e em caso de uma eventual adaptação a ele, seria impraticável modificar inteiramente um ambiente de produção por meio de substituição de equipamentos. Desta forma, este projeto de pesquisa apresenta uma infra-estrutura que consista em um sistema integrador para a adaptação gradativa a este protocolo de comunicação.

O sistema integrador proposto permite a incorporação de dispositivos XIRP ou não-XIRP. Considerando que um dispositivo já troque mensagens compatíveis com o XIRP, a sua integração e operação em um ambiente de produção

seria mais simples. Todavia, mesmo neste caso, o dispositivo ainda precisaria de suporte para trabalhar em uníssono com as demais entidades pertencentes ao ambiente. Esta sinergia é incluída na especificação do ambiente integrador, porém, não faz parte dos resultados pretendidos para a presente pesquisa. Desta forma, a hipótese prevista trata os mecanismos necessários para a incorporação de dispositivos não-XIRP em um ambiente de produção compatível com o padrão XIRP, desde o processo de indexação/identificação do dispositivo à geração automática da aplicação de controle e integração em baixo nível, de modo a permitir a operação do mesmo.

Palavras-chave: Plug' n' Produce, XIRP, XML, Orocos, Sistema integrador.

Abstract

TAVARES, D. M. ACID: Control Architecture for Device Integration. 2010. 164 f. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Paulo, 2010.

The goal of this research is to adapt an industrial integration model based on the XML Interface for Robots and Peripherals (XIRP) communication protocol, such that it could be applied to the reality of Brazilian small and medium enterprises (SMEs).

In order to enable this, it is necessary to study the existing industrial environments and the complexity involved in the device integration task. We observed these environments are manufacture centered considering the solutions created in terms of equipments and softwares. This is the example set by manufacturers like ABB, KUKA, Reis Robotics etc. The manufacturer detains the integration solution and the devices, and these can only be integrated in platforms of the same manufacturer.

The industrial communication protocol, in this case XIRP, enables the communication among devices and controller systems in a standardized way, allowing devices from one manufacturer to communicate with systems from another. This protocol is based on the eXtensible Markup Language (XML) and supplies the specification of the XML schema files, which allows the creation of a device description and the composition of messages to be exchanged among devices. Therefore a communication infrastructure and device integration system was built from the standard.

Note that the XIRP standard was created considering processor based devices. As this standard is still in development, and in case of an eventual adaptation to it, it would be impracticable to change an entire production environment by means of equipment exchange. Therefore, this research project presents an infrastructure that consists in an integrator system to gradually adapt to this communication protocol.

The proposed integrator system allows the incorporation of XIRP or non-XIRP devices. Considering a device already exchanges messages by means of XIRP compliant messages, its integration and operation would be simpler in a production environment. However, even in this case, the device would still need support to operate in unison with other entities pertaining to the

environment. This synergy is included in the specification of the integrator environment, although, it is not part of the intended results for this research. Therefore, the foreseen hypothesis deals with the necessary mechanisms to incorporate non-XIRP devices in a production environment compliant with the XIRP standard, from the device indexing/identification to the automatic generation of the control application and its low level integration, in order to allow its operation.

Keywords: Plug' n' Produce, XIRP, XML, Orocos, System integrator.

Lista de Figuras

1.1 Arquitetura de Controle para Integração de Dispositivos (ACID). . 4

2.1 Modelo de Naumann para uma arquitetura P'n'P [1] p. 287. . . . 14
2.2 Sistema de programação tradicional vs. sistema de programação orientado a processo [1] p. 288. 16
2.3 Diagrama de fluxo do módulo interconector [1] p. 289. 17
2.4 Parte da ontologia do robô [1] p. 291. 19
2.5 Holarquia e as camadas de gerenciamento e execução [2] p. 291. 20

3.1 Diferentes módulos de comunicação para diferentes fabricantes [3] p. 2. 27
3.2 Redução de esforços de comunicação por meio de padronização [3] p. 3. 29
3.3 Camadas do modelo de referência ISO/OSI [3] p. 3. 31
3.4 Estrutura básica de uma mensagem XIRP [3] p. 9. 35
3.5 Canais XIRP. 36
3.6 Estrutura da descrição de dispositivo XIRP [3] p. 6. 39
3.7 Instruções de programação configuradas para a aplicação em um sistema ABB. [3] p. 12. 42
3.8 Fluxo de dado para um XML recebido. [3] p. 16. 43
3.9 O mecanismo P'n'P proposto por [4] p. 5. 45

4.1 Arquitetura de sistema proposta para migração/adaptação ao protocolo de comunicação XIRP. 53
4.2 Diagrama UML de caso de uso para o sistema de marcadores proposto. 56
4.3 Dispositivo escolhido para a implementação do *scanner*. 59
4.4 Gerador de QR Code do projeto ZXing. 61

4.5 Sistema para gerenciamento de *tags*. 63
4.6 Software *Barcode Scanner* usado para implementar o *handset* para reconhecimento de etiquetas QR Code. (a) Disponibilidade do software no Android Market. (b) Software instalado. (c) Software em funcionamento. 66
4.7 Demonstração de reconhecimento de código de barra. 67
4.8 Especificação do *proxy* de dispositivo. 70
4.9 Conceito de adaptador *servlet* conforme apresentado em [5]. . . . 75
4.10 Controlador de célula robótica. 86
4.11 Representação do ambiente experimental de desenvolvimento para controle do IBM 7545 SCARA [6]. 94
4.12 Interface `KTaskBrowser` para os componentes implementados até o momento. 108

Lista de Listagens

4.1 Sintaxe do formato de mensagem `MECARD` proposto pela NTT DoCoMo. 61

4.2 Trecho de um esquema XML exemplo incorrendo em erro. 81

4.3 Comando XML básico gerado usando o Apache XMLBeans. 82

4.4 Comando XML básico de acordo com a especificação XIRP. 82

4.5 Trecho do arquivo de cabeçalho `lxrt_user.h` com os mapeamentos de métodos para as funções do espaço de *kernel*. 100

4.6 Trecho do `lxrt_kernel.c` com os mapeamentos das funções do espaço de *kernel*. 101

4.7 Exemplo de arquivo para a carga de propriedades para o PID da primeira junta (próxima a base) do robô IBM 7545 SCARA. . . . 103

4.8 Trecho do `scara_sv.cpp` com os mapeamentos da função para o PID1 e estabelecimento do fluxo de dados com a tarefa de supervisão. 105

4.9 Trecho do `scara_sv.cpp` contendo o procedimento para disponibilização de funções do PID1 em rede via CORBA. 106

4.10 *Shell* script para chamar o serviço de nomes e o programa servidor no PC industrial. 107

4.11 Criação de variável de ambiente no cliente para acesso às tarefas disponibilizadas pelo Orocos. 107

4.12 Trecho da árvore de processo gerada pelo `pstree`. 110

4.13 Trecho de lista de processo gerado pelo comando `ps aux`. 111

Lista de Abreviaturas

ACE — Adaptive Communication Environment
ACID — Arquitetura de Controle para Integração de Dispositivos
AIM — Automatic Identification Manufacturers
API — Application Programming Interface
ARIKT — Adaptive Roboter-Inspektion Komplexer Teile
ARS — Autonomous Reactive Robot Planning
BFL — Orocos Bayesian Filtering Library
BMBF — Bundesministerium für Bildung und Forschung
CAN — Control Area Network
comedi — Linux Control and Measurement Device Interface
CORBA — Common Object Request Broker Architecture
CPU — Central Processing Unit
DAC — Digital to Analog Converter
DSSP — Microsoft Decentralized Service Structure Protocol
EAN — European Article Number
FIFO — First In First Out
FP Code — Fine Picture Code
GenoM — Generator of modules
GUI — Graphical User Interface
HAL — Hardware Abstraction Layer
HMI — Human Machine Interface
HTTP — HyperText Transfer Protocol
IDE — Integrated Drive Electronics
IDL — Interface Definition Language
IEC — International Electrotechnical Commission
IIOP — Internet Inter-ORB Protocol

I/O — Input/Output
IPack — Industry Pack
IP — Internet Protocol
IPO — InterPOlação
ISO — International Organization for Standardization
JEIDA — Japanese Electronic Industry Development Association
JIS — Japanese Industrial Standards
JOP — Japan FA Open Systems Promotion Group
KDE — K Desktop Environment
KDL — Kinematics and Dynamics Lybrary
KRL — KUKA Robot Language
KRC — KUKA Robot Controller
LKM — Linux Kernel Modules
LXRT — Linux Real Time
MMS — Multimedia Message Service
OAC — Open Architecture Control
OCL — Orocos Component Library
OMAC — Open Modular Architecture Controller
ORB — Object Request Broker
ORC — Open Robot Control
OSACA — Open System Architecture for Controls within Automation Systems
OSEC — Open System Environment for Controllers
OSI — Open Systems Interconnection
OSS — Open Source Software
Orocos — Open Robot Control Software
OSS — Open Source Software
OWL — Web Ontology Language
OWL-DL — Web Ontology Language Description Logic
PACTI — Plano de Ação Ciência, Tecnologia e Inovação
PC — Personal Computer
PDA — Personal Digital Assistant
PDF — Portable Data File
PID — Proporcional - Integral - Derivativo
PME — Pequenas e Médias Empresas
P'n'P — Plug' n' Produce
QR Code — Quick Response Code
RFID — Radio-Frequency IDentification
RSI — Robot Sensor Interface

RTAI — Real Time Application Interface
RTL — Real Time Linux
RTT — Real Time Toolkit
SCARA — Selective Compliant Assembly Robot Arm ou Selective Compliant Articulated Robot Arm
SCXML — State Chart XML
SERCOS — Serial Real Time Communications System
SMS — Short Message Service
SO — Sistema Operacional
TAO — The ACE ORB
TCP — Transmission Control Protocol
UDP — User Datagram Protocol
UHF — Ultra High Frequency
UPC — Universal Product Code
URL — Uniform Resource Locator
VDMA — Verband Deutscher Maschinen und Anlagenbau
XIRP — XML Interface for Robots and Peripherals
XML — eXtended Markup Language
ZXing — Zebra Crossing

Sumário

1 INTRODUÇÃO **1**

1.1 OBJETIVOS 3

1.2 ESTRUTURA DA TESE 6

2 PLUG AND PRODUCE **9**

2.1 CONSIDERAÇÕES INICIAIS 9

2.2 PRÉ-REQUISITOS PARA UM SISTEMA PLUG AND PRODUCE . . 11

2.3 ARQUITETURA DE CONTROLE PLUG AND PRODUCE 12

2.3.1 Modelo de Naumann 13

2.3.2 Modelo de Arai 19

2.4 CONSIDERAÇÕES FINAIS 21

3 P'N'P E XIRP **25**

3.1 CONSIDERAÇÕES INICIAIS 25

3.2 OBJETIVOS 28

3.3 INTERFACE DE TRANSMISSÃO DE DADOS PARA ROBÔS E SENSORES 30

3.4 O PROTOCOLO XIRP 32

3.4.1 Arquivo de descrição de dispositivo 37
3.5 APLICAÇÕES 38
3.5.1 SMErobot™ 38
3.5.1.1 ABB 41
3.5.1.2 Reis Robotics 41
3.5.1.3 KUKA 42
3.5.1.4 XIRP+ 44
3.5.2 SISU 2010 46
3.6 CONSIDERAÇÕES FINAIS 48

4 ARQUITETURA DE CONTROLE PARA INTEGRAÇÃO DE DISPOSITIVOS (ACID) 50

4.1 CONSIDERAÇÕES INICIAIS 50
4.2 ARQUITETURA DO SISTEMA 51
4.3 SUBSISTEMA DE TAGGING 54
4.3.1 Arquitetura do subsistema de *tagging* 55
4.3.1.1 Implementação do *handset* 58
4.3.1.2 Composição das etiquetas 60
4.3.1.3 Identificação das etiquetas 64
4.4 SUBSISTEMA DE PROXY DE DISPOSITIVO 67
4.4.1 Arquitetura do subsistema proxy de dispositivo 69
4.4.2 Adaptadores servlet 74
4.4.3 XML data binding 76
4.5 SUBSISTEMA CONTROLADOR DE CÉLULA ROBÓTICA 83
4.5.1 Arquitetura do subsistema controlador de célula robótica . 83
4.5.2 Níveis de abertura 87
4.5.3 Plataformas OAC 88

4.5.3.1 KUKA robot sensor interface 89
4.5.3.2 Open robot control architecture e autonomous reactive robot planning 89
4.5.3.3 Generator of modules – GenoM 90
4.5.3.4 Projeto Orocos 91
4.5.4 Cenário de aplicação 91
4.5.4.1 *Drivers* do *kernel* 93
4.5.4.2 Comunicação entre módulos de *kernel* 95
4.5.4.3 *Drivers* de dispositivo baseados em LKM e acesso em tempo real 96
4.5.4.4 Interface de espaço de usuário via LXRT 99
4.5.4.5 Comunicação em rede 102
4.5.5 Limitações atuais 108
4.6 CONSIDERAÇÕES FINAIS 111

5 CONCLUSÃO 114
5.1 CONSIDERAÇÕES INICIAIS 114
5.2 RESULTADOS OBTIDOS 115
5.3 TRABALHOS FUTUROS 116
5.4 CONSIDERAÇÕES FINAIS 120

Referências 123

Glossário 131

CAPÍTULO

1

INTRODUÇÃO

Para pequenas e médias empresas (PMEs) assim como ambientes de pesquisa, onde as tarefas são tipicamente realizadas com tamanhos de lotes pequenos ou médios, a adaptabilidade rápida de robôs e da célula robótica[1] circundante, a novos produtos e processos, é mais importante do que manter os tempos de ciclo pequenos. Para que isso se torne possível, a programação de robôs e a integração de novos dispositivos em células robóticas precisa ser mais fácil do que nos sistemas atuais (e.g. especificidades de linguagem de programação, programação ponto-a-ponto usando *teach pendant* etc) [8].

A União Européia tem dado um forte enfoque no desenvolvimento de tecnologias com o intuito de viabilizar o crescimento de PMEs, tendo em vista a participação crescente destas na economia mundial. Isto pode ser observado em projetos de larga escala como o projeto SMErobot e o projeto SISU 2010 (estes projetos serão discutidos no capítulo 3).

[1]No contexto deste doutorado, uma célula robótica (*robotic workcell*) representa um sistema de dispositivos mecânicos ou computacionais, os quais incluem um ou mais robôs com máquinas associadas, equipamento de processamento ou suporte, os quais desempenham automaticamente tarefas de manufatura ou similar [7].

Considerando o cenário brasileiro, de acordo com números fornecidos no Plano de Ação Ciência, Tecnologia e Inovação (PACTI) [9]:

- 30% das empresas industriais realizam algum tipo de inovação tecnológica (a média européia é de 50%);
- Menos de 4% das empresas lançam produto novo no mercado;
- Menos de 3% introduzem processo novo no mercado.

Considerando a inovação para adequação a padrões internacionais:

- 23% das empresas inovam e diferenciam produtos;
- 13% das empresas especializadas em produtos padronizados realizaram inovação.

Considerando as estatísticas apresentadas, pode-se inferir que o campo de pesquisa e desenvolvimento em empresas industriais não é muito promissor. Entretanto, observasse que esta política de desenvolvimento rígida e pouco voltada ao desenvolvimento de novas tecnologias e inovação é uma realidade de grandes empresas. PMEs brasileiras, são considerados ambientes promissores para inovação e são elas que constituem as estatísticas apresentadas para empresas industriais que realizam algum tipo de inovação. Esse fato sugere que PMEs são potenciais centros para a inovação tecnológica, possuindo uma agilidade muito maior no desenvolvimento, emprego e adoção de novos métodos e tecnologias, quando comparadas a grandes empresas, dotadas de estruturas consolidadas e de certa forma estanques [10].

A motivação para este doutorado refere-se a este cenário, considerado promissor para a implantação de tecnologia aliada a novos padrões, com o intuito de gerar novos produtos e processos. Dessa forma, a infra-estrutura de integração de dispositivos inteligentes denominada XML-*based communication protocol for industrial robots and processor aided peripherals*, ou ainda,

XML *Interface for Robots and Peripherals* (XIRP) [11] aliada ao conceito de *Plug' n' Produce* ou P'n'P (uma espécie de *Plug and Play* para dispositivos em ambiente industrial - maiores informações no capítulo 2) são excelentes pontos de partida para uma infra-estrutura voltada ao desenvolvimento de PMEs brasileiras.

Em sua proposta original, seria difícil aplicar o conceito de P'n'P [3, 1, 8] aliado ao protocolo XIRP em PMEs brasileiras. Isso ocorre pois ambos pressupõem o uso de dispositivos microprocessados capazes de lidar com um protocolo de comunicação não trivial de alto nível. Considerando as restrições orçamentárias de PMEs brasileiras, seria inviável propor uma mudança radical na infra-estrutura destas, com o objetivo de adaptar a linha de produção para que esta utilize um sistema robótico adaptável baseado em substituição completa de sensores/dispositivos. A contribuição científica pretendida para este trabalho diz respeito ao estudo do modelo P'n'P e posterior adequação deste a realidade de PMEs brasileiras.

1.1 OBJETIVOS

O principal objetivo deste doutorado é criar um modelo arquitetural para a integração de dispositivos, que permita a adequação de uma PME, de maneira gradual, sem onerar custos proibitivos relativos a aquisição de equipamentos específicos para a operação em uma planta compatível com P'n'P. Para tanto, será usado um protocolo de comunicação industrial para a especificação de dispositivos e troca de mensagens, aliado a um *framework* de código aberto, cujo intuito é padronizar o acesso em nível mais baixo, considerando o desenvolvimento de controladores de dispositivos. Além disso, um sistema para indexação rápida de dispositivos será necessário, considerando o uso de dispositivos não compatíveis com o padrão, integrados a uma célula robótica por meio de alguma interface de baixo nível (como uma placa

de aquisição de dados). No contexto deste sistema de indexação, serão dados os primeiros passos para a construção de uma base de dados contendo os dispositivos de montagem. A figura 1.1 mostra a proposta de arquitetura pretendida, denominada Arquitetura para Controle e Integração de Dispositivos (ACID).

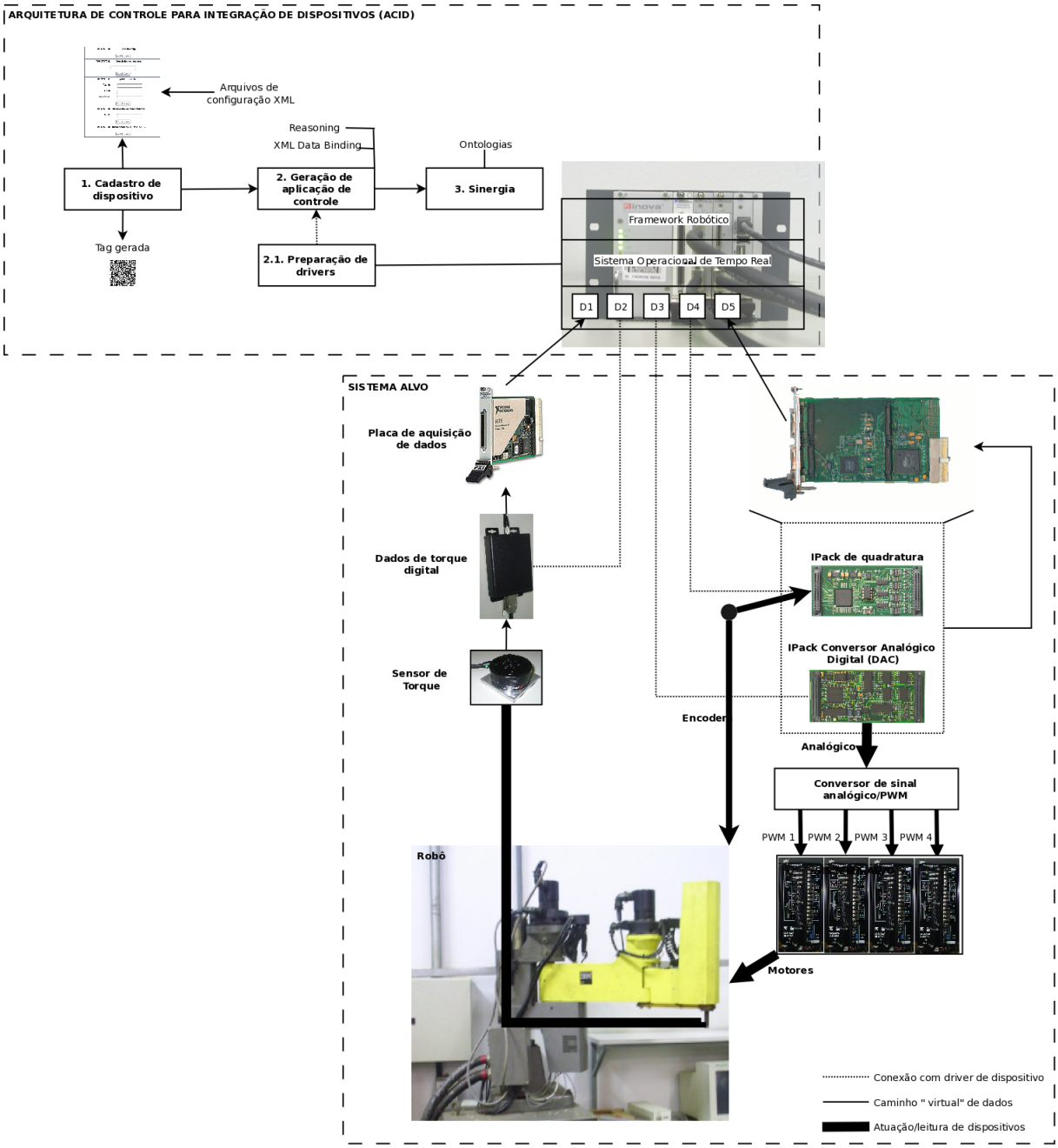

Figura 1.1: Arquitetura de Controle para Integração de Dispositivos (ACID).

A base para o desenvolvimento desta arquitetura é um protocolo para comunicação de dispositivos microprocessados de alto nível denominado XML *Interface for Robots and Peripherals* (XIRP). Esta é uma especificação definida pela *Verband Deutscher Maschinen und Anlagenbau* (VDMA), com índice de referência VDMA 66430-1 [11]. Esta especificação será analisada no capítulo 3 e modificações serão sugeridas com o intuito de integração de dispositivos microprocessados ou não, fato que deve permitir uma maior cobertura do padrão.

A proposta de arquitetura apresentada na figura 1.1 propõe a utilização de quatro subsistemas: um subsistema para o cadastro de dispositivos, um subsistema para a geração da aplicação de controle, um subsistema para a preparação dos controladores de dispositivo (*drivers*) e um subsistema para a criação da sinergia entre dispositivos.

O subsistema de cadastro de dispositivos, tem como objetivo receber os arquivos de configuração XIRP para um dispositivo, criando assim, uma base de dados de dispositivos conectados ao sistema. Este subsistema também deverá gerar uma etiqueta para acesso rápido a informações (e.g. informação de contato para um determinado dispositivo, endereço para o controlador de dispositivo disponível em rede etc).

Um subsistema para geração de uma aplicação de controle para cada dispositivo, deverá ser criado. Este subsistema usará como base os arquivos de configuração cadastrados no subsistema de cadastro de dispositivos. A geração de uma aplicação é algo mais complexo do que apenas utilizar as configurações (i.e. parâmetros e comandos) disponíveis para um determinado dispositivo. A geração de métodos para operação do dispositivo pode ser atingida por meio de XML *data binding*. Por outro lado, este subsistema deverá ir além da especificação original do XIRP, com o intuito de gerar a lógica por trás do controle de cada dispositivo. Considerando que cada dispositivo precisará ser controlado, os mecanismos de acesso a suas funções podem ser

obtidos por meio dos *drivers* disponíveis. Para o ambiente de testes criado, foi considerado o uso de uma plataforma aberta e um *framework* de controle robótico, também aberto, com *drivers* experimentais, criados para placas de aquisição de dados e dispositivos. Dessa forma, foi possível realizar o controle de um robô IBM 7545 SCARA conforme visto na figura 1.1.

O subsistema para a criação de sinergia entre os elementos da célula robótica deverá criar uma ontologia representativa dos dispositivos existentes de modo a simbolizar a correlação entre os mesmos.

Muito embora tenham sido estudados diversos sistemas P'n'P (capítulo 2) e a norma XIRP (capítulo 3), em especial os conceitos introduzidos por Naumann em [1, 8] e aprofundados por Verl e Naumann [12], a arquitetura proposta é única tendo em vista as limitações superadas para o cenário de aplicação previsto, no caso, PMEs brasileiras, e as próprias limitações da norma que define o XIRP (esta ainda representa apenas as disposições gerais do que virá a ser o padrão).

Como prova de conceito apresenta-se a arquitetura do sistema para a especificação de uma infra-estrutura de comunicação para a integração de dispositivos e os resultados experimentais obtidos (capítulo 4).

1.2 ESTRUTURA DA TESE

Os demais capítulos desta tese são brevemente resumidos a seguir:

- O capítulo 2 introduz o conceito de *Plug and Produce* contextualizando-o com os objetivos deste trabalho.

- O capítulo 3 apresenta a norma XIRP de forma resumida e sua importância no contexto de ambientes *Plug and Produce*. Além disso, são discutidas diversas iniciativas de fabricantes em direção a um padrão de

comunicação industrial comum com o objetivo de integração entre dispositivos.

- O capítulo 4 apresenta a especificação do sistema integrador. Neste contexto, serão apresentados o subsistema para o cadastro de dispositivos (denominado subsistema para geração de *tags*) e o subsistema para a geração da aplicação de controle e preparação de *drivers* de dispositivos (parcialmente implementado como sendo o subsistema de *proxy* de dispositivo e o subsistema controlador de célula de trabalho). O subsistema para a criação da sinergia entre dispositivos não foi implementado no contexto deste doutorado.

- O capítulo 5 apresenta os resultados obtidos com o ambiente experimental e as atividades futuras previstas para a continuidade deste trabalho.

CAPÍTULO

2

PLUG AND PRODUCE

2.1 CONSIDERAÇÕES INICIAIS

Considere um sistema de montagem que precisa mudar frequentemente sua configuração de dispositivos como no caso de inserção de máquinas, esteiras, alimentadores de peças, sensores, atuadores etc. A reconfiguração do sistema é necessária em situações como a quebra ou substituição de um dispositivo, a mudança de um produto etc. Existem três tipos de flexibilidade necessárias para esse tipo de sistema [2]:

1. flexibilidade para mudança geométrica;
2. flexibilidade para mudança sequencial;
3. flexibilidade para mudança na quantidade de produção.

O tipo 1 é necessário considerando mudanças no tamanho e forma dos produtos. A introdução de manipuladores industriais aumentou consideravelmente essa flexibilidade, embora garras e alimentadores de peça possuam

pouca flexibilidade. Nesse contexto, Arai [2] sugere o uso de manipulação sem fixação (*graspless*) como "*pushing, tumbling* e *pivoting*".

Para o tipo 2, foram introduzidos multi-agentes para acomodar a mudança frequente de ordens. Mudanças de montagem são computadas em tempo real, na linha de montagem. Muitos pesquisadores perceberam que o tipo 3, em realidade é muito difícil de se alcançar. A reconfiguração de sistemas de montagem parece uma questão chave para o tipo 3. Considerando uma linha de montagem flexível, como em PMEs e laboratórios de pesquisa, enfoca-se o primeiro caso.

Quando um dispositivo de montagem é adicionado ou removido, o procedimento de reconfiguração precisa de um tempo de ajuste inicial maior do que em sistemas tradicionais. Isso ocorre em parte, porque um dispositivo pode afetar o sistema de controle como um todo.

Para diminuir este tempo de ajuste inicial, a reconfiguração do sistema inteiro deve ser separada daquela do dispositivo. A interface dos dispositivos sobre o sistema de controle deve ser estabelecida tão facilmente quanto possível. Este conceito é chamado de *Plug and Produce*, o qual é similar ao conceito de *Plug and Play* em computadores pessoais.

Nesta analogia, para a instalação de dispositivos *Plug and Play*, como por exemplo uma impressora em um PC, bastaria conectar o dispositivo. Toda a configuração do dispositivo é realizada automaticamente e a aplicação oferece o serviço "imprimir" ao usuário. Portando este conceito para um ambiente de produção, isto significa que, por exemplo, ao conectar um robô a um controlador de célula robótica, ele ofereceria o comando "`move_to`" em uma interface humano-máquina (*Human Machine Interface* ou HMI). Ainda mais avançado, isto poderia significar que ao se conectar, por exemplo, uma garra ao robô no contexto do controlador de célula robótica, este reconheceria as novas possibilidades habilitadas por meio da combinação de dois ou mais dispositivos e ofereceria o comando "pegar e colocar" (*pick and place*). Para

que isto seja possível, o controlador de célula robótica precisa conhecer as funcionalidades dos dispositivos conectados e deve inferir quais serviços ele pode oferecer ao usuário [1].

2.2 PRÉ-REQUISITOS PARA UM SISTEMA PLUG AND PRODUCE

Segundo Arai [2], um sistema de montagem pode ser dividido, de maneira simplificada, em um subsistema para dispositivos de montagem e um subsistema de gerenciamento. Para implementar a abordagem *Plug and Produce*, deve-se considerar requisitos em cada um desses subsistemas.

Cada dispositivo de montagem *Plug and Produce* precisa informar suas características ao sistema de gerenciamento, como o seu nome, o tipo, os protocolos; de forma análoga a um sistema *Plug and Play*. Além dessa informação, o dispositivo precisa informar sua posição no sistema de coordenadas de mundo (i.e. referente ao espaço de trabalho), o espaço de trabalho e outras características físicas[1] e geométricas[2].

Considerando a adaptação do sistema com relação a características físicas e geométricas, foi observado em [2] que o processo de calibração de um sistema robótico coordenado é um dos ajustes mais custosos em relação a tempo. Pela introdução de um método de calibração automático, o tempo levado seria muito menor. Além disso, colisão de espaço de trabalho, também representa outro grande problema. É teoricamente verdade que uma colisão pode ser evitada por meio de algoritmos de prevenção de colisão (*collision avoidance*) em tempo real.

[1]Descrevem a cadeia cinemática de um dispositivo. Este tipo de descrição é especialmente útil para robôs com o objetivo de verificar o alcance de certos pontos [1].

[2]Descrevem a estrutura (*body*) de um dispositivo. Estas descrições poderiam, por exemplo, ser usadas para simular a célula robótica ou como dados de entrada para um algoritmo de planejamento de trajetória/prevenção de colisão [1].

Informações de ambiente devem ser mantidas para cada dispositivo e não apenas por uma base de dados central e única no sistema de gerenciamento. Isso permite que cada dispositivo de montagem seja reconfigurado facilmente. Um sistema baseado em tecnologia *Plug and Produce* permite que um dispositivo obtenha informação circundante e trabalhe de forma autônoma.

Não é fácil definir a habilidade de cada dispositivo, pois estes possuem características variadas. A preparação de uma base de dados para dispositivos de montagem é uma questão importante.

Outro aspecto importante diz respeito aos requisitos do sistema de gerenciamento para um sistema *Plug and Produce*. O sistema deve trabalhar continuamente durante a remoção/adição de um dispositivo de montagem. O sistema deve estar apto a reconhecer o dispositivo adicionado automaticamente. Além disso, a atribuição de tarefa (escalonamento) é importante no caso de sistemas de montagem. Estas são características similares ao *Plug and Play*.

Segundo Arai [2], a implementação de um ambiente *Plug and Produce* é bem sucedida caso seja possível resolver o problema de calibração automática de sistemas de coordenada e a construção de uma base de dados contendo os dispositivos de montagem. No contexto deste doutorado, serão dados os primeiros passos para a construção de uma base de dados contendo os dispositivos de montagem.

2.3 ARQUITETURA DE CONTROLE PLUG AND PRODUCE

Foram identificados dois modelos arquiteturais de interesse que implementam o conceito de *Plug and Produce* (P'n'P). Cronologicamente, o modelo de Arai [2] foi o primeiro a introduzir o conceito de *Plug and Produce*. O modelo introduzido por Naumann [8, 1] introduz uma hierarquia em camadas

e o uso de ontologias para favorecer a sinergia de dispositivos em uma célula robótica. A seguir serão discutidos o modelo de Naumann e o modelo de Arai, respectivamente, para favorecer o entendimento e contribuições destas arquiteturas.

2.3.1 Modelo de Naumann

Naumann [8, 1] apresenta um modelo abrangente para a implementação de um sistema *Plug and Produce* ou P'n'P; o qual, associado ao conceito inicial de Arai [2], oferece características bem definidas. O modelo arquitetural proposto por Naumann divide-se hierarquicamente em três camadas [1]:

- **P'n'P de aplicação.** Oferece automaticamente serviços correspondentes a funcionalidades da célula robótica ao usuário.
- **P'n'P de configuração.** Configura automaticamente todos os parâmetros que não são de interesse do usuário, como requisitos de largura de banda, valores padrão, etc.
- **P'n'P de comunicação.** Lida com protocolos de comunicação. Ajusta automaticamente os meios de comunicação básicos entre o controlador da célula robótica e os dispositivos. Inclui a descoberta e o endereçamento de dispositivos.

A relação de interdependência entre as camadas desse modelo são apresentadas na figura 2.1. Naumann desenvolveu de maneira mais aprofundada apenas os conceitos envolvidos na especificação da camada superior; a camada P'n'P de aplicação [1].

A camada P'n'P de aplicação é a camada mais alta do modelo P'n'P proposto em [8, 1] e é responsável por oferecer ao usuário meios de acesso a dispositivos de maneira tão simples quanto possível. No contexto do projeto

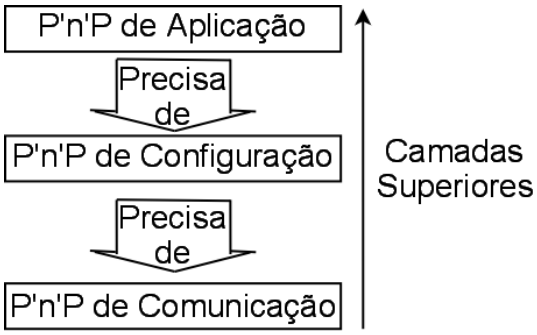

Figura 2.1: Modelo de Naumann para uma arquitetura P'n'P [1] p. 287.

SMErobot™[3] do qual é integrante, isto significa oferecer ao usuário meios de programação simplificados a células robóticas.

O estado da arte de programação de células robóticas consiste na inserção de comandos em uma caixa de diálogo em algum tipo de sistema de programação. Esses comandos são mapeados unicamente em seus respectivos dispositivos. Esse é um método de programação adequado, caso o usuário possua conhecimento detalhado a respeito da estrutura de controle de dispositivos (um robô é um tipo especial de dispositivo). Assim, espera-se em um ambiente voltado a PMEs que a programação de células robóticas seja voltada a processos que o usuário execute ao invés de programas. Naumann chama essa abordagem de **programação orientada a processo**, a qual pode ser considerada como um nível de abstração acima do modelo tradicional de **programação orientada a dispositivo**.

Em um modelo de programação orientado a processo, são considerados os seguintes requisitos quanto ao sistema de programação:

- Não ocorre integração manual de dispositivo (camada de comunicação e configuração).

- Comandos de processo são considerados entradas de usuário. Assim, ao

[3] SMErobot™ é um projeto integrado dentro do 6th *Framework Programme of the EC* para criar uma nova família de robôs adequados a PMEs e para explorar seus potenciais para manufatura competitiva. Maiores detalhes em `http://www.smerobot.org/` (acesso em 05/07/2010).

invés de mapear comandos de dispositivo considerando um dispositivo específico, as tarefas do sistema de programação são mais complexas:

- Ajuste automático de comunicação com dispositivos (camada de comunicação);
- Configuração automática de dispositivos (camada de configuração);
- Avaliação da funcionalidade de dispositivos únicos (camada de aplicação);
- Inferência da funcionalidade da célula robótica como um todo (camada de aplicação);
- Fornecimento de comandos de processo realizando uma correlação entre a funcionalidade da célula robótica e da HMI (camada de aplicação);
- Geração de sequências de comandos de dispositivo para executar os comandos de processo oferecidos (camada de aplicação).

Para cumprir essas tarefas, foi introduzido um módulo de interconexão (figura 2.2). Este módulo deve oferecer as seguintes funcionalidades:

- Descrições que contenham informação sobre os dispositivos disponíveis.
- Uma biblioteca de descrições de processo que contenha informação sobre processos que possivelmente possam ser executados pela célula robótica.
- Uma ontologia para definir e relacionar os termos das descrições supracitadas.

Para avaliar as informações dos dispositivos e estabelecer uma ontologia, os seguintes métodos são necessários:

- Um método para avaliar combinações possíveis de dispositivos, como por exemplo, a combinação de um robô e uma garra montada em sua flange.

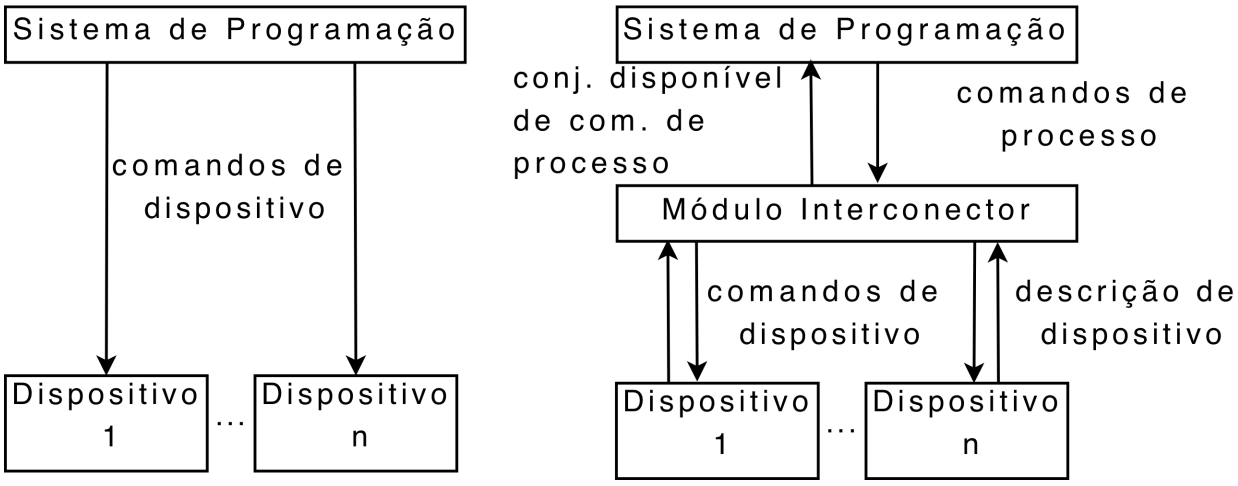

Figura 2.2: Sistema de programação tradicional vs. sistema de programação orientado a processo [1] p. 288.

- Um método para gerar descrições de dispositivos, para combinar dispositivos, tomando como base a descrição de um único dispositivo.
- Um método para determinar os processos que podem ser executados pela célula robótica.
- Um método para gerar sequências de comandos de dispositivos para todos os processos executáveis pela célula robótica.

A figura 2.3 apresenta o diagrama de fluxo para o módulo interconector. As descrições de dispositivo são fornecidas pelas camadas subjacentes de comunicação e configuração.

Em seguida, são determinadas as possíveis combinações de dispositivo. Vale observar que a geração dessas combinações pode estar sujeita ao problema de explosão combinatória[4].

As descrições de dispositivo são comparadas a descrições de processo para determinar, todos os processos executáveis. Estes processos executáveis são oferecidos ao usuário em uma HMI. O usuário define uma sequência de processos para completar uma certa tarefa.

[4]Explosão combinatória é a característica de um problema em possuir diversas alternativas de solução que ocorrem em forma de ramificações em árvores, podendo ser binária (duas ramificações em cada nó) ou não (mais de uma ramificação) [13].

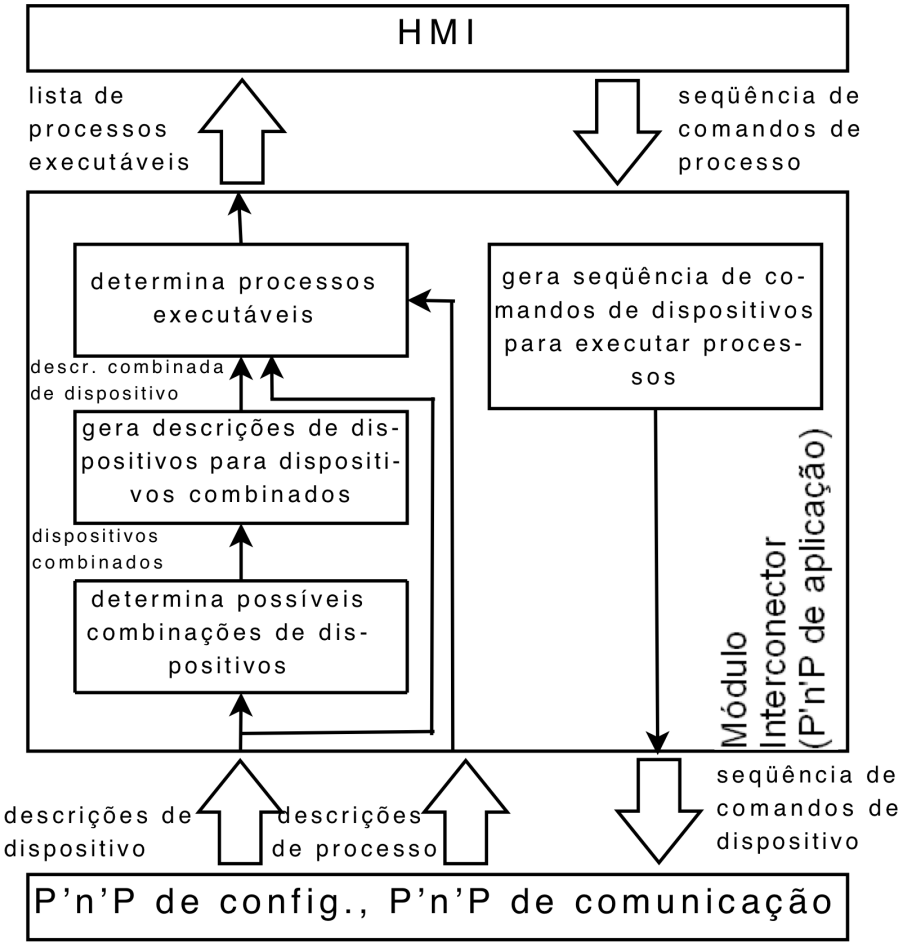

Figura 2.3: Diagrama de fluxo do módulo interconector [1] p. 289.

Conforme observado na figura 2.3, são necessários dois tipos de descrições para implementar a camada P'n'P de aplicação. A **descrição de dispositivo** contida na memória de um dispositivo, a qual é carregada no controlador da célula robótica quando o dispositivo é integrado a mesma e a **descrição de processo** que é proveniente de uma biblioteca de descrição de processo armazenada, possivelmente específica de aplicação. Ambos os tipos são divididos em seções descrevendo diferentes aspectos de dispositivos/processos [1].

As descrições citadas possuem o propósito de permitir a um módulo interconector a inferência de possíveis combinações de dispositivos, para deduzir processos que podem ser executados e gerar sequências de comandos para processos executáveis. Tudo isso deve ser feito sem a intervenção hu-

mana. Assim, o módulo interconector precisa 'compreender' as descrições, ou ainda, conhecer os conceitos representados pelos termos usados na descrição.

Para que isso seja possível, Naumann [1] introduziu uma ontologia, a qual define todos os conceitos relevantes, chamados de 'classes' do domínio considerado; sendo, no caso, o domínio robótico.

Ontologias geralmente descrevem [1]:

- **Instâncias de classes**, como por exemplo, `robot_xy` é uma instância da classe `Articulated_Robot`.
- **Classes**, como por exemplo, `Articulated_Robot`, `Flange`, `Command`, `Skill`.
- **Relações entre classes**, como "`Articulated_Robot` é uma subclasse de `Robot`" ou "Todo `Robot` possui uma `Flange`".

A figura 2.4 mostra a taxonomia de um conjunto de classes representando habilidades. Não são mostrados os atributos das habilidades, como por exemplo, no caso da classe `SkillFix` que possui o atributo `maxObjectWeight`, descrevendo o peso máximo do objeto. Todas as três subclasses de `SkillFix` herdam aquele atributo, mas podem possuir atributos mais especializados, como por exemplo, `maxObjectDiam` para `SkillFixCylinder`.

Em [1], Naumann apresenta a descrição de uma garra concêntrica que possui uma instância de `SkillFixCylinder`, significando que a garra concêntrica pode fixar cilindros. A descrição funcional de um processo de pegar e posicionar (*pick and place*) contém uma definição de um dispositivo, na forma de definição de uma classe, a qual contém uma instância de `SkillFix` necessária para executar o processo. Com o auxílio da ontologia, pode ser inferido que a garra concêntrica é adequada, por possuir a habilidade `SkillFixCylinder` que é uma sublclasse de `SkillFix`.

Este exemplo mostra que todo o domínio de conhecimento, tão trivial quanto possa parecer a um ser humano, precisa ser formalmente descrito

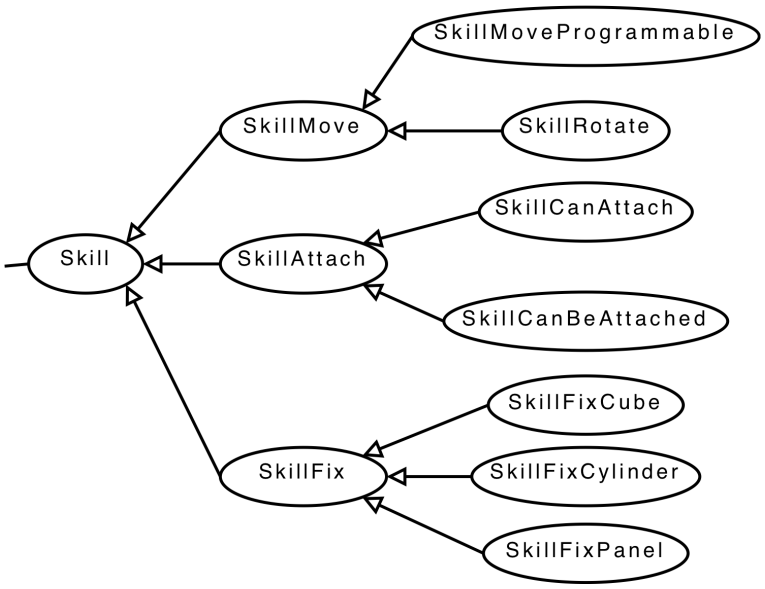

Figura 2.4: Parte da ontologia do robô [1] p. 291.

em uma ontologia. Isso permite que um computador faça uso deste conhecimento e avalie descrições de dispositivo.

2.3.2 Modelo de Arai

O trabalho de Arai [2] desenvolve os conceitos envolvidos nas camadas P'n'P de configuração e comunicação introduzidas no modelo de Naumann. Em seu sistema de produção holônico, Arai [2] especifica um sistema autônomo multi-agente e distribuído. Hólons (*holons*) possuem alta tolerância a falhas e universalidade. Boa parte da pesquisa, entretanto, lida principalmente com algoritmos de software e simulação.

Em [2], uma célula robótica foi implementada como um sistema holônico, o qual consiste em dois tipos diferentes de camadas conforme mostrado na figura 2.5. A camada de execução localiza-se no nível inferior e representa os dispositivos de montagem. Os hólons nesta camada existem estaticamente e em camada única, pois estes correspondem a dispositivos reais existentes. A camada de gerenciamento é a camada superior para gerenciamento de processos. Dentro da camada de gerenciamento, existem três tipos

de camadas:

- A **camada de tarefa** coordena todo o processo de gerenciamento de um produto. Existe apenas um hólon de tarefa (*task holon*) para a montagem de um produto. Funciona como o hólon superior da holarquia (*holarchy*) de montagem de um produto.

- A **camada de processo** gerencia um processo de montagem, o qual toma parte de todo o processo de montagem. Um hólon de processo (*process holon*) é criado dinamicamente pelo hólon de tarefa. A razão pela qual a camada de processo é introduzida é devido a uma tarefa de montagem ser constituída por uma sequência fixa determinada pelo requisito formal de um produto.

- Na **camada de operação**, um processo de montagem pode incluir uma série de dispositivos de movimento. De acordo com uma requisição de um hólon de processo, um hólon de operação faz com que o dispositivo se mova.

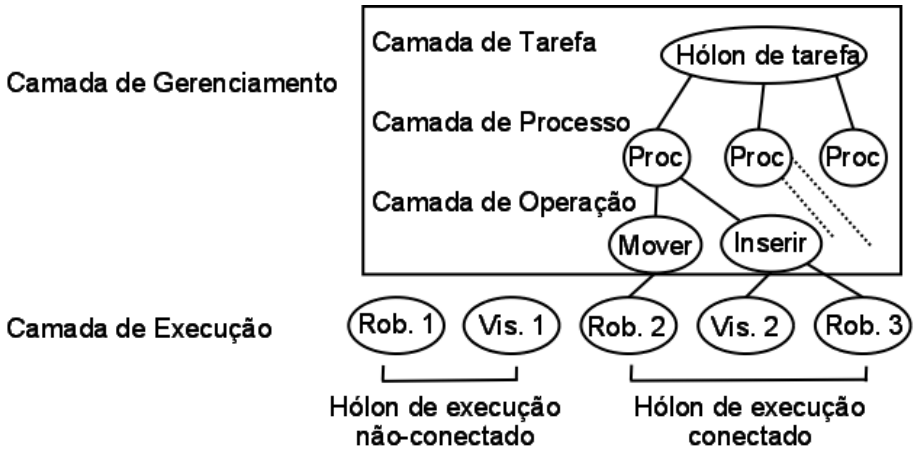

Figura 2.5: Holarquia e as camadas de gerenciamento e execução [2] p. 291.

Cada componente das camadas é um hólon, dentre os quais um hólon superior pede a hólons inferiores para executarem uma ordem e os hólons inferiores respondem em turnos (*bids*). As camadas constroem um tipo

de hierarquia chamada de holarquia, na qual hólons podem cooperar entre si com negociações entre os hólons em uma mesma camada. A holarquia é diferente de uma hierarquia, pois uma hierarquia não possui, tecnicamente, nenhuma cooperação.

2.4 CONSIDERAÇÕES FINAIS

Foram apresentados conceitos fundamentais para a implementação de uma célula robótica que permita o uso de um modelo de programação simples e acessível a usuários de PMEs com pouco conhecimento de programação (e.g. de robôs). O usuário ainda precisa realizar algum nível de programação; entretanto, em nível de comandos de processo, ao invés do uso de comandos específicos de dispositivos. Nesse contexto, um processo representa uma sequência de ações executadas por um ou mais dispositivos, conferindo a célula robótica a sinergia necessária a um sistema P'n'P.

Foram apresentadas duas abordagens para um modelo P'n'P, as quais permitem o acesso simplificado da célula robótica a usuários. O modelo de Naumann para células robóticas (seção 2.3.1) é interessante por mostrar de forma simples e clara o que se espera de um sistema P'n'P. O enfoque de seu trabalho até o momento [8, 1, 12] tem sido na camada de aplicação. Nesse contexto, apresentou sugestões interessantes para o mapeamento e associação de conhecimento, por meio do uso de ontologias, considerando um modelo hierárquico em camadas, que mascara a complexidade de implementação do usuário final.

O modelo de Naumann é genérico o suficiente para ser usado como base para a implementação de um modelo P'n'P baseado em padrões abertos. No contexto deste trabalho, o protótipo desenvolvido no capítulo 4 situa-se nas camadas P'n'P de configuração, comunicação e parcialmente na camada de aplicação. Além disso, a implementação proposta apresenta parti-

cularidades interessantes quanto a relação das camadas descritas no modelo de Naumann.

O sistema de montagem holônico de Arai (seção 2.3.2) é interessante por considerar de maneira mais detalhada os problemas envolvidos no processo de identificação automática de dispositivos em uma célula robótica e a sinergia entre os mesmos. Contrário ao modelo de Naumann, o modelo de Arai pode ser considerado uma abordagem *top down*, onde hólons de camadas superiores comunicam-se com hólons de camadas inferiores, regulando seu comportamento para que estes executem uma determinada tarefa. Esse modelo adota uma abordagem multi-agentes, a qual não é descrita com riqueza de detalhes suficiente para um trabalho de implementação prático. Além disso, não são discutidos de maneira clara requisitos de tempo real nesta arquitetura.

Uma observação interessante diz respeito a um modelo proposto por Zimmermann [14] e que não foi discutido na seção referente a modelos arquiteturais (seção 2.3). Existe a proposta para um modelo P'n'P muito semelhante ao modelo de Naumann [8], inclusive quanto aos nomes das camadas e funcionalidades. A única diferença entre ambos é que Zimmermann não especifica detalhadamente seu modelo. São fornecidas a descrição das funcionalidades de forma bastante superficial e estudos de caso, porém sem uma relação clara com o modelo proposto. Zimmerman destaca em seu trabalho a questão de tempo real em um sistema P'n'P e tendências futuras para esta área de desenvolvimento.

Zimmerman [14] afirma que tempo real e determinismo são características importantes em um sistema P'n'P. A definição enfocada por ele diz respeito a garantia de atendimento das constantes de tempo do sistema. Afirma-se que a garantia do comportamento temporal requer que a comunicação seja "previsível". Dessa forma, o termo determinismo implicaria em uma requisição precisa quanto a comunicação em tempo real. Não seria suficiente

garantir que os dados chegassem em um intervalo de tempo, e sim que os dados fossem entregues em um momento preciso.

O que Zimmermann [14] propõe para comunicação em tempo real é algo muito difícil de ser alcançado (talvez impossível). Mesmo em barramentos determinísticos, como PROFIBUS®, DeviceNet™, ControlNet™, CAN/ CANopen, InterBus e Foundation Field Bus existe uma latência inerente a operação do sistema. A operação em tempo real para detectar mudanças de estado no barramento deve estar dentro de um intervalo de tempo aceitável[5]. Em implementações de sistemas de tempo real para controle robótico que trabalham com comunicação em rede, geralmente o laço de controle em tempo real é fechado no dispositivo e não no canal de comunicação.

Quanto às tendências futuras, Zimmermann [14] observa que atividades futuras na camada P'n'P de aplicação podem ser influenciadas pela pesquisa em arquiteturas orientadas a serviço (*Service Oriented Architectures* – SOA). SOA define como componentes de software, chamados serviços, são organizados em estruturas para suportar requisitos de negócios [15]. Esta tendência pode ser verificada no trabalho de [16].

Em [16], observa-se a dificuldade para realizar a integração de dispositivos industriais. Assim, são analisadas duas arquiteturas de SOA disponíveis de modo a lidar com as questões relacionadas a integração. Apresenta-se inclusive um estudo de caso para cada arquitetura. A idéia foi realizar uma comparação compreensiva entre duas das tecnologias de SOA mais promissoras, e no processo, discutir a utilização de SOA para a integração de sistema e programação de alto nível (*high level programming* – HLP) de células robóticas. No caso, as tecnologias enfocadas em [16] são *Plug and Play Universal* (*Universal Plug and Play* – UPnP) e protocolo de estrutura de serviço descentralizado (*Decentralized Service Structure Protocol* – DSSP), o SOA presente no *Microsoft*

[5]Para maiores informações vide `http://www.freescale.com/webapp/sps/site/homepage.jsp?code=INDUSTRIALNETHOME` (acesso em 30/07/2010).

Robotics Studio (MSRS),

Este modelo arquitetural também não foi discutido na seção 2.3 por ainda não ser considerado suficientemente maduro para uso eficiente em sistemas robóticos ou em ambiente de produção.

Em [17] são feitas críticas interessantes a arquitetura SOA considerando requisitos de tempo real. Para o desenvolvimento de aplicações nesta área, são necessárias garantias de tempo real para desempenhar serviços. Isso inclui comunicação em tempo real, a qual refere-se a barramentos de campo, e a necessidade de desempenhar serviços com tempo previsível. Uma revisão de tecnologias disponíveis foi realizada e limitações inerentes de computação distribuída levam a conclusão que a abordagem SOA padrão baseada em chamadas orientadas a processo (como RPCs e *web services*), similar a orientação a objeto distribuída, não são úteis na prática.

Outra questão refere-se a diversidade e falta de padronização do SOA [18]. Em [15] afirma-se que mesmo com os padrões de SOA estabelecidos, a interoperabilidade entre plataformas (*cross-platform*) ainda é um desafio. Especificamente:

- fabricantes diferentes estão adotando padrões com diferente intensidade. Isso significa que estes padrões são parcialmente adotados e em diferentes níveis;

- dependências complexas entre padrões exacerbam a confusão na implementação.

O capítulo 3 apresentará o protocolo de comunicação XIRP, usado como base para a implementação das camadas P'n'P de configuração e de comunicação do modelo de Naumann. Este protocolo será usado para a especificação dos dispositivos e suas propriedades e para realizar a integração destes à célula robótica, independente de fabricante, via um protocolo de comunicação padronizado.

CAPÍTULO

3

P'N'P E XIRP

3.1 CONSIDERAÇÕES INICIAIS

Em ambiente corporativo é muito fácil introduzir e usar novos dispositivos. Por exemplo, para instalar uma impressora em um PC, basta plugá-la. Todas as configurações são identificadas automaticamente e a aplicação irá oferecer os serviços de impressão. Esta configuração automática é chamada de *Plug and Play*.

Levando essa noção para um ambiente de manufatura, isso significaria que existe a possibilidade de conectar, por exemplo, um robô a um controlador de célula robótica e um módulo de configuração ofereceria o serviço `move_to` em uma interface humano máquina (HMI). Por exemplo, se um robô e uma pinça (*gripper*) são conectados a um controlador de célula robótica, o módulo de configuração reconheceria o efeito sinérgico e ofereceria o serviço de "pegar e posicionar" (*pick and place*). Para conseguir isso, o módulo de configuração precisa conhecer a funcionalidade dos dispositivos conectados e

deve estar apto a tirar conclusões relativas aos serviços que pode oferecer a um usuário.

A capacidade de adicionar dispositivos a uma célula robótica e usar estes dispositivos sem a necessidade de configuração foi denominada por Naumann [8] como *Plug and Produce* (P'n'P), semelhante a abordagem *Plug and Play* em ambiente corporativo. Para muitas aplicações em automação, robôs industriais, sensores ou atores devem ser integrados em sistemas em execução. Isso significa que conexões físicas, protocolos de transferência de dados e interpretação de comandos em nível de aplicação devem ser compatíveis.

O esforço que surge a partir da integração de um robô e periféricos pode ser minimizado através da reutilização de componentes. Entretanto, essa reutilização de componentes em comunicação é apenas possível quando a comunicação é padronizada [3]. Infelizmente, apenas poucas aplicações especiais atendem esse perfil de padronização.

Exemplos de aplicações industriais que pretendem se enquadrar neste perfil de padronização podem ser encontrados na seção 3.5. Outras iniciativas de maior porte que podem ser citadas são o projeto SMErobot™, uma iniciativa da União Européia, cujo intuito foi criar uma nova família de robôs voltados a PMEs e explorar seu potencial em manufatura competitiva (com prazo de vigência entre maio de 2005 a maio de 2009) e o projeto SISU 2010 para inovação na manufatura, fundado pela Agência de Fomento Finlandesa para Tecnologia e Inovação ou Tekes. Estes projetos conseguiram sucesso em tentativas voltadas a arquiteturas P'n'P em diversos níveis, porém, as iniciativas da União Européia e da agência Tekes são as que possuem melhores propostas e resultados.

A Figura 3.1 mostra sucintamente a situação de integração de robôs e periféricos de diferentes fabricantes. No pior caso, cada robô (A e B) e sensores (X, Y e Z) possuem diferentes módulos de comunicação. Mesmo quando todos os dispositivos usam o mesmo tipo de barramento de sistema e

protocolo de transferência de dados, pelo menos na interpretação de comandos, diferenças no intercâmbio de dados irão aparecer. Neste contexto, surgem algumas perguntas:

- Quais comandos, enviados por quem, irão estabelecer determinada conexão?
- Como a transferência de dados é disparada e interrompida?
- Como é a estrutura sintática dos desvios (*offsets*) medidos pelo sensor?
- A quais eixos de coordenadas o dado se refere? Refere-se a base do robô? A uma nova ferramenta acoplada ao mesmo?

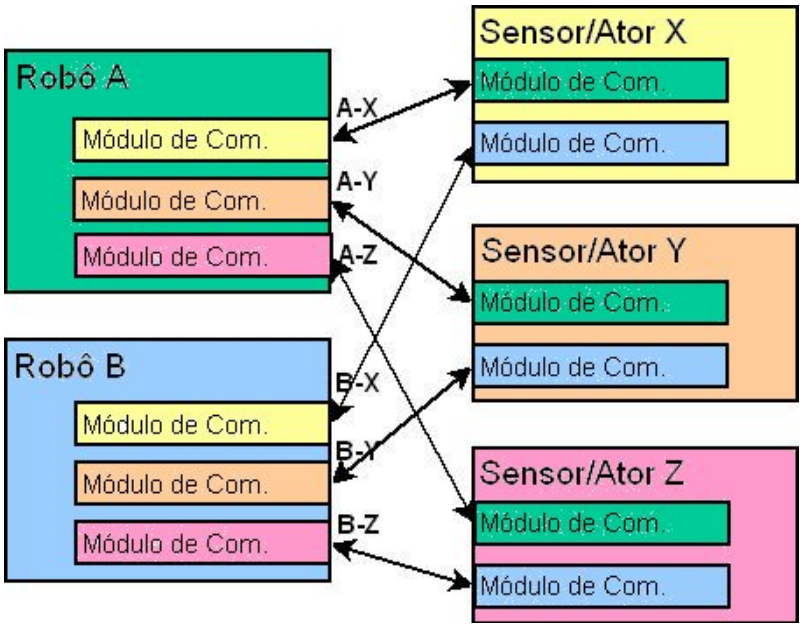

Figura 3.1: Diferentes módulos de comunicação para diferentes fabricantes [3] p. 2.

A ausência de padrões com compatibilidade de comunicação integral para robôs e periféricos, foi um assunto abordado pelo ministério alemão para para educação e pesquisa (*Bundesministerium für Bildung und Forschung* ou BMBF) e no contexto do projeto para um robô adaptativo para inspeção de partes complexas (*Adaptive Roboter-Inspektion komplexer Teile* ou ARIKT) [19, 20, 21, 22].

Posteriormente, a padronização de um protocolo de comunicação para robôs e periféricos foi tratada pela junta alemã para construção e instalação de máquinas (*Verband Deutscher Maschinen und Anlagenbau* ou VDMA). Esta iniciativa foi composta por fabricantes de robôs, sensores, sistemas de visão e usuários do setor automotivo. A VDMA definiu um perfil de comunicação; o XIRP, o qual foi chamado posteriormente de especificação VDMA com caráter de norma (*VDMA-Einheitsblatt*). O restante deste capítulo irá apresentar a especificação XIRP descrita em [11, 3].

3.2 OBJETIVOS

O principal objetivo do XIRP consiste na redução de esforços de engenharia ao ajustar a comunicação de robôs industriais e periféricos por meio de uma interface de comunicação padrão. O uso de um protocolo de comunicação padronizado, deve permitir a reutilização de componentes de comunicação. Na figura 3.2, percebe-se que existe apenas um módulo de comunicação comum a todos os dispositivos.

O reaproveitamento de componentes de software pode melhorar a qualidade dos dispositivos, pois componentes padronizados serão usados de forma difundida e irão ter menor quantidade de erros em determinado intervalo de tempo, comparado a implementações baseadas em projetos individuais de software.

Se todos os aspectos de uma certa aplicação são incluídos em uma definição para interfaces de intercâmbio em dispositivos de diferentes fabricantes, é possível referenciar esta aplicação. Nesse contexto, o protocolo de comunicação deve ter as seguintes características:

- São considerados robôs industriais e periféricos (sensores/atores) dotados de processador, os quais estão aptos a executar um protocolo de comunicação não-trivial.

- Possibilidade de intercâmbio e interpretação de informação na taxa IPO típica de robôs industriais.

- Aspectos comuns de comunicação dos dispositivos endereçados devem ser tratados de forma genérica para permitir sua reutilização.

- O protocolo é flexível e extensível para manipular certos aspectos específicos de aplicação.

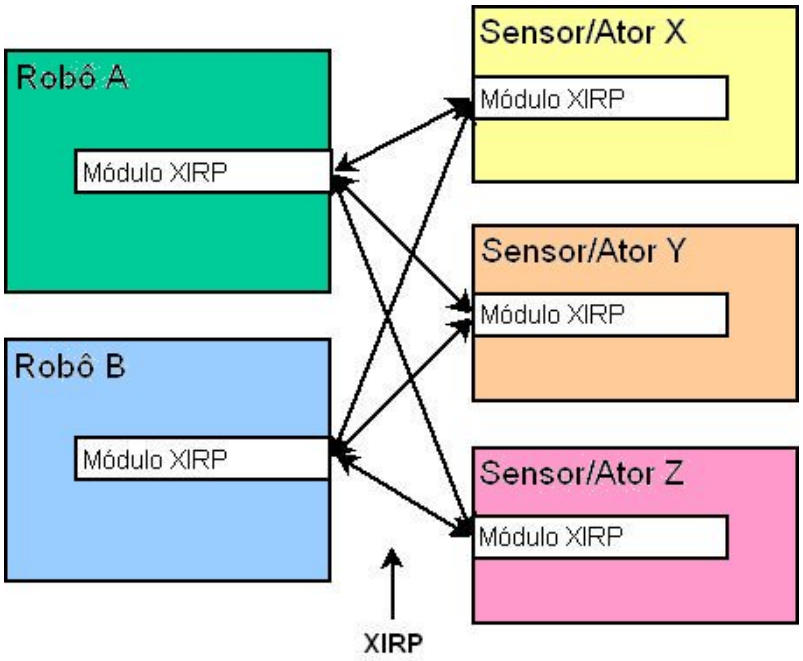

Figura 3.2: Redução de esforços de comunicação por meio de padronização [3] p. 3.

Não são parte do princípio de implementação do XIRP periféricos primitivos, não microprocessados, e o intercâmbio cíclico de dados de telegramas muito pequenos em ciclos de microsegundos como por exemplo, o controle interno de *drive* de dispositivo. Dessa forma, pode-se dizer que a criação de uma interface "universal" não faz parte dos objetivos previstos para essa especificação [3].

3.3 INTERFACE DE TRANSMISSÃO DE DADOS PARA ROBÔS E SENSORES

Até o momento de composição da especificação descrita em [11], não existia um perfil de comunicação padronizado, para a integração de robôs industriais e sensores complexos. Mesmo em trabalhos mais recentes, existe uma certa dificuldade em se definir a relação entre dispositivos e a célula robótica considerando o protocolo de comunicação a ser usado. Em [23] usa-se o termo *Plug and Play* para a descrição de um ambiente auto-configurável, citando como referência o trabalho de Gauss [3]. Conforme descrito no capítulo 2, um ambiente P'n'P não deve ser confundido com um ambiente Plug and Play. Os conceitos e a complexidade envolvida são completamente diferentes. Em [24], discute-se o modelo de produção holônico de Arai [2], apresentando-se aspectos práticos envolvidos no processo de calibração de um sistema robótico coordenado. O foco principal é a sinergia entre os diversos elementos que compõem uma célula robótica (e.g. robôs, sensores), porém não é dada atenção ao protocolo de comunicação.

Naumann [8] fornece contribuições inestimáveis com a especificação de seu modelo arquitetural; porém, apesar da citação ao protocolo XIRP, seu trabalho detalha mais profundamente aspectos da camada P'n'P de aplicação considerando a sinergia entre dispositivos e especificação de uma ontologia para a representação deste conhecimento [1].

Com um foco um pouco diferente, o trabalho desenvolvido em [25] mostra o projeto e implementação de uma infra-estrutura assistiva baseada no uso de uma rede ontológica para encapsular conhecimento em dados do produto e do processo de manufatura. Foi implementado um protótipo de ontologia com o intuito de servir como a principal fonte de dados para a geração automática de formulários digitais e diálogos por voz sintetizada para facilitar

a configuração de um processo de manufatura para placas de madeira (*wood nameplate*).

Como contribuição ao modelo de Naumann [8], serão discutidos brevemente aspectos das demais camadas de seu modelo, considerando normas consolidadas da área.

Considerando aspectos da camada P'n'P de comunicação, com a pré-norma DIN V 66311-1 ou "Automação Industrial – Interface de Sensores para Robôs e Sistemas de Fabricação – Transmissão de Dados[1] [DIN[26], 1987 apud Gauss et al.[3], 2006, p. 3] de 1987, foi dado o primeiro passo em direção a um *framework* comum para comunicação de sensores.

Como descrito na seção 3.2, muitos aspectos precisaram ser considerados para a integração completa de sensores robóticos. Considerando uma estruturação em camadas, como no modelo ISO/OSI [27], para cada camada, especialmente as camadas mais baixas que possuem aspectos especiais, existem padrões estabelecidos (figura 3.3).

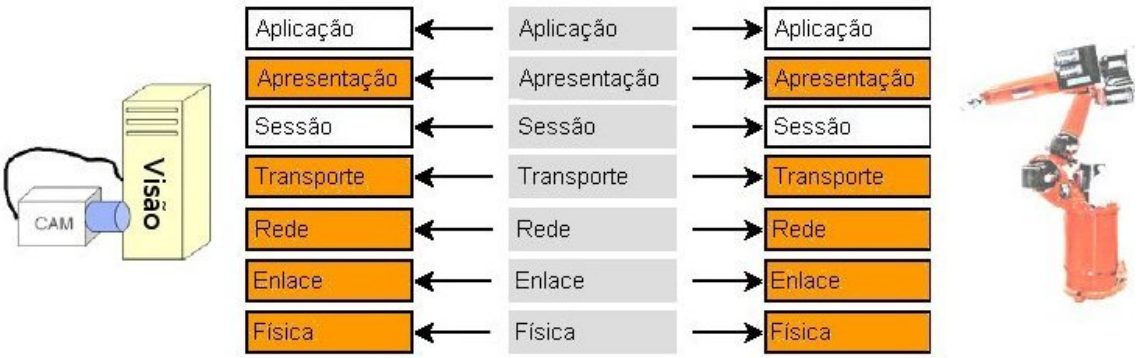

Figura 3.3: Camadas do modelo de referência ISO/OSI [3] p. 3.

Os barramentos serial RS 422/485, Ethernet e CAN são exemplos nos quais a mídia física e seu controle de acesso são padronizados. IP (camada de rede), TCP e UDP (camada de transporte) são exemplos de protocolos para transferência de dados. Considerando o nível de aplicação de robôs industriais como exemplo, o DIN EN ISO 9787 ou "Robôs Industriais – Sistemas

[1] "Vornorm" DIN V 66311-1 ou "*Industrielle Automation – Sensorschnittstelle für Roboter und Fertigungssysteme – Datenübertragung*"

de Coordenadas e Nomenclaturas de Movimento[2]", deve ser mencionado. Ele define o uso de sistemas de coordenadas.

Em automação industrial, existem perfis de comunicação completos para aplicações específicas. Por exemplo, a interface SERCOS[3] (*SErial Real-time COmmunications System*) define um perfil de comunicação completo para *drives* de controle de malha fechada. O padrão CANopen define alguns perfis de aplicação baseados em CAN. EtherCat, Interbus e PROFIBUS® são outros exemplos amplamente difundidos voltados a barramentos de campo. Esses padrões também são usados no campo de robótica industrial, porém não existe um perfil de comunicação padronizado, comum para a integração de robôs e sensores, o que remete ao nível de aplicação.

3.4 O PROTOCOLO XIRP

O padrão XIRP ou VDMA 66430-1 [11] oferece especificações para arquivos de esquema XML (ou XML schema) derivados a partir de funcionalidades de dispositivos. De acordo com [28], esquemas XML expressam vocabulários compartilhados e permitem que máquinas executem regras desenvolvidas por pessoas. Eles oferecem os meios para definir a estrutura, conteúdo e semântica de documentos XML em maiores detalhes. De acordo com [29], um esquema XML define elementos e atributos que podem aparecer em um documento, quais elementos são elementos filhos, a ordem de elementos filhos, o número de elementos filhos, se um elemento é vazio ou pode incluir texto, tipos de dados para elementos e atributos, valores padrão e fixados para elementos e atributos.

O protocolo de comunicação XIRP usa esquemas XML para estabelecer uma interface de máquina, de acordo com a pilha de protocolo or-

[2] *Industrieroboter – Koordinatensysteme und Bewegungsnomenklaturen*

[3] Para maiores informações vide `http://www.sercos.com/technology/index.htm` (acesso em 22/01/2008).

ganizada em camadas conforme definido pelo modelo de referência ISO/OSI, definido na ISO/IEC 7498-1[30]. As camadas de rede, transporte e apresentação são ajustadas para usar padrões existentes. Neste caso, os padrões citados são o Internet Protocol (IP), o Transmission Control Protocol (TCP) ou o User Datagram Protocol (UDP) e a linguagem eXtensible Markup Language (XML) respectivamente. Os mecanismos básicos do XIRP são definidos no nível da camada de aplicação para estabelecer conexões e intercambiar comandos e dados. Ele também define regras que devem ser seguidas por extensões específicas de aplicação – na forma de padrões VDMA futuros.

O uso do XIRP poderia permitir a integração de aplicações de automação, robôs industriais, sensores e atuadores em sistemas, em tempo de execução. Isto significa que conexões físicas, transferência de dados e interpretadores para comandos de protocolos no nível de aplicação deveriam ser compatíveis.

Os aspectos gerais tratados na norma VDMA 66430-1 [11] incluem os comandos básicos para estabelecer a conexão, reiniciá-la, escrever e ler parâmetros. Para executar aplicações que estão além dos comandos básicos, desenvolvimento posterior dos aspectos gerais serão necessários.

Para definir o comportamento de um dado dispositivo, é necessário criar um conjunto de arquivos de esquema XML especificados pela VDMA 66430-1 [11]. É responsabilidade do mantenedor do dispositivo especificar o arquivo XML de acordo com as características do dispositivo definidas pelo arquivo de esquema XML. Observe que esta especificação manual não seria necessária em um dispositivo com suporte nativo ao XIRP.

O XIRP especifica pelo menos os seguintes arquivos de esquema XML:

- XIRP_BasicGroup.xsd (padrão): define as operações básicas incluindo os comandos básicos disponíveis, a interface para obter (*Get*) e ajustar (*Set*)

atributos, incluindo atributos definidos pelo usuário, e a definição do tipo de canal a ser usado (canal de comando, de evento, de dados cíclicos).

- XIRP_Types.xsd (padrão): define os tipos de dados comuns disponíveis no XIRP.
- XIRP_MyApplication.xsd (definido pelo usuário): representa o arquivo principal de esquema, o qual contém a descrição de dispositivo, cujo nome é escolhido pelo usuário.

É recomendado se designar a definição para certas classes de aplicações de forma apropriada. Por exemplo, XIRP_VisionSensors.xsd para definições gerais de sensores de visão e a troca básica de mensagem, no caso, requisição ("*Request*") e resposta ("*Reply*"), além do arquivo XIRP_BasicGroup.xsd e XIRP_Types.xsd. Arquivos adicionais podem ser criados para representar comandos específicos de dispositivo, como por exemplo, XIRP_VisionGroup.xsd, complementar ao XIRP_BasicGroup.xsd e XIRP_ApplicationVision.xsd como o arquivo de descrição de dispositivo definido pelo esquema.

Considerando que os arquivos de esquema mencionados foram corretamente especificados, os arquivos XML resultantes definirão a descrição de dispositivo e o formato das mensagens que devem ser intercambiadas entre dispositivos. Mensagens incluem um nó base (raiz ou *root*), o qual pode ser "*Request*" ou "*Reply*". Uma mensagem "*Request*" é enviada do cliente para o servidor. Esta mensagem corresponde a um comando ou uma seqüência de comandos. Em uma conexão entre duas entidades, estas podem assumir o papel de dispositivo cliente ou de servidor a qualquer momento. Entre dois dispositivos, pode existir mais do que uma conexão. Em um segundo estabelecimento de conexão, os papéis de cliente/servidor podem ser diferentes.

O subnó do nó "*Request*" , pode ser "*Command*", "*Set*" ou "*Get*". A distinção é feita baseada em uma característica do comando proveniente do servidor, com base em um valor de parâmetro a ser escrito ou valores de pa-

râmetros a serem lidos. Estes estão disponíveis como subnós do terceiro nível hierárquico. A partir do terceiro nível, segue o comando de fato ou dado. O subnó do nó "*Reply*", é análogo ao subnó do nó "*Request*", e pode ser "*Command*" , "*Set*" ou "*Get*".

A figura 3.4 apresenta a estrutura do nó "*Request*" usando o modelo de conteúdo do modo de projeto para esquema XML da ferramenta XMLSpy[4]

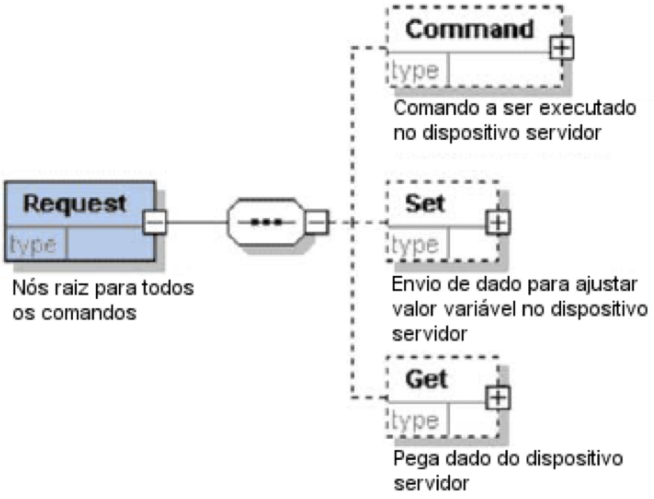

Figura 3.4: Estrutura básica de uma mensagem XIRP [3] p. 9.

A superestrutura de um comando é refletida na resposta. Assim, é possível ler um identificador de deslocamento (*offset*) de comandos, ou escrever estruturas de dados que estão sendo tratadas na resposta.

Os arquivos de esquema XIRP definem os seguintes canais lógicos para interação do dispositivo (veja a figura 3.5):

1. Canal de conexão: um dispositivo que age como um servidor deve colocar na porta 80 um arquivo de descrição de dispositivo e arquivos associados para consulta via serviço HTTP. A estrutura do arquivo de descrição de dispositivo será discutida na subseção 3.4.1. Cada cliente deve estar apto a ler os arquivos a qualquer momento, quando os dados estão dis-

[4]Disponível em: `http://www.altova.com/xmlspy/xml-schema-editor.html` (acesso em 27/07/2010).

poníveis. Se um arquivo de descrição de dispositivo for encontrado, ele deve incluir os números de porta dos canais disponíveis.

2. Canal de comando: comandos do cliente para o servidor são enviados via canal de comando. Após processar o comando, uma resposta é enviada de volta, via canal de comando. Comandos usam um canal TCP para assegurar que a transmissão seja recebida. Isto também é feito para assegurar que comandos sejam síncronos[5].

3. Canal de evento: a transmissão nesse canal é ativada por eventos. O evento pode ser percebido como um *setpoint* alcançado ou um valor dimensionado alcançado, acima de uma margem pré-definida.

4. Canal de dado cíclico: a transmissão neste canal é periódica. Existe a possibilidade de usar UDP ou TCP. Caso não se garanta que cada documento XML seja um único pacote UDP, este deve ser transmitido via TCP.

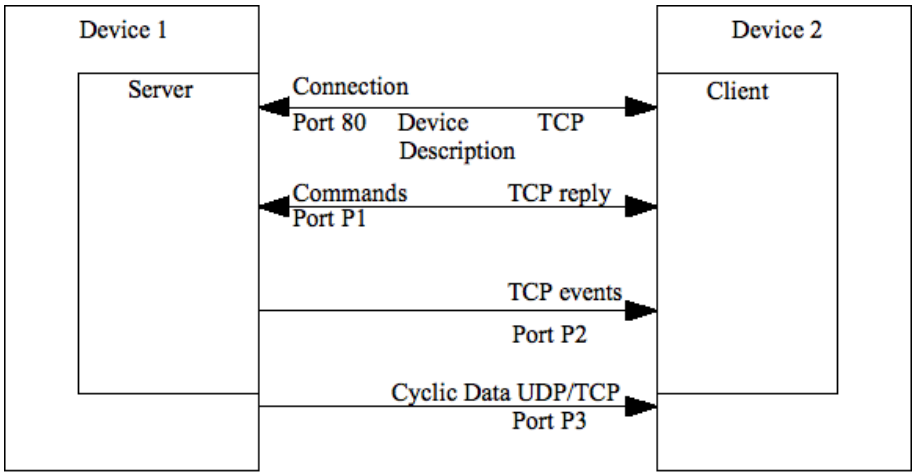

Figura 3.5: Canais XIRP.

Existe também a possibilidade para transmissão de dados binários, embora as definições XML sejam pretendidas para transmissão de dados em formato de texto.

[5]Embora o padrão preveja resposta assíncrona para comandos, recomenda-se que este comportamento não seja usado.

3.4.1 Arquivo de descrição de dispositivo

O nome do arquivo de descrição de dispositivo é "`XIRP_Device-Description.xml`" (figura 3.6). A descrição de dispositivo inclui:

- O número de versão do XIRP. Para dispositivos compatíveis com a norma VDMA 66430-1 [11], o valor é "`1.0`".

- Uma identificação eletrônica (`DeviceInfo`). Isto pode incluir, por exemplo, o tipo de dispositivo, o fabricante, o número de versão, a data de construção, etc.

- Uma referência na forma de um identificador uniforme de recurso ou *Uniform Resource Identifier* (URI) para a documentação do dispositivo (`DocumentationFile`).

- Uma referência na forma de uma URI para a interface de usuário do dispositivo (`UserInterface`).

- Uma referência na forma de uma URI para o arquivo de apresentação a ser usado (`MainSchema`) a partir dos arquivos de esquema referenciados (`Schema`).

- O nome do perfil de comunicação representando o grupo de equipamento padronizado, a ser usado como um atributo do arquivo de esquema. Se um dispositivo não é compatível com algum perfil de comunicação, ou com algum grupo de equipamento, ele deve ser inserido como uma *string* vazia.

- Um conjunto de linguagens, nas quais um esquema XML descreve comandos e disponibilidade de parâmetros (`Languages/AvailableLanguage`). Uma linguagem pré-determinada é especificada (`Languages/Default`).

- Os números de porta para comando (`Commands`), evento (`Events`) e canal de dados cíclico (`PeriodicData/TCPPortNumber` e `PeriodicData/-DefaultUDPPortNumber`). Se um canal não é oferecido a um dispositivo, seu número de porta é indicado com um "`-1`". O canal oferecido possui um par de números de porta diferentes.

A seção 3.5 apresenta aplicações e iniciativas direcionadas ao desenvolvimento de uma arquitetura P'n'P. Para demonstrar a flexibilidade da especificação XIRP, Gauss [3] implementou ambientes de teste diferenciados, incluindo soluções de grandes fabricantes como ABB, Reis Robotics e KUKA. Além disso, será apresentada uma primeira proposta para a evolução do padrão XIRP, denominada XIRP+. Estes experimentos foram realizados no contexto do projeto SMErobot^{TM}. Também será apresentado o projeto SISU 2010. Os projetos SMErobot^{TM} e SISU 2010 são parte do 6th e 7th *Framework Programme* da União Européia, respectivamente.

3.5 APLICAÇÕES

3.5.1 SMErobot^{TM}

O projeto SMErobot^{TM} foi um projeto integrado dentro do 6th *Framework Programme* da União Européia para criar uma nova família de robôs aplicáveis a PMEs e explorar seu potencial para manufatura competitiva (05/2005 - 05/2009).

O projeto SMErobot^{TM} alcançou bons resultados na criação de novos dispositivos e métodos para definição intuitiva de tarefas e movimentação de robôs, cinemática de baixo custo e fisicamente sem acidentes, na possibilidade de uma célula robótica sem "cercas" protetoras, no desenvolvimento de arquitetura e tecnologias para habilitar Plug and Produce para células robóticas e na geração de tarefas para robôs baseadas em dados de produto

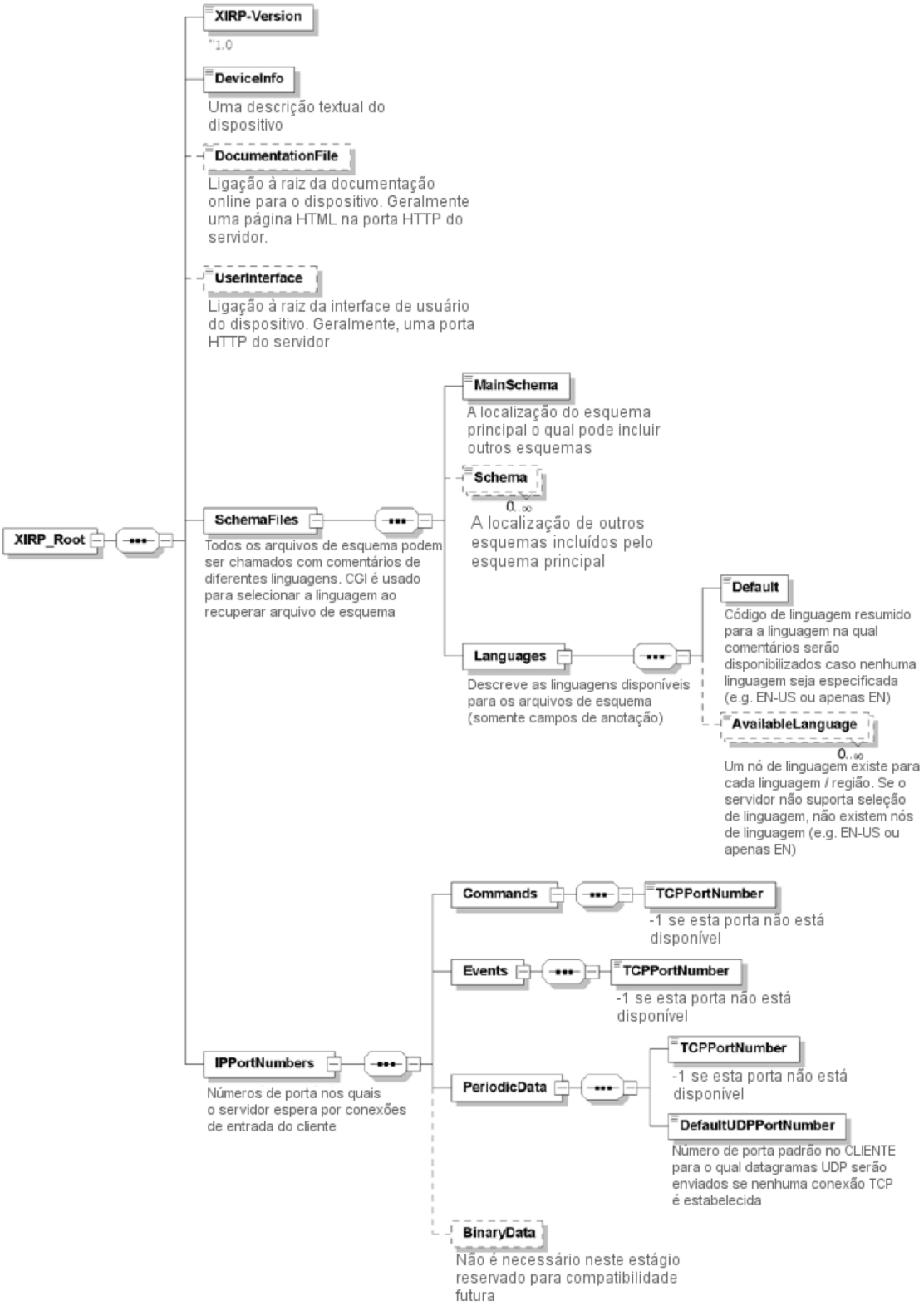

Figura 3.6: Estrutura da descrição de dispositivo XIRP [3] p. 6.

e processo. O modelo fundamental usado para tal trabalho foi produto de esforços empregados no projeto SMErobot^{TM} (seção 2.3.1).

A ampla variedade de resultados, em termos de conhecimento, demonstrações, protótipos e ferramentas de software precisam passar pelo ciclo de vida típico de desenvolvimento de produtos, seguindo o tempo de execução e penetração de mercado pós-projeto. Isso implicaria na estabilização de produtos pré-competitivos e serviços criados inicialmente como inovações, para o benefício da indústria de automação européia e usuários finais. Dessa forma, pode-se prever que:

- Em curto prazo, estima-se que principalmente produtos relacionados a software irão alcançar o mercado. Estes produtos resultantes da pesquisa desenvolvida no projeto SMErobot^{TM} serão: controladores seguros para robôs, ensino por demonstração e programação automática, ferramentas de simulação aplicáveis ao chão de fábrica, ferramentas para estimativa de custo de ciclo de vida, criação do SMErobot^{TM} *Toolbox* (para ensino e treinamento), detecção de objeto e identificação (pega de peças).

- Em médio prazo, resultados orientados a componente devem alcançar maturidade para inserção no mercado, como no caso de *drives* de alta performance, sensores de força, projetos de cinemática paralela, juntas com rigidez variável, novas garras (*grippers*), integração de novas interfaces para a instrução de robôs, como voz, varredura (*scanning*), indicação (*pointing*), redes P'n'P e sensores óticos para monitoramento de segurança ativo, etc.

- Em longo prazo, sistemas de maior porte, como no caso de projetos de cinemática inovadores para PMEs, braços robóticos com rigidez variável, assistentes para PMEs, irão alcançar maturidade para inserção no mercado.

3.5.1.1 ABB

Considerando sistemas da ABB, foram desenvolvidos dois ambientes de teste. Um deles implementava medição contínua em uma amostra de solda. Para tanto, contava com um robô ABB IRB140, um controlador robótico ABB IRC5, um *scanner laser*, um sistema de visão da Vitronic e um sistema para inspeção de solda da $VIRO^{WSI}$. O outro ambiente implementava a comunicação entre uma câmera inteligente DVT Legend 510 e um controlador robótico ABB IRC5 por meio de um barramento Ethernet. Neste caso, o arquivo descritor de dispositivo e arquivos de esquema correspondentes estão localizados na câmera e são automaticamente carregados no sistema robótico via acesso HTTP. Dessa forma, o sistema robótico é automaticamente configurado, de modo que os comandos e variáveis definidos no arquivo de esquema estejam acessíveis ao programador do robô.

A informação contida na descrição de dispositivo e arquivos de esquema podem ser usadas para configurar automaticamente a interface de comunicação do robô e sensor. Do lado do robô, por exemplo, o conjunto de instruções de programação pode ser automaticamente adaptado para o sensor ou aplicação em uso. Desta forma, o programador do robô pode usar ferramentas de programação padrão do sistema robótico (figura 3.7).

3.5.1.2 Reis Robotics

A implementação usando um controlador robótico da Reis Robotics contou com a adaptação de uma câmera baseada em rastreamento laser (*laser band*) como sensor para uma aplicação de desbaste (*trimming*) usando um cortador serrilhado (*milling cutter*), com o objetivo de testar o comportamento temporal da interface XIRP. O sensor montado no punho robótico foi movido ao longo de um componente, sobre o qual era realizada a detecção de borda via método de triangulação. A partir dos dados de borda obtidos,

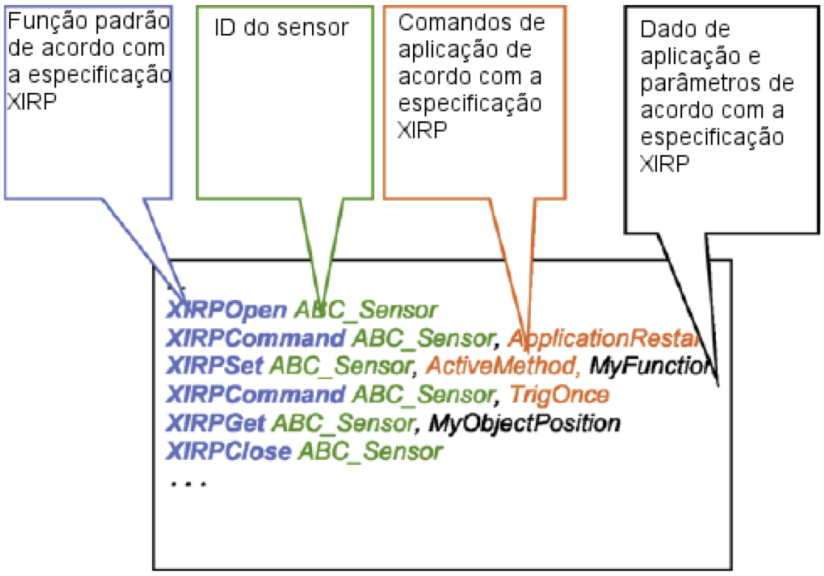

Figura 3.7: Instruções de programação configuradas para a aplicação em um sistema ABB. [3] p. 12.

o controle calculava um caminho de movimento para o processo de desbaste (*milling*).

O sensor e o controlador do robô foram conectados, ponto a ponto, para assegurar uma transferência de dados livre de colisão. Por intermédio do canal de comunicação, o controle fazia pedidos ciclicamente para o sensor, requisitando as coordenadas do componente. Por meio do telegrama de resposta, o sensor transmitia dados ao controle. Somente após receber esses dados, o controle enviaria uma nova solicitação de comando ao sensor. Este ciclo era repetido a cada 10 milisegundos. Considerando uma conexão Ethernet livre de colisão, a transmissão era realizada de forma determinística.

3.5.1.3 KUKA

A KUKA, por sua vez, implementou uma interface para dispositivos periféricos baseada em XML. O principal propósito desta interface foi permitir a fácil comunicação com dispositivos periféricos via controlador (KUKA *Robot Controller* ou KRC) e sua linguagem nativa, KUKA *Robot Language* (KRL). Por meio desta interface torna-se possível enviar e receber dados baseados em

XML, diretamente como parâmetros internos à linguagem do robô.

O envio de dados pode ocorrer facilmente, pois todas as ações são disparadas pelo controlador robótico cliente. Dados recebidos são mais complexos porque estes precisam ser armazenados. Ao invés de armazenar código XML completo, o dado é pré-processado e armazenado de forma mais eficiente, tornando seu acesso mais rápido e conveniente.

A figura 3.8 mostra o principal fluxo de dados entre um dispositivo externo e o controlador. O sistema externo determina o dado e gera uma *string* XML a qual é enviada via TCP/IP ao controlador robótico. Posteriormente, um *parser* XML implementado dentro do sistema de tempo real, no caso o VxWorks, extrai o dado e o armazena em um *buffer*. As funções da *Application Programming Interface* (API) XML permitem o acesso a partir do ambiente de programação normal do robô.

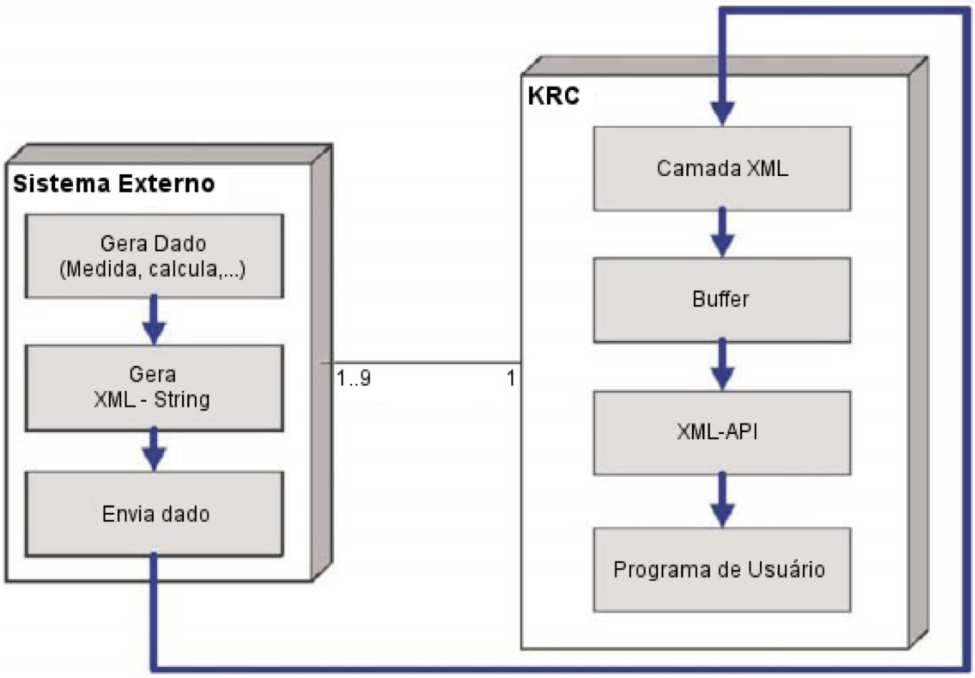

Figura 3.8: Fluxo de dado para um XML recebido. [3] p. 16.

3.5.1.4 XIRP+

Dentro do projeto Europeu SMErobot™ (divisão *Plug and Produce*), os parceiros do projeto trabalharam corporativamente em uma versão revisada do XIRP, também definida como versão 1.1 ou XIRP+ [4, 31][6].

Em [4, 31], Fan Dai et al. apresenta suas contribuições ao desenvolvimento do padrão XIRP+. São apresentadas alternativas de implementação para o conceito de P'n'P, considerando os níveis mais baixos, nas camadas de comunicação e configuração. Nesse contexto, o termo P'n'P representa a habilidade de usar um dispositivo após conectá-lo em uma rede, sem a necessidade de ajustar de forma manual os parâmetros de comunicação ou configurar as interfaces.

Conforme visto na seção 3.4, os elementos chave são a descrição de dispositivo, as definições de esquema correspondentes e o mecanismo de estabelecimento de conexão. A figura 3.9 mostra como o P'n'P funciona segundo a abordagem apresentada em [4].

O canal de descoberta de dispositivo é a primeira funcionalidade obrigatória de um dispositivo que suporta P'n'P. Após se conectar fisicamente o dispositivo a uma unidade de controle ou a rede da célula robótica, o dispositivo irá anunciar sua presença por meio do envio de uma mensagem de reconhecimento em *broadcast* (i.e. envio de mensagens a todos os pares conectados a rede - sejam dispositivos ou a unidade de controle). A unidade de controle seria ativada pelo recebimento da mensagem e iniciaria o procedimento de integração automática do dispositivo de acordo com a especificação XIRP. Desta forma, a quantidade de mensagens em *broadcast* pode ser reduzida a um mínimo durante o tempo de execução da célula robótica, pois estas são enviadas apenas quando necessário.

Após a descoberta de dispositivos, uma unidade de controle esta-

[6]Até a última data de pesquisa bibliográfica em 27/09/2010, esta versão ainda não foi publicada.

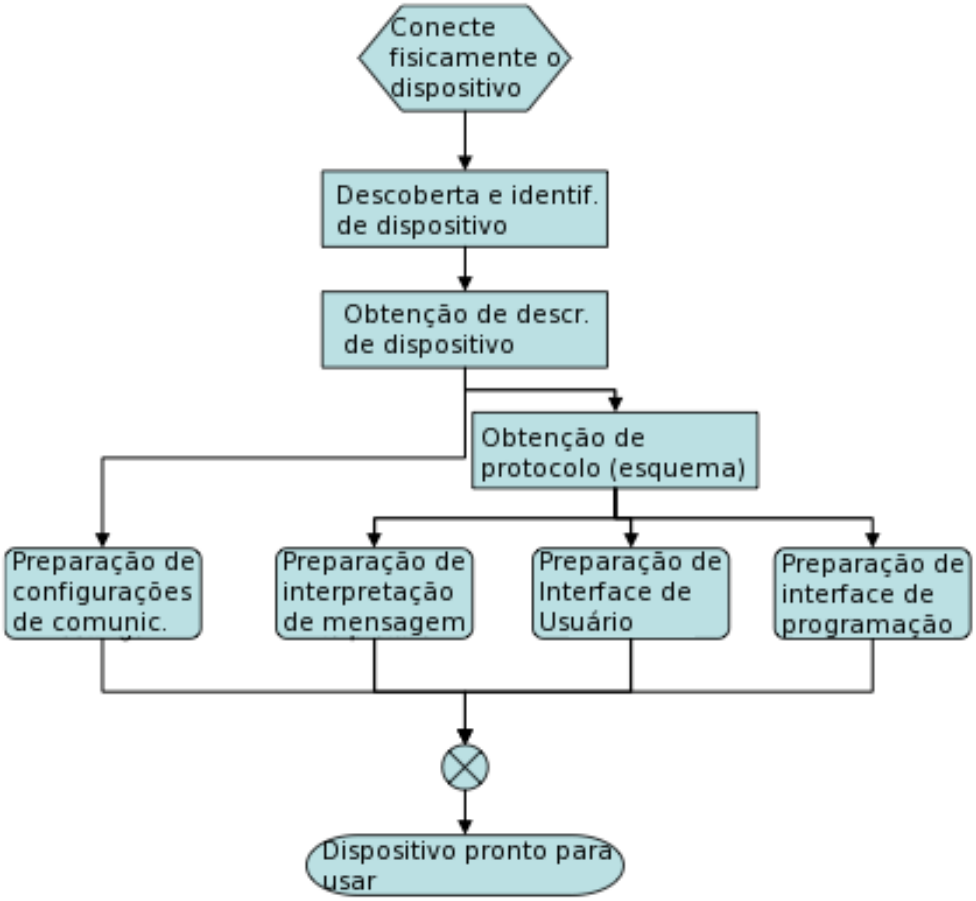

Figura 3.9: O mecanismo P'n'P proposto por [4] p. 5.

ria apta a identificar os servidores dos dispositivos e verificar que informação adicional é necessária para estabelecer a conexão a estes dispositivos. Primeiramente, as descrições de dispositivo devem estar disponíveis à unidade de controle. Um dispositivo P'n'P deve ter a descrição de dispositivo acessível via canal de arquivo. De forma alternativa, também pode haver um repositório de descrições de dispositivo disponível (e.g. para aquisição via servidor de arquivo, por exemplo). A forma mais segura para oferecer compatibilidade e descrição de dispositivo atualizada é fornecê-la no próprio dispositivo.

Com a descrição de dispositivo, a unidade de controle pode ser preparada para estabelecer a conexão ao dispositivo servidor. Isto significa que a camada P'n'P de comunicação estaria pronta. O dispositivo pode ser conectado sempre que for necessário à aplicação.

O P'n'P em nível de configuração precisa de mais informação do que oferecida diretamente no arquivo de descrição de dispositivo. Com as referências especificadas na descrição de dispositivo, a unidade de controle pode obter os arquivos de esquema. A informação oferecida com o esquema permite a unidade de controle configurar a interpretação das mensagens de comunicação. Também é possível criar ou atualizar a interface de usuário para programação de aplicação baseada em comandos e definições de tipo de dados em um esquema. Embora estes aspectos não sejam detalhados em [1, 4, 31], eles são discutidos em profundidade em [12].

3.5.2 SISU 2010

Este programa é parte da estratégia Finlandesa para o 7th *Framework Programme* da União Européia. Como tecnologia nacional, a plataforma SiSU 2010 está ligada à plataforma tecnológica MANU FUTURE[7] européia juntamente com outras plataformas de outros países europeus. Como programa tecnológico, o SISU 2010 oferece fomento para desafios de pesquisa e desenvolvimento em manufatura e age como um fórum para o intercâmbio de informações. O programa consiste em projetos de pesquisa propostos por empresas, universidades e institutos de pesquisa. O programa consumiu um volume total de 93 milhões de euros nos anos de 2005 a 2009. Projetos de cooperação possibilitam que empresas alcancem saltos tecnológicos que não seriam possíveis ao agir sozinhas.

Os objetivos do programa incluem:

- Melhoria da habilidade de manufatura de produtos discretos na indústria finlandesa para produzir de forma competitiva, em volumes pequenos ou médios, produtos altamente personalizáveis e serviços para um mercado crescente.

[7] Maiores detalhes em `http://www.manufuture.org/manufacturing/` (acesso em 28/07/2010).

- Melhoria significativa na habilidade de personalização de empresas, velocidade de entrega e confiabilidade, flexibilidade e eficiência de custo.
- Firmar a Finlândia como país desenvolvedor de classe mundial e fornecedor de tecnologia de produção e automação.
- Viabilizar projetos de pesquisa institucionais de alto padrão internacional e produzir novos modelos operacionais e tecnologias para a utilização na indústria.
- Proporcionar um aumento na colaboração internacional em pesquisa e intercâmbio de pesquisadores.

O programa beneficia os seguintes grupos:

- Manufatura discreta: tecnologia industrial com núcleo em engenharia e indústria de produtos metálicos, incluindo também manufatura de outros tipos de produtos discretos (e.g. eletrônicos, madeira, farmacêuticos e alimentícios).
- Fornecedores de máquinas e sistemas e integradores de sistema.
- Universidades e institutos de pesquisa.

As tecnologias fundamentais e temas do SISU 2010 são:

- Soluções de produção flexível para satisfazer a necessidade dos clientes rapidamente e a um custo efetivo.
- Produção avançada e tecnologias de manufatura para otimização de custo e alta qualidade.
- Auto-organização em produção, incluindo, por exemplo, inteligência embarcada e capacidade “Plug and Play” total para componentes de sistema e intercâmbio e agregação de informação.

3.6 CONSIDERAÇÕES FINAIS

Considerando o potencial tecnológico, a especificação XIRP apresenta vantagens para fabricantes de controladores, fabricantes de sensores e usuários ou integradores de sistema. Para o fabricante de controle, a solução possui vantagem ao descartar a necessidade de implementação de rotinas puramente dependentes de *drivers* de dispositivo. A mesma vantagem se aplica ao fabricante de sensores onde a integração das rotinas XIRP no código do sistema do sensor só é necessária uma única vez. Esta é a condição para a conexão a controladores aleatórios. Caso o controlador seja trocado, a interface não precisa de qualquer adaptação.

O usuário ou o integrador de sistema também leva vantagens. O conceito de XIRP possibilita a seleção e integração de dispositivos compatíveis com o XIRP que sejam adequadas a uma determinada tarefa. Isto representa um aumento considerável de flexibilidade. A desvantagem clara para o fabricante de dispositivos, sob o ponto de vista comercial, é a quebra do conceito de *keiretsu*, no qual um determinado fabricante obtém "exclusividade" sobre um determinado cliente, considerando o interesse do cliente em minimizar problemas de compatibilidade com equipamentos de outros fabricantes.

Segundo Gauss [3], existe uma tendência futura para a substituição das várias interfaces proprietárias de sensor e protocolos de comunicação pelo XIRP. Geralmente, todos os dispositivos comunicantes poderiam ser conectados a qualquer controle via protocolo XIRP. As aplicações bem sucedidas e iniciativas de grande porte da União Européia apresentadas na seção 3.5 encorajam o uso da interface XIRP e a continuidade de seu desenvolvimento. Em [32, 33] percebe-se a tendência por aplicações de redes IPs no segmento industrial, com a integração de sistemas SCADA e a necessidade por características de monitoramento e atuação em redes elétricas nos Estados Unidos. Em ambos os casos, redes baseadas em Ethernet e IP são citadas, o que elu-

cida a tendência apontada por Gauss em seu trabalho em direção ao uso de dispositivos com suporte a protocolos de alto nível (no caso do protocolo IP). Infelizmente, não é possível traçar um paralelo mais próximo tendo em vista o status de desenvolvimento incipiente do modelo PnP e da norma XIRP.

As aplicações apresentadas na seção 3.5 destacaram a versatilidade do protocolo XIRP. Considerando a solução da KUKA, a API XML desenvolvida foi usada para várias aplicações industriais. Além disso, a KUKA desenvolveu um pacote tecnológico para aplicações de visão baseadas nesta interface. Este sistema foi instalado de forma bem sucedida em ambientes industriais [Bunsendahl[34], 2003 apud Gauss et al.[3], 2006, p. 17]. Tanto o pacote tecnológico de visão quanto a API XML mostraram que o XML é muito útil e adaptável para conectar dispositivos periféricos inteligentes a um controlador robótico. Essa solução conseguiu uma redução significativa de custos em termos de penetração de mercado e ajuste de novas aplicações baseadas em sensores inteligentes.

A padronização de protocolos usando XIRP permitiria, além da redução de custos observada, um alcance maior em termos de comunicação com outras soluções existentes, permitindo a penetração em "*keiretsus*" estabelecidos, por exemplo.

O capítulo 4 apresenta uma proposta de adequação do padrão XIRP para a realidade de PMEs brasileiras. Neste contexto, a aquisição imediata de sensores/atores inteligentes para uma ampla e total adequação de uma planta é limitada, de forma que, a adoção deste padrão deverá acontecer gradativamente mantendo a infra-estrutura existente e integrando-a ao sistema de manufatura inteligente.

CAPÍTULO

4

ARQUITETURA DE CONTROLE PARA INTEGRAÇÃO DE DISPOSITIVOS (ACID)

4.1 CONSIDERAÇÕES INICIAIS

Considerando as informações apresentadas no capítulo 3, percebe-se que a proposta de uma arquitetura Plug' and' Produce (P'n'P) baseada em XIRP é bastante inovadora, e a adaptação a esta tecnologia pode ser cara em curto prazo. A realidade de PMEs brasileiras sugere que estas são ambientes promissores para a aplicação do P'n'P (conforme visto no capítulo 1).

Em PMEs, assim como em grandes empresas, as crescentes exigências a serem atendidas no tocante a produtividade e competitividade, assim como os maiores níveis de exposição externa, são desafios que as economias enfrentam, notadamente em países em desenvolvimento. Torna-se, assim, necessária a criação de um ambiente sistêmico que permita introduzir as inovações tecnológicas necessárias no contexto empresarial [35].

O cenário para a aplicação deste protocolo foi "adaptado" de modo a proporcionar um processo de aceitação e adaptação gradual, até que os dispositivos sejam produzidos em conformidade com o mesmo e a um custo acessível. O resultado foi a elaboração de uma arquitetura para um sistema integrador, de forma a implementar um modelo de integração de dispositivos gradual, utilizando como o base o XIRP. À especificação deste sistema integrador foi dado o nome de Arquitetura de Controle para Integração de Dispositivos (ACID).

Como o padrão XIRP ainda pode ser considerado experimental e em seus primeiros estágios de desenvolvimento (existe formalmente apenas um documento com as atribuições gerais do padrão e iniciativas corporativas para a sua expansão, conforme visto no capítulo 3), este projeto apresenta uma proposta de integração do XIRP em um ambiente de produção ou célula robótica, considerando a implementação de um sistema de identificação de dispositivos por meio de QR Codes (*Quick Response Codes*), aliado a um sistema intermediário (um *proxy* de dispositivos), responsável pela criação dos canais de comunicação XIRP, de modo que o dispositivo possa operar em conformidade com o padrão, além da possibilidade de integração com um *framework* robótico cujo intuito é proporcionar a integração em nível mais baixo com dispositivos. Este método é pretendido com o intuito de proporcionar uma maior cobertura do padrão, considerando sua restrição inicial a periféricos microprocessados.

4.2 ARQUITETURA DO SISTEMA

Para minimizar os esforços para a integração de um robô e seus periféricos, é possível aplicar o conceito de reuso de componentes de software, o qual tem por objetivo permitir que novas aplicações sejam criadas a partir de partes pré-fabricadas, ao invés de codificadas inteiramente [36], considerando

um cenário de comunicação padronizado. Infelizmente, somente poucas aplicações são compatíveis com este perfil de padronização.

Gauss [3] implementou vários ambientes de teste, envolvendo diferentes robôs e fabricantes de sensores (conforme visto no capítulo 3). Foram testados sistemas robóticos da ABB Robotics, Reis Robotics e KUKA. No sistema da ABB, ele testou o potencial dos arquivos de esquema na configuração, considerando a comunicação entre dispositivos e a adaptação e configuração automática do robô e a interface de sensores. No sistema da Reis, foram testadas a transmissão de dados de controle e a repetição e sincronização de troca de comandos entre sensor e robô. O objetivo foi testar o comportamento temporal da interface XIRP. No sistema da KUKA, ele testou o potencial do XML como uma interface de dispositivo para os periféricos. As cadeias de caracteres XML foram enviadas via TCP/IP a partir do sensor para o controlador robótico. Assim, esta integração permitiria a penetração de um fabricante em ambientes operacionais de outros fabricantes.

Estes foram os primeiros passos visando uma verdadeira arquitetura P'n'P para robôs, sensores e atuadores. Um empecilho encontrado, considerando a norma geral [11], é a ausência por um procedimento para a transição mais suave entre ambientes não-XIRP e os novos ambientes XIRP pretendidos. Assim, uma nova arquitetura de sistema é necessária considerando a existência de dispositivos microprocessados ou não, ainda não compatíveis com a norma. Além disso, é necessário considerar como pode ser feita a interação com dispositivos em nível mais baixo, considerando o nível de sistema operacional de tempo real ou não e ferramentas que possibilitem a implementação de um controlador de célula robótica completo. Dessa forma, como principal resultado para esta tese, propõe-se a arquitetura apresentada na figura 4.1.

A arquitetura proposta (figura 4.1) apresenta uma estruturação em três camadas. A camada inferior ou primeira camada é responsável pela

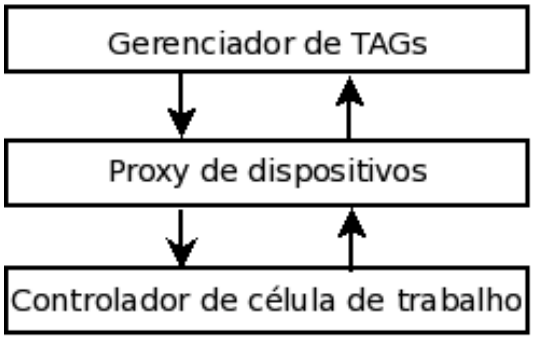

Figura 4.1: Arquitetura de sistema proposta para migração/adaptação ao protocolo de comunicação XIRP.

incorporação de dispositivos à célula robótica, em nível mais baixo. O sistema aqui descrito mostra um *framework* de código aberto como principal responsável por esta integração, porém, esta camada não se limita apenas a esta implementação. A camada de controle de célula robótica pode ser implementada utilizando qualquer sistema operacional ou tecnologia de automação (e.g. VxWorks, Linux + RTAI + Orocos, Linux + Xenomai + Orocos etc) desde que uma conexão via *socket* de rede possa ser criada para intercâmbio de comandos entre o controlador de célula robótica e o *proxy* de dispositivo.

O *proxy* de dispositivo atua como uma ponte entre dispositivos não-compatíveis em um ambiente XIRP. Isto pode significar, até certo ponto, a criação de uma interface "universal", a qual não é parte dos objetivos pretendidos nas especificações originais do padrão XIRP [3, 11]. Além disso, o *proxy* de dispositivo tem por objetivo gerar o código fonte da aplicação de interação com o dispositivo não-XIRP dinamicamente, a partir da especificação disponibilizada no sistema de *tagging*.

O sistema de *tagging* representa a inserção de um dispositivo não-XIRP no ambiente de controle. Este sistema possui um banco de dados para armazenamento das características fundamentais de um dispositivo, incluindo um identificador único par ao dispositivo, pessoa de contato, os arquivos de configuração necessários para o controle do mesmo (XIRP XML e esquema XIRP XML) e o *link* para o *proxy* de dispositivo.

As próximas seções irão discutir o uso prático do padrão VDMA 66430-1 [11] considerando a implementação do sistema de *tagging* (seção 4.3), do *proxy* de dispositivos (seção 4.4) e os ensaios realizados para a criação do controlador de célula robótica (seção 4.5), o qual no caso, representa o controlador para um robô industrial.

4.3 SUBSISTEMA DE TAGGING

No contexto deste doutorado, pretende-se identificar dispositivos por meio de algum sistema auxiliar. Isto será necessário em uma célula robótica, pois muitos dos dispositivos inseridos não possuem microprocessamento nativo, ou um protocolo para sua integração transparente (para o ambiente de testes, não foram testados dispositivos com microprocessamento nativo). Dessa forma, o sistema de identificação de etiquetas precisa possuir robustez suficiente para simplificar a sua utilização por parte de um operador com pouca ou nenhuma experiência.

O código de barra escolhido foi o QR Code, um código de barra 2D cuja especificação está disponível e a patente é de propriedade da DENSO WAVE INCORPORATED (embora esta não seja exercida). A licença para usar o QR Code estipulada pela *Japanese Industrial Standards* (JIS) e pela ISO não são necessárias. A especificação do QR Code foi disponibilizada para uso por qualquer pessoa ou organização.

O uso de telefones com câmera, para a leitura de códigos de barra em duas dimensões para vários propósitos é atualmente um tópico popular em pesquisa e para aplicações práticas [37]. Em geral, a informação oferecida por QR Codes são puramente estáticas. O que se obtém é exatamente aquilo que foi codificado. No contexto deste projeto, pretende-se usar uma noção de QR Codes que mistura um QR Code público e informação privada disponível em um servidor web, com o intuito de oferecer dados relacionados a um contexto

particular, no caso, a um dispositivo particular.

A idéia geral envolvida procura "aumentar" um documento em papel com informação eletrônica, sem modificar o formato da mídia. Este método pode ser aplicado para documentos comercialmente impressos assim como documentos obtidos em PC's. Esta técnica é chamada de realidade aumentada baseada em papel (*paper-based augmented reality*) e melhora substancialmente a utilidade do papel [38].

O sistema de marcadores ou *tagging* pode extrair a informação embutida em um QR Code e mostrá-la ao operador. A informação extraída é um *uniform resource locator* (URL) contendo o endereço do *proxy* de dispositivo e o identificador de um dispositivo legado ou incapaz de ser integrado diretamente em uma célula robótica XIRP. Dessa forma, a partir desta identificação é possível obter informações cadastradas para este dispositivo e posteriormente fornecê-las via um protocolo padronizado ao *proxy* de dispositivos.

4.3.1 Arquitetura do subsistema de tagging

O sistema de marcadores de dispositivos é parte componente de um sistema integrador (conforme visto na seção 4.2), cujo objetivo é proporcionar a identificação e integração de dispositivos por meio de um protocolo industrial padronizado, no caso, o protocolo XIRP. O problema com a utilização de um protocolo de alto nível como o XIRP refere-se a complexidade do dispositivo que o usa, tendo em vista que este precisará tratar mensagens de alto nível ao invés de apenas sinais. Nem todo dispositivo disponível hoje (e.g. sensores, atuadores etc) possui esta capacidade.

Via de regra, dispositivos que possuem esta capacidade são microprocessados, o que encarece significativamente o seu custo final. O sistema de marcadores apresentado nesta seção tem o papel intermediário para a identificação de dispositivos, sem que estes precisem ser modificados. Este sistema

representa no contexto do sistema integrador proposto, o primeiro passo para a identificação dos dispositivos em uma célula robótica. O sistema de marcadores proposto pode ser visto na figura 4.2.

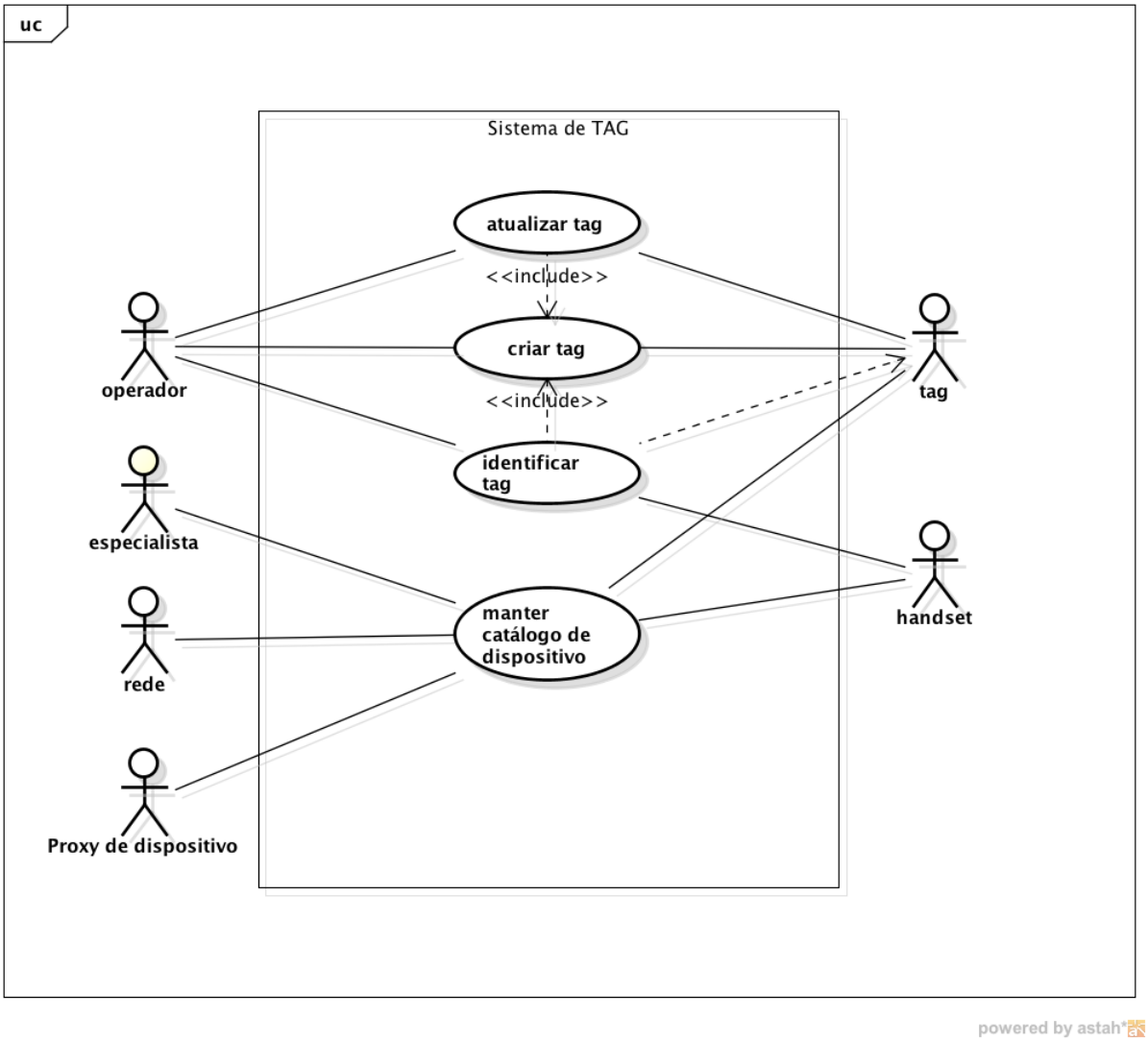

Figura 4.2: Diagrama UML de caso de uso para o sistema de marcadores proposto.

O sistema proposto (figura 4.2) possui seis atores:

- Operador: é o operador humano que irá inserir o dispositivo na célula robótica. Se este for um dispositivo XIRP, basta inserí-lo no barramento de controle da célula robótica. Caso contrário, o operador deverá conectar o dispositivo ao controlador da célula robótica e realizar a varredura de sua etiqueta com o *handset* para que este seja integrado adequadamente.

Caso a *tag* não exista, o operador deverá criar sua entrada no sistema, preenchendo as informações necessárias.

- Especialista; é o especialista responsável pela criação dos arquivos de configuração de dispositivo XIRP, caso estes não existam.

- Rede: meio de transmissão usado para realizar uma requisição pela interface com o dispositivo representado em uma *tag* para o *proxy* de dispositivo. Esta rede será compatível com a pilha de protocolos TCP/IP.

- *Proxy* de dispositivo: é o ator que "representa" um dispositivo. Dispositivos podem ser sensores, atuadores etc, cuja função é realizar I/O com o ambiente em uma célula robótica. O *proxy* de dispositivo é o responsável por transformar uma especificação XIRP de dispositivo em uma aplicação que esteja apta a controlar o mesmo. A interface de controle é uma URL acessada via dispositivo móvel (*handset*) ou PC.

- *Tag*: é a etiqueta QR Code gerada pelo sistema de tag, contendo toda a informação para a indexação do dispositivo.

- *Handset*: equipamento usado para realizar o reconhecimento da etiqueta presente no dispositivo.

Estes atores estão sujeitos a quatro casos de uso, aplicáveis a dispositivos não-XIRP:

- Criar *tag*: pressupõe a criação de uma entrada de dispositivo no sistema. Para tanto, são inseridos os arquivos de configuração XIRP e informações para referência rápida a serem inseridas na *tag* QR Code criada.

- Atualizar *tag*: Caso alguma informação inserida na *tag* seja modificada, é possível atualizar a *tag* e gerar uma nova.

- Identificar *tag*: Neste caso, como o dispositivo não tem condições de ser integrado automaticamente a célula robótica, este é identificado manualmente. Para tanto, o operador utiliza o *handset* para rastrear a etiqueta de identificação do dispositivo.

- Manter catálogo de dispositivo: Consiste na manutenção dos arquivos de configuração XIRP no sistema de gerenciamento de *tags*. Atua como um "banco de dados" dos arquivos de configuração XIRP para cada dispositivo. Estas informações são de suma importância para o *proxy* de dispositivo.

As próximas seções irão apresentar como foi realizada a implementação da arquitetura proposta (figura 4.2). Para tanto, será apresentada a implementação do *handset*, o processo de confecção da etiqueta, o software de reconhecimento para a mesma e a comunicação com o *proxy* de dispositivo.

4.3.1.1 Implementação do handset

Para implementar o equipamento de reconhecimento de etiquetas (i.e. rastreador ou *scanner*), as seguintes hipóteses foram consideradas:

- O equipamento deve ser de fácil manuseio sob o ponto de vista do operador;

- Sua implementação deve ser baseada em código aberto de modo a facilitar a portabilidade do software de reconhecimento para um novo hardware caso necessário;

- O hardware de reconhecimento deve estar apto a ler o padrão de codificação usado para as etiquetas.

Considerando as prerrogativas apresentadas e a ampla aceitação do QR Code em plataformas móveis como celulares, a plataforma de testes

foi implementada sobre um *smartphone* Google G1 (figura 4.3), com sistema operacional Android.

As principais características do hardware deste dispositivo são um display de 3.2 polegadas (8.1 cm) TFT-LCD, um processador ARM-based MSM7201A, dual-core CPU/GPU da Qualcomm, um teclado deslizante de 5 linhas QWERTY, uma câmera de 3.2 megapixel com funcionalidade de auto-foco, conectividade wi-fi (802.11b/g) e bluetooth (2.0+EDR).

Figura 4.3: Dispositivo escolhido para a implementação do *scanner*.

Não existe restrição quanto a escolha da plataforma de hardware, tendo em vista que telefones celulares e portáteis em geral hoje já possuem uma configuração bastante próxima. A única exigência é a existência de uma câmera com capacidade suficiente para o reconhecimento da etiqueta. Esta é uma restrição necessária para executar o software de reconhecimento e realizar a identificação da etiqueta adequadamente. O foco ajustável é um recurso desejável para a câmera, porém não é um impedimento para a execução da

função de reconhecimento.

4.3.1.2 Composição das etiquetas

Para a composição das etiquetas deve ser levado em consideração o padrão de codificação a ser usado (versão e código de correção de erro). Nesse contexto, existem duas possibilidades: a composição de uma URL para identificação do dispositivo posteriormente pelo *proxy* de dispositivo ou a composição de uma entrada contendo informações de contato. Para este cenário, foi escolhido o padrão MECARD da NTT DoCoMo[2], pois este pode incluir uma URL como parte integrante da mensagem. Isto proporciona um conforto a mais ao operador graças a possibilidade de tomar decisões rápidas caso seja necessário. Por exemplo, caso ele detecte um problema no dispositivo, este pode conter o contato do fabricante incluindo nome, telefone e endereço, além da URL do *proxy* de dispositivo contendo a interface para a operação do mesmo.

Para compor a etiqueta, existe a possibilidade de utilizar uma interface web disponibilizada pelo desenvolvedor do software de reconhecimento (figura 4.4). Outra possibilidade é usar a ferramenta `qrencode`[3]. Em ambos os casos o resultado produzido é suficiente para a confecção da etiqueta do dispositivo. Utilizando o software de linha de comando `qrencode` é possível especificar, por exemplo, a versão do símbolo e o nível de correção de erro, os quais representam parâmetros chave na especificação do símbolo gerado. Os valores padrão são 3 para a versão (até 77 caracteres alfanuméricos) e L para correção de erro (aproximadamente 7% de palavras de código podem ser recuperadas).

[6]Para maiores informações consulte http://www.digicamguides.com/learn/megapixels.html Acesso em: 24/02/2010.

[2]Para a especificação deste formato, visite: `http://www.nttdocomo.co.jp/english/service/imode/make/content/barcode/function/application/addressbook/index.html` Acesso em: 16/12/2010

[3]Maiores informações em `http://megaui.net/fukuchi/works/qrencode/index.en.html`. Acesso em: 24/02/2010, ou consulte a página de manual (*man page*) do mesmo.

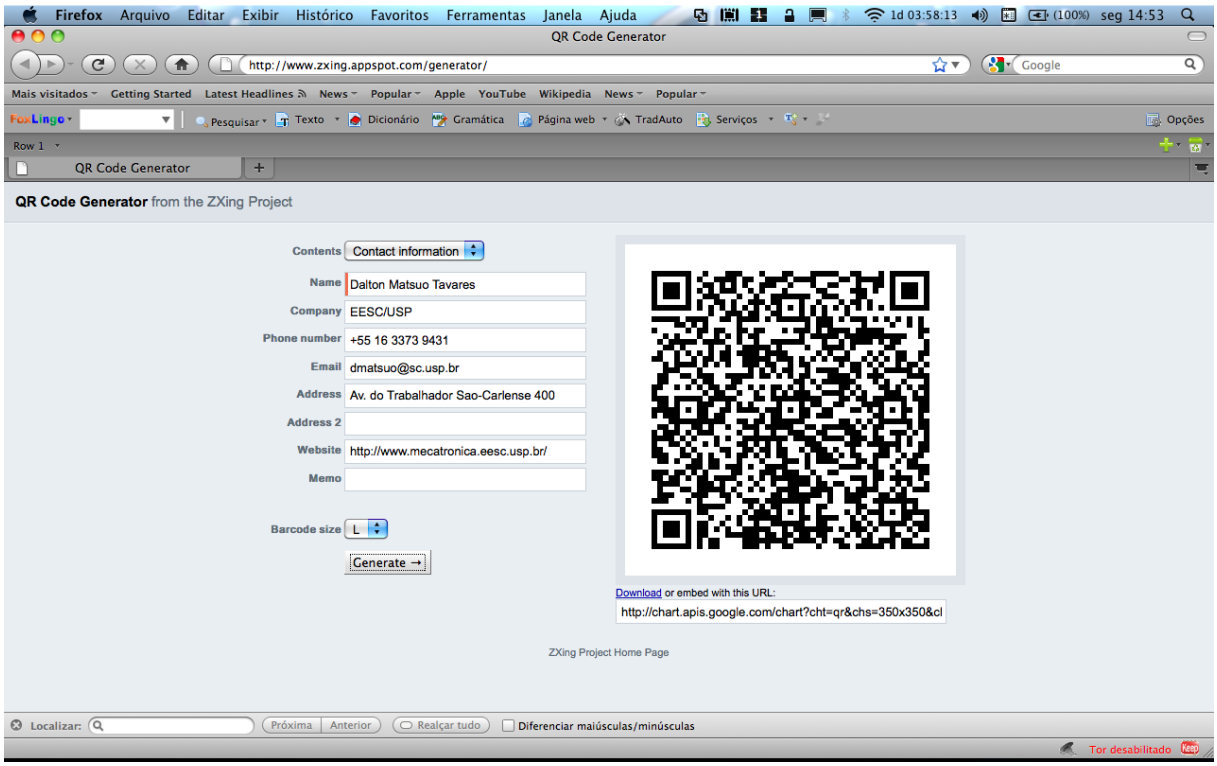

Figura 4.4: Gerador de QR Code do projeto ZXing.

Um exemplo de linha de comando utilizando a ferramenta `qrencode` é apresentada na listagem 4.1. Também foi possível adicionar a especificação de URL que é imprescindível para compor a requisição destinada ao *proxy* de dispositivo.

Listagem 4.1: Sintaxe do formato de mensagem `MECARD` proposto pela NTT DoCoMo.

```
$ qrencode -l L -v 1 -o output.png "MECARD:N:Dalton Matsuo Tavares;
TITLE:Mechatronics Engineering Doctorate (in progress);ORG:USP;
ADR:Av. do Trabalhador São-Carlense 400, Sao Carlos, SP 13566-590;
TEL:+551633739431;EMAIL:dmatsuo@sc.usp.br;
URL:http://buscatextual.cnpq.br/buscatextual/visualizacv.jsp?
id=K4701521U9;;"
```

A interface criada para a manutenção das etiquetas pode ser vista na figura 4.5. Foi usado como artifício a programação orientada a servidor

(*server side*) mediante o uso de *servlets*. Um *servlet* é uma classe da linguagem de programação Java usada para estender as capacidades de servidores, particularmente servidores web, que hospedam aplicações a serem acessadas via modelo de programação de pergunta e resposta (*request-response*)[4]. Assim sendo, para um operador acessar o sistema de *tag*, basta apontar seu navegador para a URL apropriada.

A figura 4.5(a) apresenta a janela principal da aplicação de gerenciamento de etiquetas. Nesta interface é possível criar novas etiquetas, buscar pelo cadastro de etiquetas existentes por meio do uso de palavras chave, atualizar a informação de etiquetas existentes e remover o dispositivo indexado pelo seu identificador (`tagid`). A funcionalidade de `reset` para a base de dados é usada para apagar completamente as referências e arquivos de configuração XIRP para todos os dispositivos do sistema. Em termos de desenvolvimento de software esta funcionalidade é interessante, mas em um ambiente de produção, esta deve ser utilizada com cautela.

[4]Mais informações sobre a tecnologia *servlet* podem ser encontradas em `http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html`. Acesso em 10/09/2010.

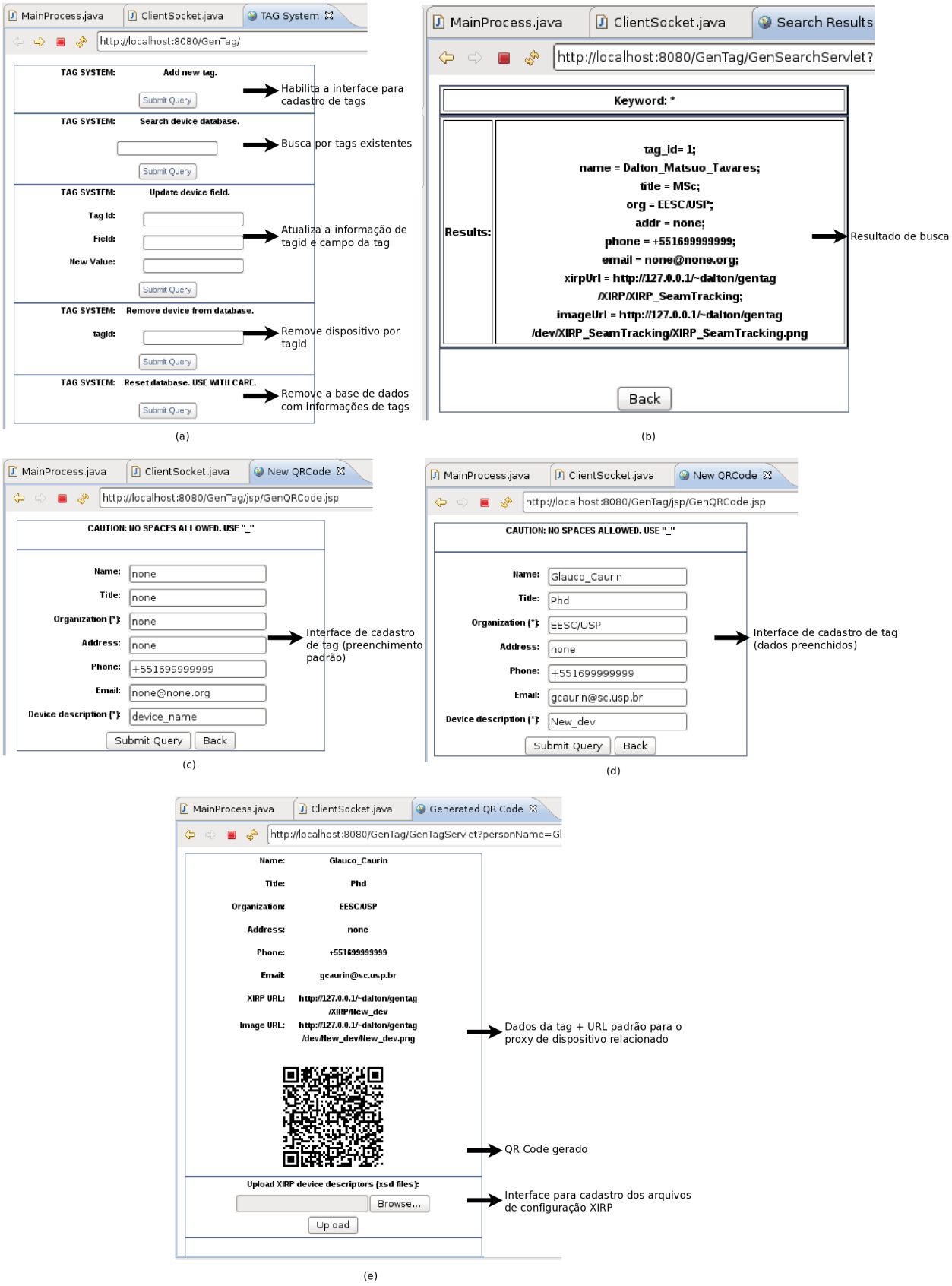

Figura 4.5: Sistema para gerenciamento de *tags*.

A figura 4.5(b) mostra o resultado de uma busca por uma *tag* no sistema. As figuras 4.5(c) e (d) mostram a interface padrão para o cadastro de *tags*. A figura 4.5(e) mostra o resultado da geração de uma *tag* e o campo para submissão de arquivos de configuração XIRP no sistema. Embora seja possível executar esta interface a partir de um celular com acesso a rede e um navegador ou qualquer dispositivo portátil, é interessante utilizar um computador com dimensões tradicionais devido a limitação de tamanho de tela destes dispositivos.

4.3.1.3 Identificação das etiquetas

Para a identificação das etiquetas, foram cogitados softwares como o KAYWA Reader[5], o Nokia Reader[6], o i-nigma Reader[7], o Active Print[8], o UpCode[9], o QuickMark[10], o BeeTagg[11], o NeoReader[12], o ScanLife[13] e o ZXing ("*zebra crossing*")[14]. Este levantamento foi realizado em grande parte em [39].

A escolha do software foi feita com base na disponibilidade, portabilidade e particularização de aplicações. Quanto a disponibilidade, todas as ferramentas mencionadas estão amplamente disponíveis, exceção feita a ferramenta Glass da Active Print que não estava disponível para obtenção. Quanto ao acesso ao código para posterior particularização, as ferramentas citadas são todas proprietárias, exceção feita ao ZXing e ao Glass. O ZXing é uma biblioteca para reconhecimento de códigos de barra implementada em Java o que a torna altamente portável e configurável por um desenvolvedor.

ZXing é uma biblioteca de código aberto implementada em Java

[5]Disponível em: `http://reader.kaywa.com/`. Acesso em: 25/02/2010
[6]Disponível em: `http://nds1.nokia.com/NOKIA_COM_1/Microsites/BetaLabs/applications/apps/Nokia_Barcode_Reader_S60_32.sis`. Acesso em: 25/02/2010.
[7]Disponível em: `http://www.i-nigma.mobi/`. Acesso em: 25/02/2010.
[8]Disponível em: `http://www.activeprint.org/` Acesso em: 25/02/2010.
[9]Disponível em: `http://www.upcode.mobi/`. Acesso em: 25/02/2010
[10]Disponível em: `http://www.quickmark.com.tw/`. Acesso em: 25/02/2010.
[11]Disponível em: `http://www.beetagg.com/`. Acesso em: 25/02/2010.
[12]Disponível em: `http://www.neoreader.com/`. Acesso em: 25/02/2010.
[13]Disponível em: `http://www.scanlife.com/`. Acesso em: 25/02/2010.
[14]Disponível em: `http://code.google.com/p/zxing/`. Acesso em: 25/02/2010.

para processamento de códigos de barra 1-D/2-D. Seu foco é o uso de câmeras integradas de telefones celulares para fotografar e decodificar códigos de barra no dispositivo, sem a comunicação com um servidor. Suporta os padrões *Universal Product Code* (UPC) e *European Article Number* (EAN), especificamente, UPC-A e UPC-E, EAN-8 e EAN-13. Além disso, suporta Code 39, Code 128, QR Code, Data Matrix (qualidade '*alpha*'), PDF 417 (qualidade '*alpha*') e *Interleaved 2 of 5* (ITF).

Devido as características desejáveis para este projeto, foi escolhida a biblioteca ZXing para a implementação do *handset* para o reconhecimento de etiquetas em dispositivos a serem integrados em uma célula robótica.

A biblioteca ZXing é dividida em vários componentes, alguns dos quais ainda estão em desenvolvimento:

- `core`: núcleo da biblioteca de decodificação de imagem e código de teste;
- `javase`: código cliente específico em J2SE;
- `android`: cliente para o sistema operacional (SO) Android chamado *Barcode Scanner*;
- `androidtest`: aplicação de teste para o SO Android;
- `android-integration`: suporta a integração com a aplicação *Barcode Scanner* via método *Intent* no SO Android;
- `zxingorg`: código fonte da biblioteca disponível [15]
- `zxing.appspot.com`: código fonte do gerador de código de barras baseado em web.

Relevantes para este projeto são o gerador de código de barra (apresentado na figura 4.4) e a aplicação para SO Android, o *Barcode Scanner*.

[15] `http://zxing.org/w`. Acesso em: 04/10/2010.

Esta aplicação pode ser obtida gratuitamente para o SO Android por meio de seu repositório de aplicações *Android Market* (figura 4.6).

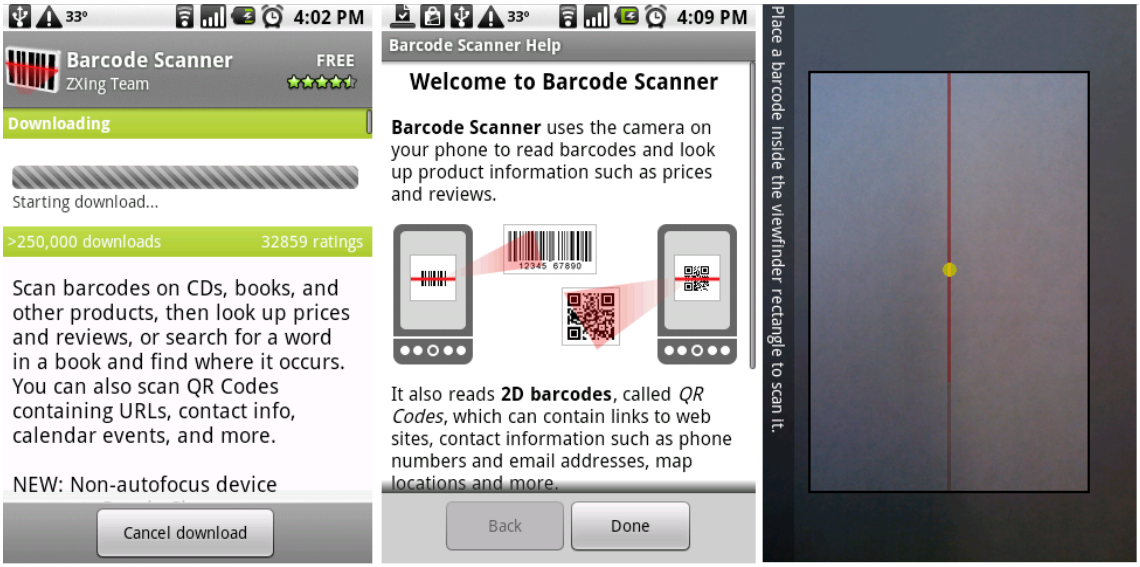

Figura 4.6: Software *Barcode Scanner* usado para implementar o *handset* para reconhecimento de etiquetas QR Code. (a) Disponibilidade do software no Android Market. (b) Software instalado. (c) Software em funcionamento.

O reconhecimento do código de barra gerado com o comando apresentado na listagem 4.1 é apresentado na figura 4.7. Independente da orientação do QR Code o reconhecimento é realizado de forma bem sucedida. Além disso, usando o formato MECARD é possível guardar informações de contato do responsável pelo dispositivo (e.g. pesquisador, técnico, empresa etc). O software *Barcode Scanner* identifica o formato MECARD e habilita ações como adicionar contato (*add contact*), mostrar o endereço do contato no Google Maps (*show map*), discar o número do contato (*dial number*) ou enviar e-mail (*send e-mail*). Além disso, a URL armazenada no código de barra, ao ser acionada, permite entrar em contato com o *proxy* de dispositivo que estará devidamente configurado para fornecer uma interface de controle para o dispositivo.

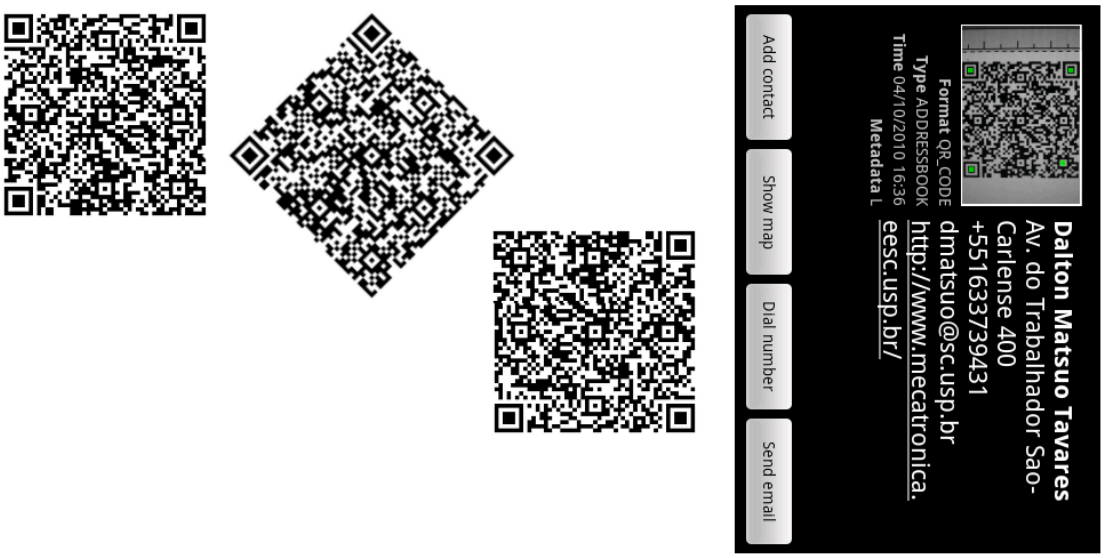

Figura 4.7: Demonstração de reconhecimento de código de barra.

A seção 4.4 mostrará o desenvolvimento para a segunda camada da arquitetura proposta para um sistema integrador de dispositivos baseado em XIRP. Este sistema tem o objetivo de geração automática de código fonte com base em arquivos de configuração XIRP de modo a viabilizar a criação automática de uma aplicação de controle. A proposta para o *proxy* de dispositivos leva em consideração a criação de uma interface padronizada a qual disponibiliza as funcionalidades do dispositivo ao usuário, mediante o uso de um navegador web, geração de código automática com base em arquivos de configuração XML XIRP e a especificação da lógica de programação para as funções mapeadas com base nestes arquivos.

4.4 SUBSISTEMA DE PROXY DE DISPOSITIVO

O *proxy* de dispositivo é parte do sistema integrador ACID cujo objetivo é incorporar dispositivos (XIRP ou não-XIRP) de maneira transparente em uma célula robótica ou linha de produção. O sistema integrador usa um sistema de *tagging* [40], um *proxy* de dispositivo [41] e um controlador de célula robótica [42].

Conforme discutido na seção 4.3, o sistema de *tag* é usado para indexar dispositivos usando uma etiqueta QR Code, com o objetivo de criar um acesso simples a informações do dispositivo em um PDA (*personal digital assistant*) ou telefone celular, desde que dotados de câmera. Esta informação inclui o endereço do *proxy* de dispositivo de modo a permitir o controle do dispositivo.

A criação e manutenção das etiquetas e a hospedagem dos arquivos de configuração XIRP também são funções deste sistema. Os arquivos de configuração XIRP podem ser armazenados localmente, no caso de uma única máquina hospedando o sistema de *tag* e *proxy* de dispositivo ou em separado, caso em que se pode configurar duas máquinas, cada uma hospedando um sistema. Esta compartimentalização é interessante pensando-se na redundância dos arquivos de configuração, os quais, em um primeiro momento são fornecidos pelo especialista ao sistema de *tag*.

O funcionamento do *proxy* de dispositivo depende intrinsecamente da precisão sintática/semântica dos arquivos de configuração XIRP fornecidos ao sistema de *tag*, com relação a composição dos arquivos XML e esquemas XML (XML *schemas*). Estes arquivos são usados para gerar dinamicamente o código para processar o arquivo de descrição de dispositivo e composição das mensagens XIRP.

O principal conceito por trás do *proxy* de dispositivo é a criação de uma aplicação orientada a servidor gerada dinamicamente, na qual cada dispositivo tem seu código automaticamente gerado. Para conseguir isto, são necessárias duas etapas distintas. O primeiro passo é gerar as classes Java e código fonte para lidar com os arquivos XML/esquemas XML do XIRP. Isto inclui o arquivo de descrição de dispositivo e as mensagens XIRP a serem intercambiadas entre dispositivos. O segundo passo consiste em realocar o código fonte gerado no primeiro passo para o lugar correto (e.g. para um diretório reconhecido pelo servidor web) e, mediante o uso de uma representação

funcional do dispositivo (gerada pelo especialista), gerar os códigos fonte necessários para o controle do dispositivo.

O controlador de célula robótica é um elemento pretendido para o controle de um robô e seus dispositivos. Sua função primária é efetuar o controle em baixo nível para os dispositivos conectados a ele, incluindo a existência de controladores para esses dispositivos e a coordenação de tarefas de tempo real. Além disso, considerando um cenário de integração transparente para dispositivos XIRP e não-XIRP, este sistema precisa de uma unidade codificadora/decodificadora para mensagens XIRP. Os ensaios relativos ao controlador de célula robótica e a sua estrutura lógica serão apresentados na seção 4.5.

Os aspectos relacionados a geração automática de código a partir dos arquivos de configuração XIRP serão discutidos a seguir. A estrutura do *proxy* de dispositivos será apresentada na seção 4.4.1. Em seguida, a seção 4.4.2 apresenta a justificativa para a utilização de um arquitetura orientada a servidor. Além disso, serão discutidos os resultados pertinentes a estrutura de geração de código e interpretação dos arquivos de configuração XIRP (primeiro passo).

4.4.1 Arquitetura do subsistema proxy de dispositivo

As principais funções do *proxy* de dispositivos são apresentadas na Figura 4.8. Este sistema possui sete atores:

- Sistema de *tag*: neste contexto, o sistema de *tag* é responsável por fornecer/manter os arquivos de configuração XIRP (XML/esquema XML). Dependendo da implementação, o sistema de *proxy* pode atuar apenas com permissão de leitura nesta base de dados ou como leitura e escrita permitindo a atualização desta por um especialista.

- Rede: meio de comunicação que permite o acesso à base de arquivos de

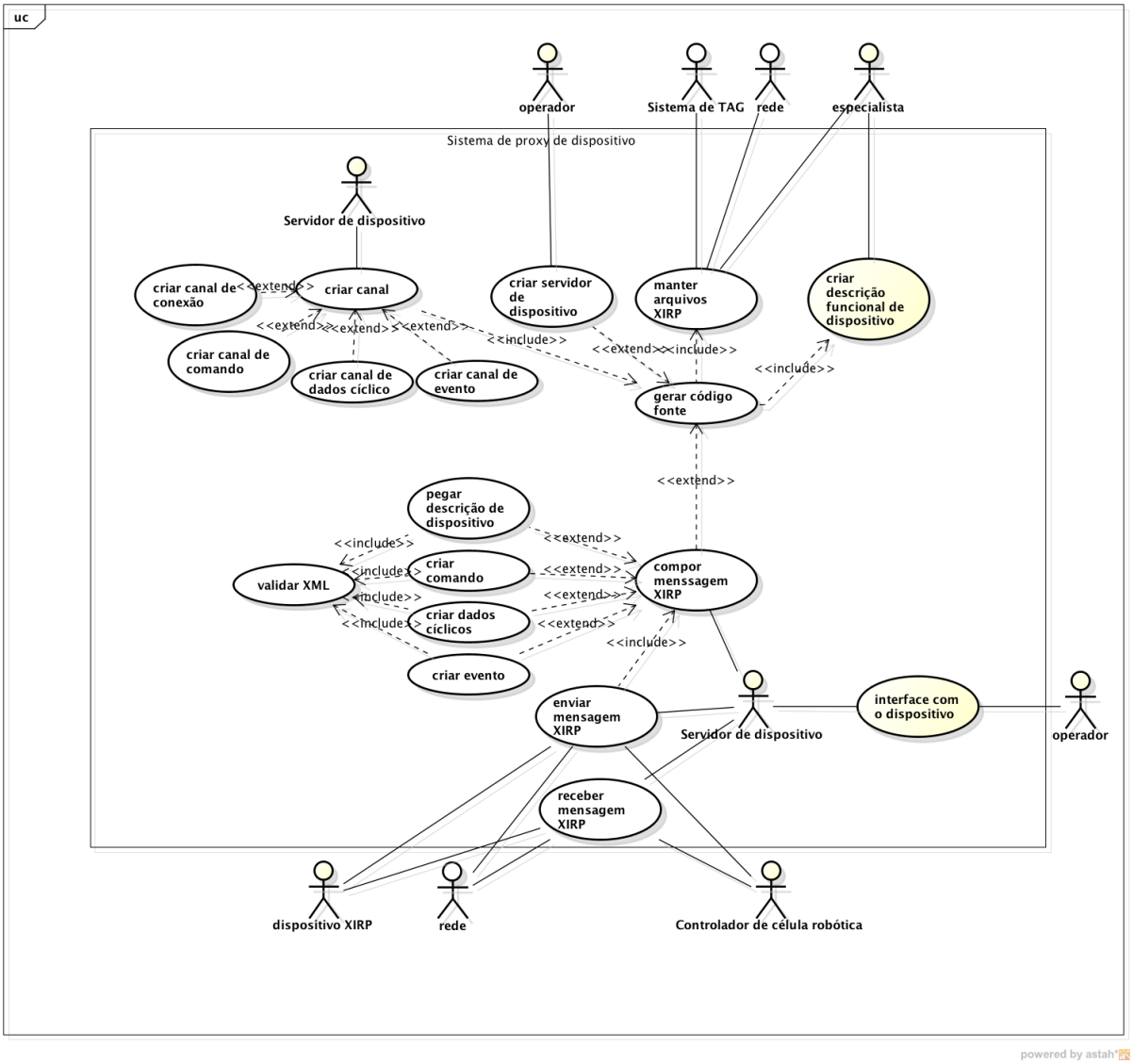

Figura 4.8: Especificação do *proxy* de dispositivo.

configuração XIRP disponível no sistema de *tag*. Também permite a troca de mensagens XIRP entre dispositivos XIRP e não-XIRP por meio do *proxy* de dispositivos e controlador de célula robótica.

- Especialista: é o responsável pela manutenção e criação dos arquivos de configuração XIRP. Neste contexto, também é responsável pela especificação funcional do comportamento do dispositivo. A especificação funcional de dispositivos não faz parte do escopo das atividades previstas.

- Servidor de dispositivo: é a aplicação *servlet* criada a partir dos arquivos

de configuração XIRP e da descrição funcional do dispositivo. Apenas o processamento dos arquivos de configuração XIRP foi implementada até o momento, para a criação de mensagens XIRP e dos canais de comunicação apropriados.

- Operador: entidade responsável pela operação do dispositivo. Faz interface direta com o servidor de dispositivo por meio de um navegador web. Como a aplicação foi desenvolvida com tecnologia orientada a servidor, não existe necessidade por *plug-ins* adicionais.

- Dispositivo XIRP: dispositivo compatível com o padrão XIRP. Pode efetuar comunicação direta com o *proxy de dispositivos* e por meio deste, com a célula robótica e dispositivos não-XIRP.

- Controlador de célula robótica: entidade responsável pelo controle de dispositivos não-XIRP. Pode ser responsável pela sinergia entre dispositivos caso todos os dispositivos se registrem em seu contexto (arquivos de configuração XIRP + descrição funcional). Maiores detalhes na seção 4.5.

Estes atores estão sujeitos a dezoito casos de uso:

- Criar canal: cria os canais necessários à comunicação com o dispositivo. Pode ser canal de conexão, canal de comando, canal de dados cíclico, canal de evento.

- Criar canal de conexão: o servidor de dispositivo coloca na porta 80 o arquivo de descrição de dispositivo e arquivos de configuração XIRP para consulta via serviço HTTP.

- Criar canal de comando: o servidor de dispositivo cria um canal de comandos TCP para o cliente. Após processar o comando, uma resposta é enviada nesse canal.

- Criar canal de dados cíclico: o servidor de dispositivo cria um canal de dados cíclico para a transmissão de dados periódica. Existe a possibilidade de usar UDP ou TCP. Caso não se garanta que cada documento XML seja um único pacote UDP, este deve ser transmitido via TCP. A escolha de implementação foi feita por um canal periódico TCP por simplicidade.

- Criar canal de evento: Por meio de um evento ajustado no canal de comando, o servidor de dispositivo cria um canal de evento. O evento pode ser percebido como um *setpoint* alcançado ou um valor dimensionado alcançado, acima de uma margem pré-definida (*noise margin*).

- Criar servidor de dispositivo: o servidor de dispositivo é o resultado da geração de código automática (processamento de arquivos de configuração XIRP + descrição funcional de dispositivos). É a entidade responsável por controlar um determinado dispositivo. Cada dispositivo possui seu próprio servidor de dispositivo. Este consiste em um *servlet* com as características de criação de canais para troca de mensagens XIRP.

- Manter arquivos XIRP: este caso de uso é opcional. Depende da postura de implementação adotada. Os arquivos XIRP podem ser mantidos de forma centralizada no sistema de *tag* ou podem possuir um caráter de manutenção distribuída e serem mantidos por um especialista também a partir do *proxy* de dispositivo. As vantagens/desvantagens de manutenção distribuída dos arquivos de configuração XIRP no contexto do sistema fogem ao contexto desta tese e não serão discutidas aqui.

- Gerar código fonte: procedimento de geração automático para o servidor de dispositivo. Este é baseado fundamentalmente na interpretação dos arquivos de configuração e geração de classes (maiores informações na seção 4.4.3) e no mapeamento funcional do dispositivo.

- Criar descrição funcional de dispositivo: mapeamento funcional do dis-

positivo. Pode ser feio via criação de uma máquina de estados que represente o seu comportamento. Este caso de uso não foi implementado no contexto deste trabalho de doutorado.

- Pegar descrição de dispositivo: obtenção do arquivo de descrição de dispositivo via canal de conexão.

- Compor mensagem XIRP: compõe uma mensagem XIRP. Esta pode ser um comando, a solicitação por um dado cíclico a ser registrado, ou a solicitação por monitoramento de um evento específico.

- Criar comando: a partir do arquivo de descrição de dispositivo e esquemas XIRP XML obtidos no canal de conexão, o *servlet* gerado para um dispositivo específico, pode criar o canal de comando na porta apropriada e gerar comandos XIRP neste canal, caso seja necessário.

- Criar dado cíclico: é possível configurar o envio de dados periódicos. Assim, um comando apropriado é inserido no canal de comandos (e.g. `<X_Offset><Periodic TimeUnit="s">0.015</Periodic></X_Offset>`), o que instrui o servidor de dispositivo a criar um canal de dados cíclico para lidar com o envio de um ou mais dados cíclicos solicitados para um determinado dispositivo.

- Criar evento: é possível configurar o monitoramento de um evento específico. Assim, um comando apropriado é inserido no canal de comandos (e.g. `<X_Setpoint><OnEvent><OnSetpoint SetPoint="10">-</OnSetpoint></OnEvent></X_Setpoint>`) e o canal de evento é criado. Quando a condição é atingida (via *setpoint* ou margem de erro), um evento é gerado no canal de evento.

- Validar XML: confere se a mensagem XML XIRP criada está de acordo com o esquema XML correspondente.

- Enviar mensagem XIRP: envia ao dispositivo XIRP ou controlador de célula robótica as instruções do operador codificadas como mensagens XIRP.
- Receber mensagem XIRP: recebe de um dispositivo XIRP ou controlador de célula robótica a resposta a uma requisição/solicitação XIRP.
- Interface com o dispositivo: o servidor de dispositivo implementa uma interface gráfica disponibilizada para o navegador, a qual o operador pode usar para codificar instruções enquanto o servidor de dispositivo as codifica para mensagens XIRP análogas.

A seção 4.4.2 discute por que foi escolhida uma abordagem orientada a servidor, por meio do uso de *servlets*, para a criação do servidor de dispositivos.

4.4.2 Adaptadores servlet

Uma proposta para a implementação do *proxy* de dispositivos é o uso do conceito de *servlet*, conforme visto na seção 4.3.1.2. Além da questão relacionada ao relaxamento da sobrecarga dos clientes (i.e. navegadores Internet), de modo a incorporar todo o sistema necessário no lado servidor, existem questões relacionadas a segurança de informação e acesso a recursos. No caso, o recurso em questão são as portas de comunicação, implementadas por meio de *sockets* de rede.

Para que um servidor de dispositivo funcione adequadamente, as portas relacionadas ao *socket* criado devem estar liberadas no contexto da rede corporativa ou intranet. Entretanto, a maioria dos ambientes de rede possuem restrições de segurança, as quais dificultam o acesso a recursos de forma indiscriminada. Por exemplo, uma aplicação de usuário não estaria facilmente acessível caso as portas utilizadas não fossem liberadas em um *firewall*.

Uma solução para este problema é usar o conceito de adaptadores *servlet* [5]. Um adaptador *servlet* é um *servlet* Java que intercepta uma requisição HTTP de um cliente para um servidor, no caso o controlador de célula robótica ou dispositivo XIRP, e redireciona essa comunicação via *socket* TCP/IP. Isto permite usar um protocolo não-HTTP após o cliente passar pelo *firewall*, por meio de um servidor web usando HTTP. Este procedimento pode ser interpretado como a criação de um túnel através de um *firewall* por meio de um servidor web (porta 80 ou 8080 no caso de um contêiner como o Tomcat[16]). Esta abordagem permite ao *proxy* de dispositivo prestar serviço em portas arbitrárias, e não apenas àquelas disponibilizadas pelo *firewall* (figura 4.9).

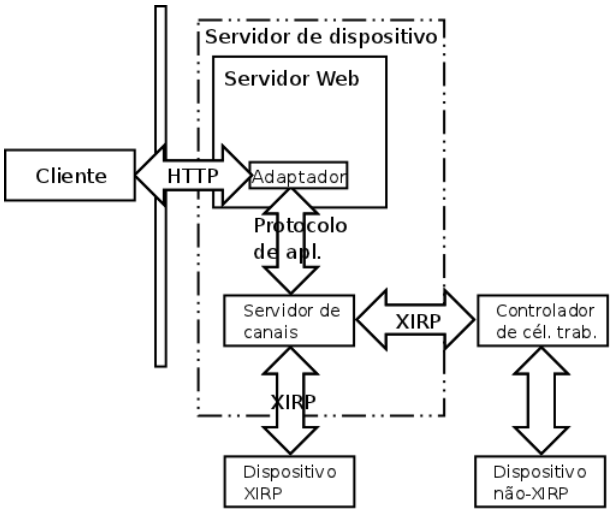

Figura 4.9: Conceito de adaptador *servlet* conforme apresentado em [5].

O adaptador *servlet* permite disponibilizar ao usuário uma interface padronizada com a qual ele pode enviar comandos a um dispositivo XIRP ou não-XIRP. O responsável por esta comunicação é um servidor de canais, o qual cria, para cada dispositivo, um conjunto de canais disponível, começando pelo canal de comandos e disparando os demais canais discutidos na seção 4.4.1, caso seja necessário. Com isto, é possível iniciar uma comuni-

[16]Disponível em `http://tomcat.apache.org/`. Acesso em: 12/09/2010.

cação com o controlador de célula robótica ou um dispositivo XIRP, usando como protocolo padrão o XIRP.

Dessa forma, o adaptador *servlet* não possui restrições com relação a comunicação via canais com os dispositivos, assumindo que o servidor web e o servidor de canais estejam ambos dentro do perímetro do *firewall*. Vale observar que, o conceito de servidor de dispositivo é aplicado ao conjunto servidor web, adaptador *servlet* e servidor de canais. Isto cria um servidor de dispositivos para um dispositivo específico, incluindo acesso a comandos básicos como `SystemReset`, `ApplicationRestart`, `CloseAllPorts` etc, e comandos específicos de dispositivo.

A seção 4.4.3 irá discutir o conceito de XML *data binding*, o qual é fundamental para o procedimento de geração automática de código, procedimento necessário à implementação do servidor de dispositivos.

4.4.3 XML data binding

XML *data binding* refere-se a habilidade de compilar um esquema XML em uma linguagem de programação adequada, com o objetivo de acessar os tipos de dados mapeados e estruturas de dados. De acordo com [43], isto pode ser conseguido por meio de um *framework* de serialização (*marshalling framework*) e um compilador de esquema.

O *framework* de serialização suporta a de-serialização (*unmarshalling*) de documentos XML em grafos de instâncias inter-relacionadas de classes existentes e derivadas de esquemas e a serialização de tais grafos novamente em documentos XML. Estes processos são governados por metadados para cada classe, os quais definem a tradução entre um documento XML externo e instâncias internas de classes associadas. Assim, a especificação estende a plataforma para estabelecer convenções para classes anotadas com os metadados necessários. Também define APIs para operações de serialização e

de-serialização, assim como aos serviços de suporte de baixo nível.

O compilador de esquema traduz um esquema em um conjunto de classes derivadas com acesso e mutação apropriados (i.e. métodos *get* e *set*), assim como aos metadados necessários pelo *framework* de serialização. O código gerado pelo compilador verifica se os documentos XML de chegada são válidos com respeito a um arquivo de esquema. O código gerado também reforça as restrições expressas no esquema, assegurando assim, que somente documentos válidos sejam produzidos pelo processo de serialização. O compilador de esquema deve ser uma ferramenta de linha de comando ao invés de uma extensão a plataforma, com o objetivo de permitir a geração automática de código.

Assim, é possível acessar qualquer tipo de dado definido dentro de um arquivo XML, especificamente, um arquivo XIRP XML, o qual foi criado com base em um esquema XML definido pela VDMA 66430-1 [11].

Para escolher a ferramenta apropriada para XML *data binding* e geração automática de código, cinco aspectos foram considerados:

1. O *proxy* de dispositivo deve ser uma aplicação orientada a servidor, aliviando a carga de instalação de programas adicionais ao usuário e limitações referentes a segurança de perímetro de rede;

2. A ferramenta de XML *data binding* deve ser *open source*, de modo a evitar dependências com uma empresa específica;

3. A ferramenta de XML *data binding* deve ser bem documentada, de modo a minimizar a curva de aprendizado;

4. Considerando o primeiro aspecto, a linguagem de programação suportada pela ferramenta de XML *data binding* deve ser o Java, dada a sua portabilidade e facilidade de integração com aplicações web;

5. A ferramenta de XML *data binding* deve suportar a geração automática de

código a partir de arquivos de dispositivo baseados em esquemas XML, considerando a possibilidade de P'n'P e integração automática de dispositivos no *proxy* de dispositivos.

Assim, a ferramenta escolhida deve traduzir esquemas XML em uma ou mais classes, processar um dado arquivo de descrição de dispositivo e permitir a implementação de uma aplicação cliente/servidor para criar a mensagem de comando ou fluxo de comandos dependendo da especificação do canal de dados a ser usado. Esta aplicação cliente/servidor também irá funcionar como uma ponte entre dispositivos de baixo nível e o cliente XIRP que emite comandos ao dispositivo controlado. Este intercâmbio de informação não diminui quaisquer capacidades de tempo real considerando que o laço de controle é mantido fora da infra-estrutura de comunicação.

As classes automaticamente geradas manipulam a interpretação dos documentos XIRP XML que são mapeados por um esquema e instâncias inter-relacionadas das classes derivadas. Elas também asseguram que as restrições expressas no esquema sejam mantidas como instâncias das classes geradas.

A operação de *data binding* mapeia os componentes de um documento XML em objetos em memória que representam, de forma óbvia e útil, o significado pretendido para o documento de acordo com seu esquema. Isto permite que programas Java que manipulam conteúdo XML sejam escritos no mesmo nível conceitual que o conteúdo, ao invés de em nível de eventos de interpretação (*parser events*) ou árvores de interpretação (*parser trees*).

Existem diversos exemplos de ferramentas para realizar XML *data binding*. De acordo com [44], alguns exemplos são o Liquid XML Data Binder, o Castor, o Happy Mapper, o Hyperjaxb, o Codalogic, o Artix Data Services, o GenerateDS, o gSOAP, o Delphi, o CodeSynthesis, o DB to XML, o Dingo, o XML Spy, o Javolution, o Jakarta Digester e o XBinder. O JaxMe2 e o

XMLBeans também foram adicionados a esta lista.

De acordo com o critério estabelecido, é possível filtrar algumas das ferramentas apresentadas. Software proprietário como o Liquid XML Data Binder, Codalogic, Artix Data Services, Delphi, DB to XML, XML Spy e XBinder não são considerados com o intuito de evitar dependência tecnológica. Happy Mapper, GenerateDS, gSOAP, CodeSynthesis não são considerados pois estes não utilizam a linguagem de programação escolhida.

O software Castor poderia ser uma boa escolha, porém ele precisa de alguma personalização para lidar com valores `AnyType` e `xs:any`, os quais são característicos de esquemas XIRP XML[17]. A mesma afirmação é feita para o JaxMe2[18].

O HyperJAXB introduz características de persistência para integrar objetos Java em tabelas relacionais[19]. Embora esta seja uma característica desejável, ela não foi usada neste estágio do projeto, dessa forma, o HyperJAXB também foi desconsiderado.

O Jakarta Digester suporta apenas o mapeamento em um sentido, de XML para objetos Java. Ele não oferece mecanismos para serializar objetos Java em XML. A configuração do Digester é moderadamente complexa, pois o desenvolvedor da aplicação precisa escrever regras para dizer ao Digester como mapear uma entrada XML em objetos Java[20]. Embora o processamento direto de XML seja importante no XIRP, o arquivo XML nem sempre contém todos os parâmetros de configuração disponíveis para o dispositivo. Esta é uma característica do esquema XML. Assim, o Jakarta Digester também não foi considerado.

[17]Mais informação em `http://www.castor.org/how-to-unmarshal-raw-xml.html` e `http://www.mail-archive.com/castor-dev@exolab.org/msg15516.html`. Acesso em: 26/05/2010.

[18]Maiores informações em `http://ws.apache.org/jaxme/features.html`. Acesso em: 26/05/2010.

[19]Maiores informações em `https://hyperjaxb.dev.java.net/doc/reference/en/pdf/reference.pdf`. Acesso em: 26/05/2010

[20]Maiores informações em `http://wiki.apache.org/commons/Digester/WhyUseDigester`. Acesso em: 27/05/2010.

O Dingo poderia ser uma boa escolha, dadas as suas funcionalidades[21]. O problema é que este é dependente da plataforma Windows, o que tornaria difícil construir ferramentas para gerar código de dispositivo automaticamente em um cenário independente de plataforma. Isto poderia ser superado com o uso do Apache Ant como uma ferramenta auxiliar[22]. A documentação do Dingo ainda é um pouco escassa, além de não apresentar exemplos com o uso do Apache Ant ou qualquer outra ferramenta para geração automática de código. Assim, o Dingo também foi desconsiderado.

O Javolution é um *framework* de tempo real com capacidade de serialização/de-serialização de XML[23]. O problema é que sua documentação também é bastante escassa o que tornaria sua curva de aprendizado relativamente maior. Assim, o Javolution também foi desconsiderado.

O Apache XMLBeans mostrou-se como a ferramenta para XML *data binding* mais adequada, de acordo com as necessidades apresentadas nesta seção. Possui documentação extensa, e está apto a lidar com valores `AnyType` e `xs:any`[24]. Seu sistema de geração de código é independente de plataforma ao usar o Apache Ant. A linguagem alvo é o Java. Assim, esta foi a ferramenta escolhida para converter esquemas XIRP XML em classes Java, com o intuito de acessar as características do dispositivo.

O primeiro passo para a geração de código já foi testado com certo nível de sucesso. O primeiro problema encontrado refere-se ao formato dos esquemas XIRP. Os esquemas não são válidos por si. Isto ocorre pois atributos `minOccurs`/`maxOccurs` não são permitidos em *tags* `<all/sequence/choice>` que sejam filhas diretas de um grupo nomeado. Isto é especificado nas recomendações da W3C para a estrutura de esquemas XML [45]. Um trecho

[21]Maiores informações em: `http://dingo.sourceforge.net/features.shtml`. Acesso em: 28/05/2010

[22]Maiores informações em `http://ant.apache.org/manual/index.html`. Acesso em: 28/05/2010.

[23]Maiores informações em: `http://javolution.org/`. Acesso em: 28/06/2010

[24]Maiores informações em `http://xmlbeans.apache.org/docs/2.0.0/guide/conXMLBeansSupportBuiltInSchemaTypes.html`. Acesso em: 28/06/2010.

de um esquema XML exemplo (XIRP_BasicGroup.xsd) incorrendo em erro é apresentado na listagem 4.2. O atributo `maxOccurs` aparece com erro ao ser validado contra o padrão da W3C.

Listagem 4.2: Trecho de um esquema XML exemplo incorrendo em erro.

```
...
<xs:group name=''BasicCommands''>
        <xs:choice maxOccurs=''unbounded''>
        <xs:element name=''SystemReset''
...
```

Propõe-se duas soluções para este problema. A primeira solução seria cercar a cláusula `xs:choice` entre duas cláusulas `xs:complextype`. A outra possibilidade seria simplesmente remover as cláusulas `minOccurs`/ `maxOccurs` ao incorrer em erro. Optou-se por remover essas cláusulas, visando obter maior compatibilidade com relação a estrutura padrão de esquema XML da W3C.

A geração de XML pelo Apache XMLBeans possui algumas limitações. O arquivo XML gerado respeita as recomendações da W3C e é corretamente validado contra o arquivo de esquema XIRP XML. O problema é que o cabeçalho não respeita precisamente o padrão XIRP. Por exemplo, mostra-se na listagem 4.3 uma mensagem de resposta XML (XML *reply*) gerada pelo Apache XMLBeans e sua contra-parte no padrão XIRP na listagem 4.4. Isto significa que a mensagem gerada precisa de algum pós-processamento para fazê-la precisamente compatível com o padrão.

Listagem 4.3: Comando XML básico gerado usando o Apache XMLBeans.

```
<Reply xmlns=``http://www.xirp.org/Schema''>
  <Command xmlns=```''>
      <SensorOff/>
  </Command>
</Reply>
```

Listagem 4.4: Comando XML básico de acordo com a especificação XIRP.

```
<?xml version=``1.0'' encoding=``UTF-8''?>
<XIRP:Request
  xmlns:XIRP=``http://www.xirp.org/Schema''
  xmlns:xsi=``http://www.w3.org/2001/XMLSchema-instance''
  xsi:schemaLocation=``http://www.xirp.org/Schema
  ..\XIRP_XML_SeamTracking_Schema\XIRP_ApplicationTracking.xsd">
  <Command>
    <SensorOff/>
  </Command>
</XIRP:Request>
```

Embora este não seja o cenário ideal, a compatibiidade pode ser facilmente alcançada usando *shell scripting* e o editor *sed* (*stream editor*) no Linux. Originalmente escrito e projetado para Unix, o *sed* foi portado para o MS-DOS, Windows, OS/2 e outros sistemas operacionais. O *sed* é um editor não-interativo que funciona a partir de linha de comando; não é uma GUI como a maioria das aplicações Windows. O *sed* edita blocos de texto, sem a necessidade de abrir uma tela, mover um cursor e pressionar alguma tecla[25]. Assim, é relativamente fácil criar um pós-processador usando o *sed* e *shell scripting* para remover a primeira, segunda e última linhas da mensagem ge-

[25]Mais informação em: `http://www.pement.org/sed/` Acesso em: 02/06/2010.

rada pelo Apache XMLBeans (`<Reply..>`, `<Command...>`, `</Reply>`) e inserir as cláusulas equivalentes no formato exigido pelo padrão XIRP.

Com os aspectos gerais para a comunicação de dispositivos via XIRP implementados, o próximo passo é discutir a interface de baixo nível para dispositivos não-XIRP. A seção 4.5 apresenta os resultados experimentais obtidos na implementação de uma célula robótica utilizando para isso ferramentas de código aberto como prova de conceito.

4.5 SUBSISTEMA CONTROLADOR DE CÉLULA ROBÓTICA

Considerando as limitações estruturais e financeiras inerentes a PMEs, optou-se por utilizar uma plataforma de pesquisa aberta baseada em um robô IBM 7545 SCARA, disponível no Laboratório de Mecatrônica do Departamento de Engenharia Mecânica e controlada por software de código aberto. Com isso, é possível destacar claramente as contribuições desenvolvidas para o módulo controlador de célula robótica, sem levar em consideração restrições características de sistemas proprietários e fechados.

A principal característica pretendida para o controlador de célula robótica levando em conta o ambiente de PMEs é a rápida adaptação do robô e outros dispositivos a um cenário considerando a mudança de produtos e reconfiguração de processos.

4.5.1 Arquitetura do subsistema controlador de célula robótica

O sistema proposto (figura 4.10) possui cinco atores:

- Dispositivo XIRP: dispositivo com suporte nativo ao protocolo XIRP. Neste contexto, esta comunicação também poderia ser feita por outro controlador de célula robótica via *proxy* de dispositivo.

- Rede: rede de comunicação para a recepção/envio de mensagens XIRP.

Além disso, também pode ser usada para a criação de um canal no qual o operador pode monitorar a situação da célula robótica e de seus dispositivos.

- *Proxy* de dispositivo: no contexto deste módulo, o *proxy* de dispositivo é usado como um codificador/decodificador de mensagens XIRP. Este ator é usado no contexto do controlador de célula robótica para reaproveitamento de código. Pode ou não estar contido na mesma máquina que o controlador de célula robótica. Isto é possível graças a troca de mensagens via *sockets* de rede.

- Gerente de célula robótica: entidade responsável pelas atividades básicas de controle de um dispositivo não-XIRP dada a decodificação da mensagem recebida. Além disso, é responsável por criar a sinergia entre dispositivos, partindo-se do pressuposto de que as representações funcionais para os dispositivos e para a célula robótica estão disponíveis. Este caso de uso não faz parte do escopo desta pesquisa considerando os aspectos relativos a integração cooperativa de dispositivos.

- Dispositivo não-XIRP: dispositivo integrado à célula robótica por meio de artifícios como placas de I/O e implementação de controladores de dispositivo (*device drivers*) em baixo nível.

Estes atores estão sujeitos a cinco casos de uso:

- Criar sinergia: considerando que todos os dispositivos XIRP e não-XIRP se registram no controlador de célula robótica, fornecendo suas descrições funcionais, este está apto a criar a correlação entre dispositivos considerando o ambiente controlado. Este caso de uso não foi desenvolvido no contexto desta pesquisa.

- Tratar mensagem XIRP: responsável pela codificação/decodificação de

mensagens XIRP, com base nas funcionalidades disponibilizadas pelo *proxy* de dispositivo.

- Carregar configuração: considerando a decodificação da mensagem XIRP destinada a um dispositivo, esta pode ser "traduzida" para os comandos específicos de dispositivo. Estes comandos contém, entre outros, informações sobre o controlador do dispositivo (módulo de *kernel*, seção 4.5.4) a ser carregado quando solicitado.

- Controlar dispositivos: execução dos comandos de dispositivo obtidos a partir da mensagem XIRP recebida. Estes comandos foram previamente carregados.

- Fornecer *status* da célula robótica: com base em arquivos de *log* dos dispositivos, é possível disponibilizar ao operador a situação corrente da célula robótica. Estes registros podem ser obtidos a partir da observação e rastreamento das atividades de dispositivos, com o intuito, por exemplo, de detecção ou predição de erros.

Considerando este cenário, serão apresentados os resultados experimentais decorrentes da implementação de um controlador de célula robótica, por meio do uso de um sistema de controle robótico aberto, voltado a atualização de um robô IBM 7545 SCARA [46]. A implementação será discutida do nível mais baixo (i.e. próximo ao dispositivo) ao nível mais alto (acima do sistema operacional, sobre o *framework* de controle). O trabalho de recuperação realizado sobre o robô é similar àquele apresentado em [47], no qual os *drives* de potência foram mantidos, porém a implementação original incluindo a parte eletrônica e software, foram completamente atualizadas.

O software pretendido para o controlador de célula robótica é inteiramente baseado em código aberto, usando um sistema operacional Linux adaptado à operação em tempo real, o qual integra novos dispositivos por

meio de módulos de *kernel* (i.e. núcleo do sistema operacional). A principal contribuição deste sistema é o suporte a inserção de novas funcionalidades (i.e. dispositivos) à medida que estas se tornam disponíveis, com mínima perturbação à operação do sistema. A interação em espaço de usuário foi feita utilizando o *framework* de controle robótico Orocos[26].

[26]Disponível em: `http://www.orocos.org/`. Acesso em: 13/09/2010.

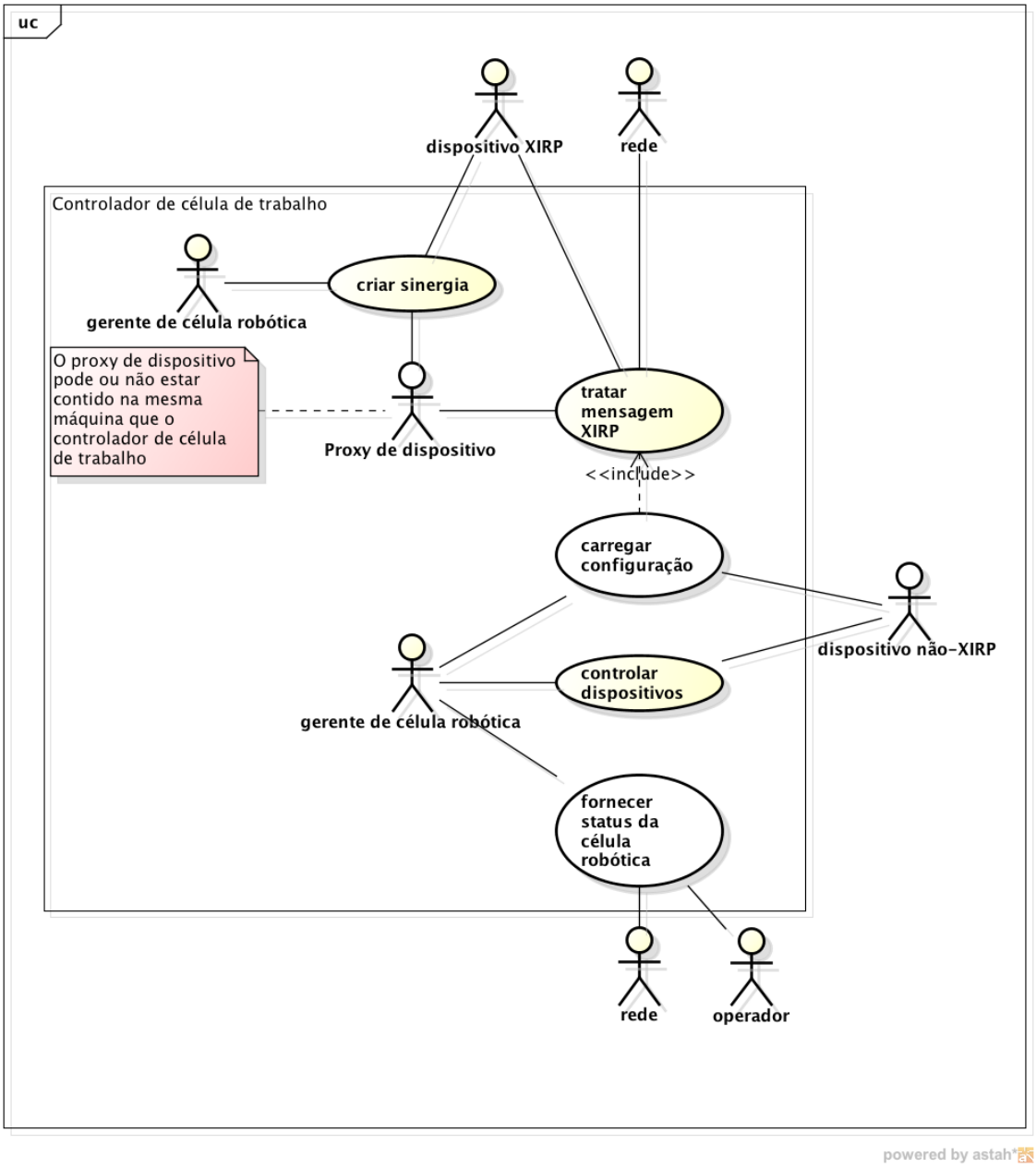

Figura 4.10: Controlador de célula robótica.

A seguir serão apresentadas considerações relativas ao grau de abertura de sistemas de controle robóticos (seção 4.5.2), discussão que deve fundamentar a escolha do *framework* Orocos para a implementação deste subsistema. O cenário de aplicação experimental para o controlador de célula robótica será apresentado na seção 4.5.4.

4.5.2 Níveis de abertura

Durante a última década, a abertura de sistemas de controle tem sido tratada de várias maneiras, considerando projetos de pesquisa no campo de máquinas ferramentas (e.g. OSEC/JOP, OMAC, OSACA [48]) e no campo de robótica (e.g. OROCOS [49]). Embora um tratado universal com relação a definição de arquitetura de controle aberta (*Open Architecture Control* – OAC) ainda não tenha sido alcançado, um conjunto básico de requisitos pode ser descrito e resumido da seguinte forma [50]:

- um OAC permite a portabilidade de software para diferentes plataformas de hardware e sistemas operacionais;

- um OAC é modular possuindo a capacidade de substituição de módulos antigos por novos, permitindo a classificação (*scalability*) em desempenho e funcionalidade;

- um OAC permite a fácil adição de novas funcionalidades. Ele é baseado em padrões para deixar que terceiros desenvolvam hardware e software que atendam a novos requisitos e ofereçam mecanismos *plug and play* para a integração rápida;

- um OAC é reconfigurável oferecendo mecanismos para a fácil adaptação de seus parâmetros com o intuito de personalizar o sistema para diferentes cenários de aplicação;

- um OAC é econômico desde que seja baseado em módulos de hardware e software comercialmente disponíveis (*off-the-shelf*).

Um ponto chave de OACs modernos é o uso de software de código aberto (*open source software* ou OSS), o qual tem sido usado no campo de controladores numéricos para máquinas ferramentas [51]. Um dos usos mais promissores de OSS em OAC é em nível de sistema operacional, o qual apresenta como vantagem a total disponibilidade do código fonte, o que permite aos desenvolvedores a criação de suas próprias particularizações e, além disso, a redução de custo do controlador como um todo, considerando custos de licença com sistemas operacionais de tempo real comerciais (*Real Time Operating Systems* ou RTOS) [50].

4.5.3 Plataformas OAC

Existem atualmente algumas arquiteturas e sistemas caracterizados por vários níveis de abertura. Esta seção não pretende servir como uma ampla pesquisa em OACs, mas irá apresentar algumas iniciativas, com grau de abertura crescente. As abordagens reunidas aqui variam quanto ao foco proprietário ou de código aberto.

Iniciativas orientadas a fabricante são fechadas em sua grande maioria, embora elas permitam algum nível de liberdade considerando a integração de novas características. Neste caso, o ponto principal é mascarar toda a complexidade de sistema do usuário com o objetivo de garantir a facilidade de uso e bloquear a possibilidade de perturbação de rotinas de segurança do sistema pelo usuário.

Abordagens orientadas a código aberto possuem foco em modularidade, considerando a possibilidade de incluir novas características sob demanda. Neste caso, um conhecimento completo do sistema é necessário.

Considerando o perfil do desenvolvedor, a última abordagem representa a tendência de institutos de pesquisa hoje.

4.5.3.1 *KUKA robot sensor interface*

O *Robot Sensor Interface* (RSI) da KUKA [52] é um exemplo de sistema robótico "semi-aberto" o qual usa uma abordagem modular para integrar sensores em um sistema de controle. Sua "abertura" reside na inclusão de chamadas de sistema de alto nível usando uma biblioteca de objetos padrão. Diferente do caso usual, no qual o sistema do sensor é conectado por meio de interfaces externas, aqui, o sensor é integrado usando a linguagem KUKA *Robot Language* (KRL). A aplicação do sensor é implementada usando esta linguagem. Este sistema é considerado semi-aberto por manter algumas partes em um "nível de proteção de desenvolvedor".

Embora o RSI permita um grau moderado de liberdade considerando a integração de sensores suportados, os controladores (*drivers*) do sensor estão fechados sob o ponto de vista do usuário, como forma de prevenir que estes perturbem as medidas de segurança implementadas para o sistema. De certa forma, é uma maneira de proteger o usuário de si mesmo.

4.5.3.2 *Open robot control architecture e autonomous reactive robot planning*

Considerando um nível de abertura maior, o Open Robot Control architecture (ORC) [47], criado no Lund *Institute of Technology*, desenvolveu uma arquitetura hierárquica que integra programação *on-line* e *off-line* em um nível mais alto, criando vistas de usuário especializadas.

Em níveis mais baixos, ele manteve os *drives* de potência originais e funções de segurança do robô protótipo (um sistema ASEA Irb-2000/S3b da ABB) de modo que ele pudesse ser portado mais facilmente para um ambi-

ente industrial. A infra-estrutura para a integração de sensor foi desenvolvida considerando esta arquitetura como base.

O *Autonomous Reactive Robot Planning* (ARS) [53], desenvolvido no Department of Mechanical Engineering da Lund *University*, propôs uma abordagem para a integração *on-line* entre uma célula robótica simulada (incluindo sensores simulados) e a célula robótica real, permitindo a retroalimentação de dados lidos para o modelo simulado. Assim, é possível inserir novos parâmetros que não foram previstos na programação *off-line*, permitindo a adaptação da célula robótica real mesmo após a carga dos programas de controle.

4.5.3.3 Generator of modules – GenoM

O *Generator of Modules* (GenoM) [54] é um ambiente para a descrição e implementação de componentes de software robóticos que oferece o seguinte:

- Modelo de componente: o GenoM define a interação específica de componentes e padrões de composição;
- Implementação do modelo de componente: o GenoM oferece um conjunto dedicado de elementos de software executáveis necessário para suportar a execução de componentes de software que estão em conformidade com o modelo;
- Arquitetura de componente: o GenoM define a arquitetura interna de componentes de software, sua estrutura e funcionamento;
- Ferramentas para geração de componentes: o GenoM oferece um conjunto de ferramentas para descrever componentes de software e para gerar modelos (*templates*).

O processo de geração é baseado em um arquivo de descrição de componente que é interpretado pela ferramenta. Assim, são gerados *templates*

que podem ser usados por desenvolvedores especializados para a implementação de algoritmos. Finalmente, componentes binários são gerados, em tempo real ou não, juntamente com programas de teste.

4.5.3.4 Projeto Orocos

O projeto Orocos começou como um software de código aberto para robótica [55], mas superou suas raízes. "Orocos" é o acrônimo para o projeto *Open Robot Control Software*. O objetivo do projeto é desenvolver um software de código aberto, na figura de um *framework* modular para controle de robôs e máquinas ferramentas. O projeto Orocos disponibiliza quatro bibliotecas em C++: o *Real-Time Toolkit* (RTT), a *Kinematics and Dynamics Library* (KDL), a *Bayesian Filtering Library* (BFL) e a *Orocos Component Library* (OCL) [49].

O RTT não é uma aplicação em si, mas oferece a infra-estrutura e funcionalidades para construir aplicações robóticas em C++. A ênfase é em aplicações de tempo real, *on-line* e interativas. A OCL oferece alguns componentes de controle prontos para usar. Componentes para gerenciamento e componentes para controle e acesso a hardware estão disponíveis. A KDL é uma biblioteca em C++ a qual permite o cálculo de cadeias cinemáticas em tempo real. A BFL oferece um *framework* independente de aplicação para inferência de redes de Bayes dinâmicas.

A seção 4.5.4 apresenta o cenário de aplicação e, com base no que foi exposto quanto a plataformas OAC, faz uma escolha quanto a plataforma a ser utilizada.

4.5.4 Cenário de aplicação

O estágio atual de desenvolvimento do controlador de célula robótica para o robô IBM 7545 SCARA baseia-se em um esquema hospedeiro-alvo

(*host-target*). O software foi desenvolvido em uma estação Linux e enviado para a máquina alvo executando um *Real Time Linux* (RTL) baseado na extensão de *kernel* RTAI.

Um RTL, mais especificamente baseado no RTAI, é bastante adequado para controlar um robô com seis graus de liberdade [56], com pouco retardo de escalonamento (ou *scheduling jitter*), inferior a 10 μs no pior caso. Usando um estudo comparativo feito por [57], é possível concluir que o RTAI pode ser preferencialmente usado ao invés de outros RTLs devido ao seu melhor desempenho em condições de sobrecarga. Embora as indicações usadas atualmente para a escolha do RTAI sejam apenas teóricas e empíricas, esses resultados deverão ser averiguados com maior rigor em um momento futuro, considerando a plataforma em uso.

Um PC industrial foi usado para hospedar o sistema de tempo real, consistindo de um *rack* CompactPCI, com as seguintes placas:

- Placa de CPU Inova AMD K6-2 de 266 MHz;

- *Carriers* da Acromag: os *carriers* CompactPCI são placas que agem como uma ponte entre os *Industry Packs* (IPacks) e o barramento CompactPCI no PC industrial. Existem duas *carriers* com dois *slots* cada;

- Placa de I/O *National Instruments*: placa CompactPCI com 48 entradas analógicas e digitais usada para ler dados fornecidos pelos sensores de fim de curso e futuramente, pelo sensor de força a ser posicionado no punho do robô.

Os IPacks usados foram:

- IPack com 48 I/Os Digitais da Tews: IPack com entradas e saídas digitais para ler dados provenientes de sensores e fornecer dados digitais como saída;
- IPack *Quadrature* da Tews: IPack para ler os valores dos *encoders* e informar ao software a posição dos eixos do robô;
- IPack com conversor analógico digital (*Digital to Analog Converter* ou DAC) da Tews: IPack para converter dados digitais em sinais analógicos, fornecendo informação para controlar os motores.

Outros ítens usados no procedimento de reestruturação (*retrofit*) foram:

- Interface de energia: placas de energia para realizar a interface entre os I/Os digitais dos IPacks com os motores do robô;
- Robô: um robô IBM 7545 SCARA que sofreu o processo de reestruturação.

Também foram desenvolvidos *drivers* de software para acessar as *carriers* e os IPacks posicionados em seus *slots*, com o intuito de ativar os motores do robô, ler os *encoders* e sensores, além de adaptações para a operação com o PC industrial. Uma representação do ambiente de desenvolvimento pode ser encontrada na figura 4.11.

4.5.4.1 Drivers do kernel

Antes de discutir a abordagem usada para criar um sistema de controle de célula robótica modular usando um OSS de tempo real e módulos de *kernel* do Linux (*Linux Kernel Modules* ou LKMs), é necessário definir alguns termos. Como o sistema operacional possui uma camada de abstração

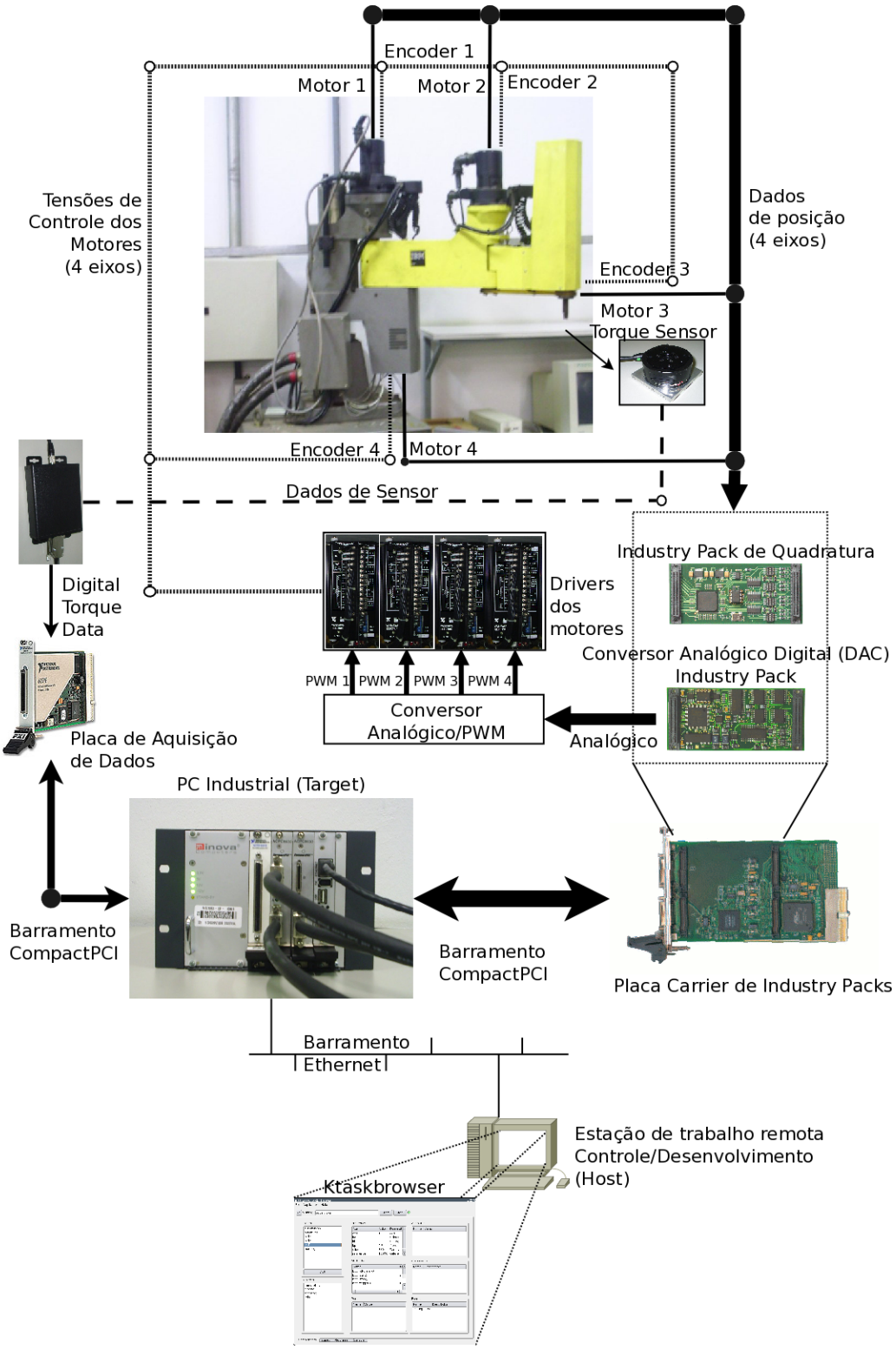

Figura 4.11: Representação do ambiente experimental de desenvolvimento para controle do IBM 7545 SCARA [6].

de hardware e deve apresentar compartilhamento de recursos, todo o acesso do hardware a partir de programas de usuário deve ser feito por meio do SO. Para reforçar esta cadeia de acesso, de modo a prevenir que aplicações maliciosas ou mau codificadas danifiquem o hardware, uma proteção em nível de hardware faz-se necessária [58].

O hardware deve prover pelo menos dois níveis de execução:

- Modo de *kernel*: neste modo, o software tem acesso a todas as instruções e a cada dispositivo de hardware.

- Modo de usuário: neste modo, o software é restrito, não pode executar algumas instruções e é negado o acesso a alguns dispositivos de hardware, como alguma área da memória principal ou acesso direto ao barramento IDE.

Assim, em nível de software, dois espaços são definidos:

- Espaço de *kernel*: código sendo executado no modo de *kernel* está localizado dentro do espaço de *kernel*.

- Espaço de usuário: qualquer outro programa, sendo executado em modo de usuário, está localizado em espaço de usuário.

A decisão por usar LKMs está relacionada à confiabilidade do espaço de *kernel* ao lidar com tarefas de tempo real implementadas pela extensão de *kernel* RTAI.

4.5.4.2 Comunicação entre módulos de kernel

O *kernel* do Linux fornece uma maneira simples e prática para realizar a comunicação transparente entre módulos do *kernel*. Cada módulo escrito pode 'exportar' uma ou mais funções para outros módulos de *kernel*, simplesmente adicionando a macro `EXPORT_SYMBOL` ao final do código fonte.

Quando um módulo exporta uma função que é usada por outros módulos, uma dependência de módulo é automaticamente criada, bloqueando ao usuário a habilidade de remover um módulo que está em uso por outro. Nesta situação, primeiro o módulo que usa recursos disponibilizados por um módulo fornecedor deve ser removido, e em um segundo passo, o módulo que fornece a função poderia ser removido. Em um SO Linux, uma lista de símbolos do *kernel*, incluindo os módulos exportados, pode ser obtida a partir de qualquer módulo, por meio do comando `'cat /proc/ksyms'`.

4.5.4.3 Drivers de dispositivo baseados em LKM e acesso em tempo real

Em programas implementados como LKMs, todas as tarefas de tempo real são implementadas como processos em execução dentro do *kernel* do Linux como módulos. Os módulos de tempo real podem ser facilmente iniciados ou interrompidos com comandos simples como `'insmod'`, que instala um novo módulo no kernel em execução, `'rmmod'`, o qual remove determinado módulo e `'lsmod'` que lista todos os módulos carregados no momento. O aspecto mais interessante desta abordagem é que o sistema pode admitir modificações em seu comportamento, representada pela inserção de novos módulos, em tempo de execução, sem o seu desligamento.

O desenvolvimento dos *drivers* de baixo nível para acessar o hardware foi iniciado em [46]. O driver de dispositivo da placa *carrier* (`apc8620`) e a camada de abstração de hardware (*Hardware Abstraction Layer* ou HAL) para os IPacks (`carrier1_hal` e `carrier2_hal`) foram desenvolvidos como prova de conceito. O `carrier1_hal` e o `carrier2_hal` eram LKMs monolíticos os quais incorporavam os *drivers* para o IPack *quadrature*, IPack DAC (`carrier1_hal`) e para o IPack Digital com 48 I/Os (`carrier2_hal`).

Na primeira implementação, a comunicação entre espaço de usuário/*kernel* foi conseguida usando um recurso para implementar filas chamado *First In First Out* (FIFO) de tempo real, o qual cria um caminho de comunicação

entre os dois modos operacionais. Assim, o FIFO deveria ser implementado em espaço de usuário e espaço de *kernel*. Um módulo `scara_fifo` foi implementado para receber comandos a partir de programas em espaço de usuário, os quais enviam dados inseridos por usuários aos processos de tempo real, como por exemplo, uma interrupção para iniciar a leitura de um *encoder*, uma tensão para iniciar o movimento do motor etc.

Para provar a modularidade (*scalability*) do conceito de LKM, foi desenvolvido um controle para as leituras de fim de curso, as quais são mapeadas no IPack digital e um controlador proporcional - integral - derivativo (PID) para cada junta. Foram especificadas resoluções de 5μs para o PID e 20 μs para o monitor de fim de curso. Embora no contexto deste trabalho a precisão do sistema de tempo real não tenha sido formalmente validada, em [59], foram realizados testes com uma plataforma similar (PC com processador Pentium II 400MHz, 256MB de memória), obtendo-se tempo máximo de resposta de até 5μs.

Uma restrição observada na infra-estrutura implementada foi a ausência de uma maneira simples e padronizada para a interação do usuário com o ambiente de controle robótico. Neste momento, o sistema exigia que o programador conhecesse a implementação das rotinas do *kernel* para poder desenvolver programas em espaço de usuário.

A resposta mais óbvia para lidar com esta situação foi criar um conjunto padronizado de funções primitivas para acessar os IPacks e tarefas comuns, como procedimentos de calibração inicial, controladores para as juntas incluindo PIDs, redes neurais etc. Dessa forma, seria possível alcançar um modelo de acesso simplificado do sistema para o usuário.

Considerando as abordagens apresentadas na seção 4.5.3, a mais adequada para implementar a biblioteca de usuário baseia-se em um *framework* aberto e padronizado. O GenoM e o Orocos foram as escolhas mais apropriadas nessa categoria. O Orocos foi escolhido por sua extensa docu-

mentação e classificação como sistema de código aberto, o que torna o acesso as suas funções mais simples.

Considerando a filosofia proposta pelo Orocos foi necessário realizar algumas adaptações no sistema original. Essas adaptações consistiram na modularização dos *drivers* de *kernel* (módulos HAL) dos IPacks e na criação de interfaces entre espaço de *kernel* e usuário para acesso a dispositivos a partir dos *templates* específicos do Orocos. Na versão original, a simples troca de IPacks entre *slots* da *carrier* poderia representar um problema. Isto ocorria porque a inicialização dos IPacks havia sido codificada diretamente nas rotinas de início da *carrier* em um único *driver* de *kernel*.

A nova versão implementa um módulo de *kernel* separado para cada IPack. Cada módulo é responsável por auto-detectar se o IPack adequado está inserido no *slot* correto da *carrier*. Desta maneira, agora é possível trocar os IPacks entre os *slots* da *carrier* sem a modificação detalhada do código fonte. As bibliotecas de espaço de *kernel* e as funções disponíveis são as seguintes:

- `apc8620_hal`: possui um conjunto de funções primitivas responsável pela abstração de hardware (HAL) considerando as placas *carriers*. Ao ser carregado, este módulo realiza a inicialização das *carriers* e disponibiliza funções por meio da macro `EXPORT_SYMBOL` para a interface com os módulos de *kernel* dos IPacks.

- `ip-dio48-et`: verifica se, para um determinado *slot* da *carrier*, o IPack digital está presente. Caso esteja presente, disponibiliza funções de entrada (`get_input`) e saída (`outbyte`) por meio da macro `EXPORT_SYMBOL`.

- `ip-optodaq`: verifica se, para um determinado *slot* da *carrier*, o IPack analógico está presente. Caso esteja presente, disponibiliza uma função de saída (`output_dac`) e um relê lógico (`relay`) por meio da macro `EXPORT_ SYMBOL`.

- `ip-quadrature`: verifica se, para um determinado *slot* da *carrier*, o *encoder* está presente. Caso esteja presente, disponibiliza uma função para leitura (`read_enc`) e uma função para iniciar a contagem de pulsos (`reset_enc`) por meio da macro `EXPORT_SYMBOL`.

4.5.4.4 Interface de espaço de usuário via LXRT

A segunda modificação implementada no código original para controle do robô IBM 7545 SCARA refere-se à forma pela qual programas em espaço de usuário acessavam funções de espaço de *kernel*. O Orocos usa o *LinuX RealTime* (LXRT) para implementar a interface entre aplicações em espaço de usuário e *drivers* de dispositivo em espaço de *kernel*. Assim, a versão original usando FIFOs de tempo real foi adaptada para lidar com este pré-requisito. O mapeamento das funções de espaço de espaço de *kernel* para o espaço de usuário é conseguido por meio do arquivo de cabeçalho `lxrt_user.h` e do módulo de *kernel* `lxrt_kernel.ko` (módulo carregável de kernel ou *loadable kernel module* – LKM). O LKM `lxrt_kernel.ko` é um arquivo objeto contendo referência ao código relocável das principais funções (disponíveis em códigos objetos) de baixo nível para a operação do robô IBM 7545 SCARA (e.g. função para operação de um relê para ligar/desligar o PC industrial, funções para reinício/leitura dos *encoders*, função para envio de dados no IPack DAC, função para leitura dos canais digitais).

O arquivo de cabeçalho `lxrt_user.h` cria uma abstração para os *drivers* de baixo nível de modo que as funções primitivas de *kernel* possam ser acessadas como uma API padrão (listagem 4.5). Os principais parâmetros são:

- `IP_DIGITAL_READ` que é um índice que identifica unicamente a função de espaço de *kernel* `get_input`, definida em `lxrt_kernel.c` na listagem 4.6;

- `&arg` que representa o endereço da estrutura contendo o argumento; neste caso, um valor qualquer considerando que esta função não possui um parâmetro passado; e

- `i[LOW]` que é um tipo de dado de retorno do `rtai_lxrt`, um módulo de *kernel* que disponibiliza chamadas da API LXRT.

O mapeamento das funções de espaço de *kernel* é feito pelo módulo de *kernel* `lxrt_kernel.ko`. Esse módulo implementa a interface com as funções de espaço de usuário disponíveis em `lxrt_user.h`. No exemplo apresentado na listagem 4.6 mostra-se como a função de espaço de *kernel* `get_input`, usada para leitura no IPack digital e mapeada para a chamada de espaço de usuário `IP_Digital_read()`. Para tanto, é criado um vetor de estrutura `rt_fun_entry` onde cada elemento realiza o mapeamento para uma função. A constante `NON_RT_SWITCHING` ajustada para `0` é responsável por especificar que este é um processo de tempo real *soft*.

Listagem 4.5: Trecho do arquivo de cabeçalho `lxrt_user.h` com os mapeamentos de métodos para as funções do espaço de *kernel*.

```
(...)
static inline int IP_Digital_read ()
{
  int retval;
  struct {unsigned int dummy; } arg = { 0 };
  retval = rtai_lxrt(MYIDX, SIZARG,
  IP_DIGITAL_READ , &arg).i[LOW];
  return retval;
}
(...)
```

Listagem 4.6: Trecho do `lxrt_kernel.c` com os mapeamentos das funções do espaço de *kernel*.

```
#define NON_RT_SWITCHING 0
(...)
extern int get_input (void);
(...)
static struct rt_fun_entry rt_fun[] = {
(...)
    [ IP_DIGITAL_READ    ] = {NON_RT_SWITCHING, get_input},
(...)
};
```

Considerando que os demais métodos são mapeados de maneira análoga, a infra-estrutura para a implementação dos métodos de espaço de usuário para acessar o hardware do robô IBM 7545 SCARA está parcialmente completa. Uma função simples para executar a calibração inicial do robô, assim como controladores PID foram criadas como demonstração para alguns dos princípios de programação do Orocos.

Para exemplificar uma tarefa e suas características, neste caso métodos (*methods*) e eventos (*events*), foi especificada uma rotina de calibração (*homing*) usando as bibliotecas do Orocos. O modelo (*template*) para a definição desta rotina de calibração está disponível no arquivo de cabeçalho `HomingInterface.hpp` do Orocos.

Também foram criadas funções primitivas de sistema, incluindo funções para leitura periódica do IPack digital (`IP_DInPeriodicRead.hpp`), envio de sinal de saída para o IPack analógico (`IP_AOutInterface.hpp`) e interface com o *encoder* (`IP_EncoderInterface.hpp`). As principais funcionalidades criadas executam o procedimento de calibração (`HomePosition.hpp`). Para tanto, gera-se um evento do tipo "*home switch*", o qual indica que o sis-

tema foi calibrado e retorna o status de calibração ou *reset* quando desejado. Tudo isso é implementado usando uma linguagem semelhante ao C++, característica do *framework*.

Outro exemplo prático implementado foi um controlador PID empírico para testes (`PID_Control.hpp`) usando as funções criadas para interface com o IPack analógico e o IPack para leitura de quadratura. Esse controlador aceita o ajuste de parâmetros (`setpoint` , `k_p`, `k_i`, `k_d`) sob demanda pelo usuário. Dessa forma, para ensaios em laboratório em disciplinas de controle, esta pode ser uma aplicação interessante, tendo em vista que a planta robótica ainda não é completamente conhecida.

Outra possibilidade estudada foi a definição de um arquivo com extensão '`.cpf`' para a carga de propriedades existentes em uma tarefa usando uma linguagem similar ao XML. Assim, o estado inicial da tarefa pode ser definido de maneira simples pela modificação de valores nesse arquivo. Por exemplo, tanto no caso do PID empírico quanto na função para calibração, existem parâmetros pré-configurados, que foram obtidos a partir de experimentos anteriores (listagem 4.7).

Considerando a possibilidade de criação de uma máquina de estado finito para representar o comportamento de um sistema, toda a base para a construção de métodos e programas para o controle do robô IBM 7545 SCARA está disponível.

4.5.4.5 Comunicação em rede

Após o desenvolvimento das funções primitivas para a operação do sistema, foram realizados experimentos para a implantação da infra-estrutura de comunicação em rede. Para tanto, foi utilizado o suporte nativo a CORBA (*Common Object Request Broker Architecture*) implementado pelo Orocos. O suporte a CORBA depende da instalação correta das bibliotecas ACE (*Adaptive*

Communication Environment) e TAO (*The ACE ORB*)[27] e compilação adequada do Orocos. Além disso, é necessário iniciar o servidor de nomes do CORBA (*Corba Naming Service*) na plataforma alvo e compilar as aplicações com as *flags* corretas.

Listagem 4.7: Exemplo de arquivo para a carga de propriedades para o PID da primeira junta (próxima a base) do robô IBM 7545 SCARA.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "cpf.dtd">
<properties>
<simple name="axis" type="ulong">
        <description>Axis index</description><value>1</value></simple>
<simple name="offset" type="ulong">
        <description>Offset</description><value>360</value></simple>
<simple name="kp" type="double">
        <description>Kp</description><value>0.3</value></simple>
<simple name="ki" type="double">
        <description>Ki</description><value>0</value></simple>
<simple name="kd" type="double">
        <description>Kd</description><value>0</value></simple>
</properties>
```

Embora a comunicação em redes Ethernet possua sérias restrições quanto a execução em tempo real, foi possível obter resultados preliminares interessantes desde que respeitadas as seguintes regras [60][28]:

[27]Disponíveis em `http://www.cs.wustl.edu/~schmidt/ACE.html` (acesso em 04/10/2010).

[28]Nesse contexto, não são considerados projetos como o RTnet, o qual redefine a pilha de protocolos de rede como sendo de tempo real estrito, por meio do uso de extensões do *kernel* do Linux, como o Xenomai e o RTAI. Maiores informações em: `http://www.rtnet.org/index.html`. Acesso em: 15/12/2010.

- Os componentes não devem chamar outros componentes remotos durante execução em tempo real. Entretanto, componentes de tempo real podem ser chamados por quaisquer componentes, locais ou remotos. Dessa forma, um componente pode receber sempre um pedido de qualquer outro; todavia, nem todo componente pode enviar um pedido a qualquer outro componente. Caso esta regra seja violada o programa não irá falhar, mas seu tempo de execução pode ser severamente comprometido.

- Caso o método `CORBA.shutdown()` seja chamado por um componente, toda a parte CORBA daquele executável é terminada. Isso causará o retorno da função `RunOrb()`.

- É necessário exportar pelo menos um componente. Exportar um componente, neste contexto, significa exportar uma interface de acesso aos seus métodos via CORBA. Caso se tente o acesso a um componente par pela rede, isto será detectado e um componente par deste também será automaticamente exportado. Isto significa que caso cada par seja visitado, todos os seus componentes par serão conhecidos na rede.

Para disponibilizar via rede as funções criadas em espaço de usuário para interação com o robô IBM 7545 SCARA, foi necessário criar um programa principal com suporte à biblioteca do Orocos `ControlTaskServer.hpp` e a declaração do *namespace* adequado (`using namespace RTT::Corba`). A partir deste momento, todas as declarações de tarefas passam a ser disponibilizadas via *Interface Definition Language* (IDL) CORBA[29].

Foi criada uma tarefa de supervisão que foi ligada via método `connectPeers()` às tarefas de calibração inicial, PID e às funções primitivas de operação do robô IBM 7545 SCARA. O procedimento de conexão para

[29]O Orocos não requer conhecimento em IDL ou CORBA por parte do usuário quando dois componentes Orocos se comunicam. Entretanto, caso se deseje acessar um componente Orocos a partir de um programa não-Orocos, como uma interface gráfica de usuário (*Graphical User Interface* ou GUI) do Windows, é necessário usar os arquivos de IDL do Orocos.

estabelecimento do fluxo de dados entre uma tarefa para controle usando PID e uma tarefa de supervisão é mostrada na listagem 4.8.

Listagem 4.8: Trecho do `scara_sv.cpp` com os mapeamentos da função para o PID1 e estabelecimento do fluxo de dados com a tarefa de supervisão.

```
(...)
// PID para a junta 1 e leitura de constantes a partir do arquivo
// pid1.cpf
PID_Control _pid1("pid1","cpf/pid1.cpf");
// O PID é criado como uma atividade periódica, de alta prioridade,
// com um período de 0.005 segundos e acesso ao motor de execução
PeriodicActivity _pid1_activity(RTT::OS::HighestPriority,
                                0.005, _pid1.engine());
// Tarefa de supervisão executada como uma tarefa periódica de alta
// prioridade com um período de 0.01 segundos e acesso ao motor de
// execução
TaskContext _supervision("supervision");
PeriodicActivity _supervision_activity(RTT::OS::HighestPriority,
                                       0.01,_supervision.engine());
// conexão de portas da tarefa de supervisão e do PID
connectPeers(&_supervision, &_pid1); (...)
```

Para inicializar a interface ORB (*Object Request Broker*), basta chamar o método `InitOrb()` e `Create()` disponibilizados por `ControlTaskServer`. Ao disparar a tarefa de supervisão (`_supervision_activity.start()`) e executar o ORB (`ControlTaskServer::RunOrb()`), os métodos das tarefas estarão disponíveis via rede (listagem 4.9).

Para executar no PC industrial o programa descrito, foi criado um *script* que inicia o serviço de nomes do TAO e posteriormente inicia o servidor (listagem 4.10). Nesse caso, o serviço é iniciado sem a opção de *multicasting*

(`-m 0`) e foi inserido um endereço e porta para a interface baseado no protocolo *Internet Inter-ORB Protocol* (IIOP). O serviço de nomes irá operar em segundo plano (`-ORBDaemon`). Além disso, o servidor de aplicação Orocos é iniciado e é realizada uma correlação deste com o endereço atrelado ao serviço de nomes.

O lado cliente pode ser executado em qualquer máquina. Para tanto, basta criar uma variável de ambiente que aponte exatamente para o endereço no qual o servidor está atrelado (listagem 4.11).

Listagem 4.9: Trecho do `scara_sv.cpp` contendo o procedimento para disponibilização de funções do PID1 em rede via CORBA.

```
(...)
// Ajuste do CORBA e exportação das funções da tarefa
ControlTaskServer::InitOrb (argc, argv);
ControlTaskServer::Create (&_supervision);
_supervision_activity.start();
// Espera por requisições
ControlTaskServer::RunOrb ();
_supervision_activity.stop();
ControlTaskServer::ShutdownOrb ();
// Limpeza do Corba
ControlTaskServer::DestroyOrb ();
(...)
```

A interação com o usuário e acesso às funcionalidades dos componentes são feitas por meio de uma interface gráfica fornecida pelo Orocos denominada `KTaskBrowser`. Essa interface é baseada no ambiente gráfico K *Desktop Environment* (KDE) 3 e permite inspecionar o estado de variáveis e o progresso de *scripts* e máquinas de estado em componentes. Ele se conecta a componentes Orocos, disponíveis em rede, por meio do serviço de nomes do CORBA que permite a navegação do serviço (figura 4.12).

Listagem 4.10: *Shell* script para chamar o serviço de nomes e o programa servidor no PC industrial.

```
#!/bin/sh
# Início do serviço de nomes e especificação do endereço IP
# e porta.
Naming_Service -m 0 -ORBListenEndpoints iiop://192.168.0.41:2809
        -ORBDaemon
sleep 1
./server -ORBInitRef NameService=
        corbaloc:iiop:192.168.0.41:2809/NameService
```

Listagem 4.11: Criação de variável de ambiente no cliente para acesso às tarefas disponibilizadas pelo Orocos.

```
NameServiceIOR="corbaloc:iiop:192.168.0.41:2809/NameService"
export NameServiceIOR
```

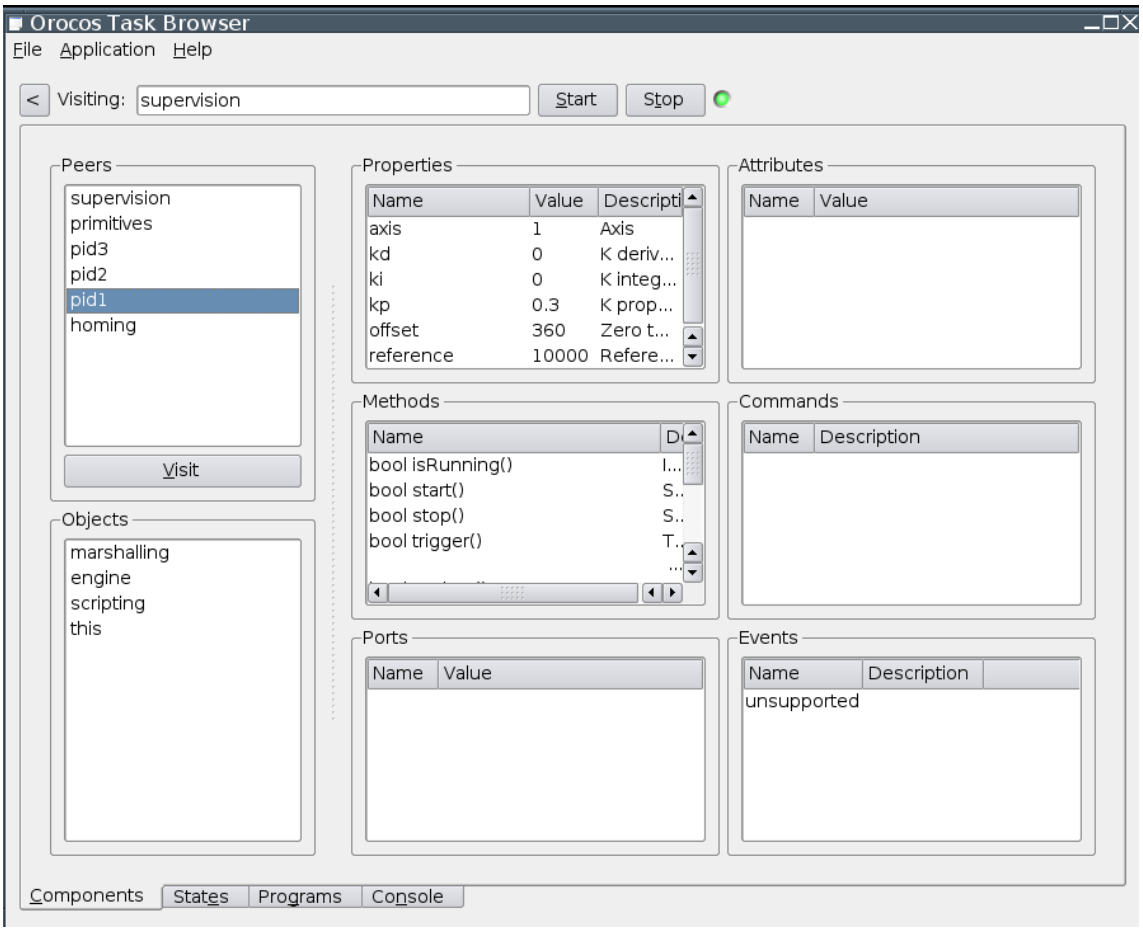

Figura 4.12: Interface `KTaskBrowser` para os componentes implementados até o momento.

4.5.5 Limitações atuais

Embora tenha sido possível atingir resultados suficientes para provar que o Orocos é uma abordagem viável para a continuidade deste projeto, algumas limitações foram encontradas, considerando o sistema de tempo real subjacente.

O uso do LXRT incorre em uma latência de escalonamento adicional. Observa-se que, mesmo sob esta circunstância, considerando um ambiente de desenvolvimento conforme o apresentado no contexto deste projeto, esta sobrecarga extra é compensada pela possibilidade de desenvolvimento estruturado e tolerante a falhas (seção 4.5). Nesse caso, a maior vantagem é a possibilidade de desenvolvimento em tempo real *soft*, no qual uma im-

plementação defeituosa não irá desabilitar o sistema em sua totalidade. Isso permite a finalização de qualquer processo fora de controle sem a necessidade de reinício do sistema.

Outras variantes de Linux para tempo real exigem que as tarefas de tempo real sejam desenvolvidas como módulos do *kernel*, restringindo as ferramentas que podem ser usadas para desenvolvimento e exigindo maiores privilégios para o usuário.

Em [61], as medidas tomadas em um hardware modesto (Intel PII 300 Mhz) mostraram uma sobrecarga adicional de 3μs sobre a latência escalonar[30] média de 3μs.

Considerando o ambiente desenvolvido no contexto deste doutorado, ainda é necessário realizar tomadas de tempo para a validação prática e um posterior comparativo com os resultados presentes na literatura pesquisada.

Algumas questões envolvendo o RTAI, as quais ainda não são muito bem documentadas, tornam difícil a interpretação de seu impacto no sistema como um todo. A carga média no sistema (ou *load average*) usando o módulo de *kernel* LXRT é muito alta, com valores variando entre cinco e sete; cinco sendo a carga inerente do módulo de *kernel* e a carga adicional sendo uma característica do processo de tempo real. Este é um fato bem conhecido, porém relativamente inexplorado na comunidade RTAI.

A única informação encontrada explica que estes processos estão em um estado `UNINTERRUPTIBLE_SLEEP`, característico de processos de I/O, disparado por alguns *daemons* internos iniciados pelo LXRT (`kthread_b`, `kthread_m`[31]). Esses *daemons* estão esperando por requisições de processos

[30]No contexto do *kernel* do Linux, latência escalonar é o intervalo entre a sinalização de um *wakeup* (estímulo) indicando que um evento ocorreu e o escalonador do *kernel* conseguir uma oportunidade para escalonar a *thread* que está esperando a ocorrência do *wakeup* (a resposta). *Wakeups* podem ocorrer devido a interrupções de hardware, ou provocado por outras *threads* [62].

[31]`kthread_b` é responsável pelo primeiro escalonamento de qualquer objeto apto a ser escalonado que passe a trabalhar em tempo real *hard*. `kthread_m` é responsável por cri-

em estado TASK_UNINTERRUPTIBLE. Como conseqüência, a espera acumulada é qualificada como sendo a carga real pelo Linux[32]. Nesse caso particular, a árvore de processo mostra o kthread_m iniciando quatro processos no estado UNINTERRUPTIBLE_SLEEP (listagem 4.12).

Listagem 4.12: Trecho da árvore de processo gerada pelo pstree.

```
init-+-RTAI_KTHRD_M:0-+-F:HARD:0:1
     |                |-F:HARD:0:2
     |                |-F:HARD:0:3
     |                `-F:HARD:0:4
(...)
```

Usando o comando ps, é possível ver cinco processos insmod rtai_lxrt.ko em estado UNINTERRUPTIBLE_SLEEP. Considerando a árvore de processo e os processos dormentes em espera listados na listagem 4.13, a relação de um processo em tempo real do usuário executando em espaço de usuário e sua trajetória exata no interior do *kernel* do sistema ainda não foi explorada no contexto deste trabalho.

Em [63], Mantegazza descreve o LXRT como um auxiliar à fase de desenvolvimento, tornando-a menos sujeita a riscos e mais rápida. Sob a ótica do *framework* Orocos, deverá ser investigado o potencial do LXRT, considerando sua confiabilidade, robustez e flexibilidade, sendo essas, características imprescindíveis a uma plataforma de controle robótico de tempo real. Para tanto, é necessário criar testes de stress para comprovar a efetividade do sistema. Além disso, é preciso testar outras opções de sistemas de tempo real que funcionem com o Orocos. Um exemplo seria o sistema de tempo real

ar/apagar *threads* de *kernel* de tempo real *hard*, i.e. o que nos escalonadores únicos do *kernel* é chamado uma tarefa. Eles também tomam conta do reservatório de *threads*. Fonte: https://mail.gna.org/public/rtai-dev/2004-07/msg00067.html. (acesso em 04/10/2010).

[32]Fonte: https://mail.rtai.org/pipermail/rtai/2004-July/008077.html e https://mail.rtai.org/pipermail/rtai/2003-August/004634.html (acesso em 04/10/2010).

Xenomai[33]. Um trabalho futuro pode envolver a comparação entre o RTAI e o Xenomai e uma análise mais profunda da especificação interna do sistema de tempo real.

Listagem 4.13: Trecho de lista de processo gerado pelo comando `ps aux`.

```
(...) STAT START   TIME COMMAND
(...)
(...) D    12:34   0:00 insmod rtai_lxrt.ko
(...) D    12:34   0:00 insmod rtai_lxrt.ko
(...) D    12:34   0:00 insmod rtai_lxrt.ko
(...) D    12:34   0:00 insmod rtai_lxrt.ko
(...) D    12:34   0:00 insmod rtai_lxrt.ko
(...)
```

4.6 CONSIDERAÇÕES FINAIS

Esta seção apresentou os primeiros passos para a implementação de uma arquitetura P'n'P cujo foco são PMEs brasileiras. Esta hipótese é válida principalmente em questões relativas a aplicação gradual de um modelo similar ao modelo de Naumann (seção 2.3.1), considerando a necessidade pelo processamento de um protocolo de comunicação industrial como o XIRP.

A prerrogativa de uso de periféricos microprocessados tornaria uma infra-estrutura baseada em dispositivos antigos inadequada para a aplicação do padrão. Mesmo dispositivos mais novos precisariam da capacidade de processar comunicação em rede (TCP/IP) e tratar um protocolo de comunicação em nível de aplicação, considerando uma estrutura em camadas similar àquela apresentada no modelo ISO/OSI (conforme visto na seção 3.3). Mesmo em países como a Alemanha, este cenário de aplicação restritivo é problemático considerando que, em 2008, apenas 22% de todos os negócios com

[33]Maiores informações em `http://www.xenomai.org` (acesso em 04/10/2010).

pequeno volume de produção usam robôs ou infra-estrutura compatível [12]. Por fazer parte de uma iniciativa de vulto da União Européia envolvendo diversos centros de pesquisa e grandes fabricantes, estes números não são tão impressionantes (o projeto SMErobot foi concluído em maio de 2009).

Com base em um cenário de aplicação no qual dispositivos nem sempre possuem a capacidade de se adequar as exigências de uma infra-estrutura P'n'P, foi criado uma arquitetura para um sistema integrador cujo objetivo primário é permitir a incorporação de dispositivos, mesmo que estes não possuam a capacidade de processar informações conforme especificado pela norma VDMA 66430-1 [11] ou XIRP. A este sistema integrador foi dado o nome de ACID. O ACID é formado por um subsistema de gerenciamento de *tags* [40], um subsistema para *proxy* de dispositivo [41] e um ou mais subsistemas controladores de célula robótica [42].

O subsistema de *tag* indexa dispositivos e gera uma etiqueta baseada em QR Code. Assim, o operador teria o acesso rápido às informações do dispositivo, a partir de um PDA ou telefone celular dotado de câmera, incluindo o acesso a uma interface de controle disponibilizada via servidor web. Além disso, o sistema de *tag* também é responsável por armazenar e disponibilizar os arquivos de configuração XIRP necessários para a integração de um novo dispositivo.

O subsistema para *proxy* de dispositivo é o sistema responsável pela geração automática de código fonte. Para tanto, são necessários os arquivos de configuração XIRP hospedados no sistema de *tag* e especificações funcionais adicionais para a inferência da lógica de programação do dispositivo. Até o momento, o processamento de arquivos de configuração XIRP (esquemas XIRP XML) foi implementado, possibilitando o uso de métodos em Java para a composição e interpretação de mensagens XIRP. Alguma dose de pós-processamento é necessária para a composição das mensagens, porém

isto não configura um empecilho para a utilização futura desta implementação.

O subsistema controlador de célula robótica foi implementado usando o *framework* Orocos e um modelo de integração de dispositivos/funcionalidades baseado na implementação de módulos de *kernel* (LKMs). A priori, um subsistema com estas características permite a inserção de novos elementos na célula robótica, desde que exista um LKM apropriado para o mesmo.

O capítulo 5 irá apresentar as conclusões para este trabalho e propostas de trabalhos futuros com o intuito de implementação de uma arquitetura P'n'P funcional.

CAPÍTULO

5

CONCLUSÃO

5.1 CONSIDERAÇÕES INICIAIS

O objetivo desta tese foi desenvolver um sistema de integração similar ao desenvolvido no contexto do projeto SMErobot; porém, aplicável ao cenário nacional de PMEs. A realidade de PMEs nacionais é bastante diversa daquela observada em países de primeiro mundo que desenvolvem pesquisa nesta área. Geralmente, não é possível adquirir equipamentos de última geração de modo a proporcionar a infra-estrutura mínima para um sistema integrado de monitoramento e controle conforme proposto pelo modelo de Naumann [1, 8] ou ainda, reafirmado pela norma VDMA 66430-1 [11].

Este panorama foi discutido no capítulo 4, no qual fica claro que para o ambiente de PMEs, existe um campo vasto para o desenvolvimento de aplicações. Isto se deve a uma mentalidade mais flexível quanto ao emprego de novas tecnologias. Por outro lado, não existe um investimento de capital próprio adequado para a pesquisa e desenvolvimento de novos sistemas. Desta

forma, a criatividade do pesquisador é necessária de modo a viabilizar um sistema de baixo custo e funcionalmente similar a hipótese proposta, no caso, o desenvolvimento de um sistema integrador para o monitoramento/atuação de dispositivos em células robótica, com o intuito de produção em pequena escala. Além disso, este sistema deve estar apto a ser reconfigurado de maneira rápida e operado de forma mais simples do que os sistemas atuais, permitindo dessa forma, novos ajustes da planta, os quais permitem uma maior variação quanto ao produto final produzido.

Considerando-se este cenário, foram estabelecidos os primeiros passos para a criação de um sistema autossuficiente, que exija mínima manutenção ou grau de especialização do usuário final; no caso, um operador de uma PME. Este sistema de integração é composto por um subsistema de gerenciamento de *tags*, um subsistema que atua como *proxy* de dispositivos e um subsistema para o controle de célula robótica (discutidos no capítulo 4).

Os resultados obtidos por este doutorado são apresentados na seção 5.2 e o contexto de trabalhos futuros é apresentado na seção 5.3.

5.2 RESULTADOS OBTIDOS

A proposta inicial seguiu o modelo proposto por Naumann (discutido na seção 2.3.1), quanto a funcionalidade. Conforme o conhecimento do modelo e o trabalho de pesquisa amadureceram, esta idéia inicial precisou ser modificada. O ponto crucial diz respeito às funcionalidades previstas neste modelo, quanto as camadas P'n'P de aplicação, configuração e comunicação (conforme visto na seção 2.3.1).

Serviços correspondentes às funcionalidades da camada P'n'P de aplicação são desempenhados pelo subsistema de *tag* (indexação e cadastro de arquivos de configuração XIRP) e pelo subsistema de *proxy* de dispositivo (interface e geração automática de aplicação). A função conjunta de ambos os

módulos é prover a indexação de dispositivos não-XIRP e o conhecimento dos mesmos à célula robótica via protocolo XIRP. O ponto crucial para esta descoberta/configuração do dispositivo é o protocolo XIRP, o qual especifica um protocolo de comunicação que permite a 'descoberta' do dispositivo quando este é conectado na célula robótica.

As características da camada P'n'P de configuração são dependentes do meio de comunicação sendo utilizado e de arquivos de configuração pré-definidos, fornecidos com o dispositivo, no caso de um dispositivo XIRP nativo, ou especificados pelo especialista em nível de *proxy* de dispositivo ou de controlador de célula robótica, no caso de um dispositivo não-XIRP. No caso de um dispositivo não-XIRP, é o controlador de célula robótica que realiza o acesso ao barramento em uso (e.g. Ethernet, USB, CAN etc). Dessa forma, a configuração e acesso ao meio é uma característica deste subsistema.

Considerando a camada P'n'P de comunicação, existem funções desta presentes no *proxy* de dispositivo (i.e. lidar com protocolo de comunicação – no caso o protocolo XIRP e a descoberta de dispositivo) e no controlador de célula robótica (i.e. endereçamento de dispositivo, ajuste dos meios de comunicação básicos). Vale observar, que esta adaptação foi necessária, considerando o tratamento dado a um dispositivo não-XIRP. A descoberta, nesse caso é iniciada a partir da inserção do dispositivo no subsistema de *tag* ou ainda, a partir do rastreamento e identificação da *tag* para um dispositivo já cadastrado.

5.3 TRABALHOS FUTUROS

O sistema de *tag* desempenha um papel fundamental dada a possibilidade de implementação de indexação para dispositivos não-microprocessados. Além disso, é possível obter informações sobre um dispositivo de maneira rápida e simples. Um trabalho futuro diz respeito a forma como os

arquivos de configuração XIRP XML são armazenados no sistema. Hoje, estes são armazenados como arquivos hospedados em um servidor web.

Em [12] observa-se o uso do software de código aberto `eXist-db`[1] para a implementação de banco de dados com a capacidade de indexação e acesso rápido a conteúdo XML. Isto permitiria ao sistema de *tag* o armazenamento direto dos arquivos de configuração XIRP XML dentro do banco de dados, simplificando o acesso/armazenamento da informação. A análise do impacto desta implementação seria um trabalho interessante.

Outra questão importante diz respeito a geração de código fonte. Analisando-se o trabalho desenvolvido em [12] observa-se que a questão é bem mais profunda do que apenas a interpretação dos arquivos de configuração XIRP (embora esta seja uma tarefa imprescindível para o processo). Isto ocorre porque o XIRP não possui informações relativas a funcionalidade de dispositivo; apenas os comandos implementados e parâmetros de entrada. Não existem informações quanto ao fluxo de ações realizado pelo dispositivo. Desta forma, Verl e Naumann [12] propõe a implementação de um módulo para geração de aplicações, o qual recebe como entrada:

- Descrições de dispositivo, fornecidas por um especialista, as quais contém informação sobre a funcionalidade e programação de um dispositivo;
- Descrições de processo, fornecidas por um especialista, as quais contém informação sobre a funcionalidade necessária e fluxo de processo;
- Descrição da aplicação, criada pelo operador, a qual inclui os processos usados e seus parâmetros.

Percebe-se que a questão refere-se a geração de uma aplicação completa, configurável pelo operador, desde que um especialista disponibilize os alicerces necessários para a geração automática desta. Estes alicerces são

[1]Disponível em: `http://exist.sourceforge.net/` Acesso em: 14/09/2010.

uma descrição funcional do dispositivo (residente no próprio dispositivo ou disponibilizada ao subsistema de *tag*) e uma descrição do processo como um todo (hospedada na célula robótica). O operador também é parte integrante do procedimento e fornece *feedback* importante para a implementação desta 'aplicação dinâmica', tendo em vista que ele insere as funcionalidades necessárias para a configuração dinâmica da aplicação.

Atualmente, a integração entre o subsistema de *tag* e o *proxy* de dispositivo está funcional. O *proxy* de dispositivo é o responsável por adquirir os arquivos de configuração, a partir do subsistema de *tag* e desempenhar o passo fundamental para a geração automática de código. Este passo é a geração dos métodos para a serialização/de-serialização dos esquemas XML com o objetivo de geração de mensagens XIRP a serem trocadas com outros dispositivos ou com a célula robótica. Além disso, o servidor de canais, aplicação a ser disparada pelo servidor de dispositivos *servlet* para cada dispositivo também já está funcional. No momento, todavia, a integração entre um *servlet* de teste e o servidor de canais para o envio de mensagens ainda está em andamento.

Neste ponto, para dar continuidade ao trabalho, seria fundamental o estudo de State Chart XML (SCXML) e Web Ontology Language (OWL). O SCXML seria usado para implementar a descrição genérica do processo, enquanto o OWL seria usado para implementar a descrição do dispositivo [12].

Somente a partir da geração das descrições de dispositivo e do processo será possível iniciar os estudos relativos a implementação de sinergia entre dispositivos na célula robótica. Para tanto, alguma tecnologia de 'racionalização' (*reasoning*) será necessária. Estima-se que será usado o OWL-DL (*description logic*) para a implementação de racionalização[2]. Com isso, a descrição lógica de dispositivo e da célula robótica pode ser alcançada.

Outro ponto a ser observado diz respeito a participação do ope-

[2]Informações introdutórias em tecnologias para racionalização podem ser obtidas em `http://pellet.owldl.com/papers/sirin07pellet.pdf`, `http://clarkparsia.com/pellet/`. Acesso em: 15/09/2010.

rador na especificação da aplicação a ser gerada. De acordo com a proposta de Verl e Naumann [12], somente com a entrada fornecida pelo operador, a geração da aplicação será concluída, de modo a compor dinamicamente as funções ou estados de interesse para cada operador/aplicação.

A integração entre o *proxy* de dispositivo e o controlador de célula robótica ainda precisa ser implementada. Todavia, as bases para a integração de dispositivos já existem. Considerando o modelo arquitetural desenvolvido, a comunicação é processada em camadas e de maneira hierárquica (seção 4.2). Dessa forma, pressupõe-se que a etapa de 'descoberta' de dispositivo seja realizada a partir do subsistema de *tag* (cadastro do dispositivo e de arquivos de configuração), passando pelo *proxy* de dispositivo (geração de aplicação de controle) e comunicação com o controlador de célula robótica, onde o LKM adequado será descoberto e carregado (caso de uso 'carregar configuração' discutido na seção 4.5).

A criação de LKMs hoje é totalmente manual, no contexto do controlador de célula robótica. É necessário desenvolver algum mecanismo, mais simples, para a criação/gerenciamento dos LKMs para integração de dispositivos não-microprocessados. Um projeto de código aberto interessante neste contexto é o Linux Control and Measurement Device Interface[3] (ou *comedi*). Sua proposta é criar uma infra-estrutura para a criação/uso padronizado de *drivers*. Dessa forma, as particularidades do *driver* ficam limitadas ao desenvolvedor do *driver* (i.e. aos gerentes do projeto *comedi*). Um especialista de nível mais alto, simplesmente usará chamadas de sistema simplificadas para fazer a conexão/desconexão de um determinado dispositivo que já possui um *driver* implementado pelo projeto.

O maior problema detectado com este projeto é sua ausência de documentação recente, embora ainda exista o desenvolvimento recente para *drivers* de placas de aquisição. Dessa forma, seria necessário aprender a

[3]Disponível em: `http://www.comedi.org/`. Acesso em: 15/09/2010.

utilizar o modelo de programação do *comedi*, para a criação de novos *drivers* caso estes ainda não existam. A contrapartida é que quaisquer equipamentos com *drivers* existentes não demandariam esforço de implementação.

Um último resultado obtido, foi a atualização e melhoria do sistema Real Time Linux Live CD desenvolvido em [64]. Este foi o resultado do trabalho desenvolvido por Godoy [65] e orientado no contexto deste doutorado. Neste contexto, foi possível inserir o conteúdo originalmente contido em um LiveCD, em um LiveUSB de modo que este possa ser modificado e atualizado. Assim, confere-se uma maior liberdade quanto a utilização/modificação/atualização deste sistema. Além disso, foram realizadas atualizações no sistema quanto a versão do *kernel* e do sistema de tempo real RTAI. Como continuidade para este trabalho, pretende-se inserir o suporte ao Orocos como forma de implementar um sistema de controle robótico portátil.

Como último desdobramento para este projeto, considerando que este alcance uma versão estável, existe a intenção de aplicá-lo em PMEs com o intuito de validação. Este é um processo demorado, considerando questões relativas a propriedade intelectual, estabilidade do sistema e a existência de um parceiro disposto a permitir a coleta de dados em suas dependências (eventualmente, atuação de dispositivos em menor escala).

5.4 CONSIDERAÇÕES FINAIS

Este doutorado foi resultado do estudo de uma proposta existente para o desenvolvimento de tecnologias voltadas a PMEs. Neste contexto, o conceito de Plug' n' Produce foi estudado e um modelo foi escolhido como base para o início da pesquisa. Vale observar que o caráter de ineditismo não é proveniente unicamente do desenvolvimento tecnológico alcançado mediante o uso de um modelo existente. O destaque e o ineditismo são fruto da adequação desse modelo às necessidades de PMEs. Isto inclui estabele-

cer premissas que tornem o modelo aplicável a ambientes tecnologicamente restritivos, considerando o contexto de PMEs brasileiras, as quais não teriam condições de adquirir uma infra-estrutura unicamente baseada em dispositivos microprocessados, o que poderia implicar na total substituição de uma linha de produção.

A implementação proposta tem como ganho principal a possibilidade de permitir uma adequação a uma infra-estrutura similar àquela desenvolvida para o projeto SMErobot, porém, com a possibilidade de migração gradual. Isso confere ao padrão uma abrangência maior do que àquela para a qual este foi previsto originalmente. Além disso, prevê-se que a implantação de inovação tecnológica aliada a aplicação de um padrão internacional de comunicação pode causar impacto e modernização quanto a geração de novos produtos e processos no contexto de PME's.

Desta forma, o alcance da solução proposta não seria apenas nacional, mas possui âmbito internacional considerando uma adaptação a um cenário previamente proposto no contexto de um projeto de maior escala (*sixth framework programme* – FP6 – da União Européia). Isto pode alavancar parcerias e intercâmbio tecnológico, mediante o uso dos contatos realizados no decorrer deste trabalho de pesquisa.

Conforme discutido, a arquitetura de sistema proposta e a implementação parcial alcançada ainda não constituem o sistema funcional, porém, consegue demonstrar o potencial e o valor agregado desta pesquisa. Além disso, considerando que o objetivo deste doutorado é "criar um modelo arquitetural para a integração de dispositivos, que permita a adequação de uma PME, de maneira gradual, sem onerar custos proibitivos relativos a aquisição de equipamentos específicos para a operação em uma planta compatível com P'n'P", pode-se dizer que este objetivo foi cumprido, na forma da especificação inicial da arquitetura ACID.

Levando em consideração os resultados alcançados e os desdo-

bramentos pretendidos para este projeto (seção 5.3), percebe-se que o mesmo não representa uma linha de desenvolvimento estanque. Existem investimentos previstos no Brasil considerando a pesquisa, desenvolvimento e inovação, visando a modernização e ampliação da capacidade de PMEs [9]. Nesse contexto, o resultado desta pesquisa possui vantagem estratégica em aplicações industriais em curto, médio e longo prazo.

Referências Bibliográficas

1 NAUMANN, M.; WEGENER, K.; SCHRAFT, R. D. Control architecture for robot cells to enable plug'n'produce. In: *2007 IEEE International Conference on Robotics and Automation*. Rome, Italy: [s.n.], 2007. 15, 3, 6, 11, 12, 13, 14, 16, 17, 18, 19, 21, 30, 46, 114

2 ARAI, T. et al. Holonic assembly system with plug and produce. *Computers in Industry*, v. 46, n. 3, p. 289–299, October 2001. 15, 9, 10, 11, 12, 13, 19, 20, 30

3 GAUSS, M. et al. A standard communication interface for industrial robots and processor based peripherals - xirp. In: *VDI-Wissensforum et al.: ISR 2006 – ROBOTIK 2006 – Proceedings of the Joint Conference on Robotics*. [S.l.: s.n.], 2006. 15, 3, 26, 27, 28, 29, 30, 31, 35, 38, 39, 42, 43, 48, 49, 52, 53, 137

4 DAI, J. U. F. Towards a standardized and extensible mechanism for robot device integration – a xirp-based approach and test-bed implementation. In: FILIPE, J.; CETTO, J. A.; FERRIER, J. (Ed.). *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics*. Funchal, Madeira, Portugal: INSTICC Press, 2008. p. 255–262. 15, 44, 45, 46

5 NAZARIEV, A. *Servlet adapters can solve your socket problems*. Jan 2001. Disponível em: <http://adtmag.com/Articles/2001/12/01/Servlet-adapters-can-solve-your-socket-problems.aspx?Page=1>. 16, 75

6 AROCA, R.; TAVARES, D. M.; CAURIN, G. A. P. Free open source tools for real-time robotics & automation. In: *In Proceedings of The 6th Brazilian Conference on Dynamics, Control and Their Applications - DINCON'2007*. [S.l.: s.n.], 2007. 16, 94, 133

7 BOCZKAJ, B. F. Software aspects of plcs application in robotic workcells. In: IEEE. *Thirty-First IAS Annual Meeting, IAS '96., Conference Record of the 1996 IEEE Industry Applications Conference, 1996.* IEEE, 1996. v. 3, p. 1575–1581. Disponível em: <http://ieeexplore.ieee.org/iel3/4112/12178-/00559277.pdf?tp=&isnumber=&arnumber=559277>. 1

8 NAUMANN, M. et al. Robot cell integration by means of application-p'n'p. In: *VDI-Wissensforum et al.: ISR 2006 – ROBOTIK 2006 – Proceedings of the Joint Conference on Robotics*. [S.l.: s.n.], 2006. 1, 3, 6, 12, 13, 21, 22, 26, 30, 31, 114

9 Ministério da Ciência e Tecnologia. O plano de ação de ciência, tecnologia e inovação para o desenvolvimento nacional – PACTI 2007-2010. In: Ministério da Ciência e Tecnologia. *VI Seminário de Ciência e Tecnologia de Interesse da Defesa Nacional*. Ministério da Ciência e Tecnologia, 2008. Disponível em: <https://www.defesa.gov.br/ciencia_tecnologia/palestras-/2008/vi_seminario/terca/PACTI_2007_2010_Seminario_Defesa.pdf>. 2, 122

10 CONSOLI, M. A.; MARTINELLI, D. P.; JOYAL, A. Comparação da realidade de pmes brasileiras e canadenses – uma pesquisa em andamento. In: *III SIMPOI – Simpósio de Administração da Produção, Logística e Operações Internacionais*. São Paulo: [s.n.], 2000. 2

11 Verband Deutscher Maschinen- und Anlagenbau. *XML-basiertes Kommunikationsprotokoll für Industrieroboter und prozessorgestützte Peripheriegeräte (XIRP) Teil 1: Allgemeine Vereinbarungen*. 1. ed. [S.l.]: Verband Deutscher Maschinen- und Anlagenbau, 2006. 3, 5, 28, 30, 32, 33, 37, 52, 53, 54, 77, 112, 114

12 VERL, A.; NAUMANN, M. Plug'n'produce für roboterzellen: Generierung ausführbarer programme aus abstrakten beschreibungen. *wt Werkstattstechnik online*, v. 98, n. 5, p. 384–390, 2008. 6, 21, 46, 112, 117, 118, 119

13 MURAKAMI, L. T. *SOLUÇÃO DE PROBLEMAS DE OTIMIZAÇÃO UTILIZANDO ARQUITETURA HÍBRIDA*. Tese (Doutorado) — Escola Politécnicada Universidade de São Paulo, 2008. 16

14 ZIMMERMANN, U. E.; BISCHOFF, R. Communication, configuration, application: The three layer concept for plug-and-produce. In: FILIPE,

J.; CETTO, J. A.; FERRIER, J. (Ed.). *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics*. Funchal, Madeira, Portugal: INSTICC Press, 2008. p. 255–262. 22, 23

15 BIERHOFF, K.; GRECHANIK, M.; LIONGOSARI, E. S. Architectural mismatch in service-oriented architectures. In: *SDSOA 07: Proceedings of the International Workshop on Systems Development in SOA Environments*. Washington, DC, USA: IEEE Computer Society, 2007. p. 4. ISBN 0-7695-2960-7. 23, 24

16 VEIGA, G.; PIRES, J.; NILSSON, K. Experiments with service-oriented architectures for industrial robotic cells programming. *Robotics and Computer-Integrated Manufacturing*, v. 25, n. 4-5, p. 746–755, August-October 2008. 23

17 NILSSON, K.; BENGEL, M. Plug-and-produce technologies: real-time aspects; service oriented architectures for sme robots and plug-and-produce. In: FILIPE, J.; CETTO, J. A.; FERRIER, J. (Ed.). *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics*. Funchal, Madeira, Portugal: INSTICC Press, 2008. p. 249–254. ISBN 978-989-8111-31-9. 24

18 KREGER, H. *Making Sense of SOA Standards*. 2009. Disponível em: <http://www.cioupdate.com/reports/article.php/3841701/Making-Sense-of-SOA-Standards.htm>. 24

19 ARIKT: Adaptive Robot Based Visual Inspection. 2003. Acesso em: 22/01/2008. Disponível em: <http://wwwipr.ira.uka.de/?fileid=302>. 27

20 GAUSS, M. et al. Adaptive robot based visual inspection of complex parts. In: *International Symposium on Robotics (ISR) 2003 (Technical Conference Proceedings)*. [s.n.], 2003. Acesso em: 22/01/2008. Disponível em: <http://wwwipr.ira.uka.de/?fileid=302>. 27

21 AN Architecture and Communication Protocol for Interaction of Industrial Robots and Vision Systems. In: ICAR 2003, The 11th International Conference on Advanced Robotics. University of Coimbra, Portugal: [s.n.], 2003. Acesso em: 22/01/2008. Disponível em: <http://wwwipr.ira.uka.de/?fileid=305>. 27

22 VDI-BERICHTE/MESS UND AUTOMATISIERUNGSTECHNIK. Adaptive roboterbasierte visuelle inspektion komplexer teile. In: *Applied Machine Vision*. Stuttgart, 2003. 27

23 BROGÅRDH, T. Present and future robot control development—an industrial perspective. *Annual Reviews in Control*, v. 31, n. 1, p. 69–79, 2007. 30

24 MAEDA, Y. et al. "plug & produce" functions for an easily reconfigurable robotic assembly cell. *Assembly Automation*, v. 27, n. 3, p. 253–260, 2007. 30

25 HAAGE, A. N. M.; NUGUES, P. Toward ontologies and services for assisting industrial robot setup and instruction. In: FILIPE, J.; CETTO, J. A.; FERRIER, J. (Ed.). *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics*. Funchal, Madeira, Portugal: INSTICC Press, 2008. p. 263–270. 30

26 INDUSTRIELLE Automation – Sensorschnittstelle für Roboter und Fertigungssysteme – Datenübertragung. [S.l.], 1987. 31

27 THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Open systems interconnection (OSI)*. [S.l.], 1995a. 31

28 SPERBERG-MCQUEEN, C. M.; THOMPSON, H. *XML Schema*. World Wide Web Consortium, 2010. Disponível em: <http://www.w3.org/XML/Schema>. 32

29 W3SCHOOLS. *Introduction to XML Schema*. 2010. Disponível em: <http://www.w3schools.com/schema/schema_intro.asp>. 32

30 ISO/IEC. *Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. 2. ed. [S.l.]: ISO/IEC, 1994. 33

31 DAI, F. et al. *Erfahrungen und Weiterentwicklungen Standardisierter Roboter-Geräte-Kommunikation*. 2008. Presentation. 44, 46

32 THE Dual Challenge of Converging Communications for Industrial Automation Networks. [S.l.], 2009. Disponível em: <http://www.patton.com-/whitepapers/converging_industrial_comms_white_paper.pdf>. 48

33 BAKKEN, D. E. et al. *Towards More Flexible and Robust Data Delivery for Monitoring and Control of the Electric Power Grid*. Pullman, WA USA, May 2007. Disponível em: <http://www.arcadiannetworks.com/>. 48

34 BUNSENDAHL, J. The integration of vision systems into robot controllers. In: *International Symposium on Robotics (ISR) 2003 (Technical Conference Proceedings)*. [S.l.: s.n.], 2003. 49

35 MARTINELLI, D. P.; JOYAL, A. *Desenvolvimento Local e o Papel das Pequenas e Médias empresas*. [S.l.]: São Paulo: Manole, 2004. 50

36 HUMMEL, O.; ATKINSON, C. Extreme harvesting: Test driven discovery and reuse of software components. In: *IEEE International Conference on Information Reuse and Integration*. [S.l.]: IEEE Systems, Man, and Cybernetics Society (SMC), 2004. p. 66–72. 51

37 BI, L. et al. Design and implementation of the airline luggage inspection system base on link structure of qr code. In: IEEE (Ed.). *International Symposium on Electronic Commerce and Security*. [S.l.], 2008. p. 527–530. 54

38 HULL, J. J. et al. Paper-based augmented reality. In: IEEE (Ed.). *17th International Conference on Artificial Reality and Telexistence*. [S.l.], 2007. p. 205–209. 55

39 QR-CODE Readers. 2010. Disponível em: <http://www.mobile-barcodes-.com/qr-code-software/>. 64

40 TAVARES, D. M.; CAURIN, G. A. de P. Proposal for the use of QR Code in supply chain management. Pending approval at Production Online Magazine (Revista Produção Online). July 2010. 67, 112

41 TAVARES, D. M.; CAURIN, G. A. P. Proposal for a device proxy using xirp. In: IEEE/IAS. *In Proceedings of the 9th IEEE/IAS International Conference on Industry Applications - INDUSCON 2010*. [S.l.], 2010. 67, 112

42 TAVARES, D. M.; AROCA, R.; CAURIN, G. A. P. Upgrade of a scara robot using orocos. In: *In Proceedings of the 13th IASTED International Conference on Robotics and Applications - RA 2007*. [S.l.: s.n.], 2007. 67, 112

43 Java Community Process. *JSRs: Java Specification Requests JSR 31: XML Data Binding Specification*. March 2003. Disponível em: <http://jcp.org-/en/jsr/detail?id=31>. 76

44 XML Data Binding Tools. 2010. Disponível em: <http://www-.xmldatabinding.org/>. 78

45 XML Schema Working Group. *XML Schema Part 1: Structures Second Edition*. [S.l.], October 2004. Disponível em: <http://www.w3.org/TR-/xmlschema-1/>. 80

46 AROCA, R.; TAVARES, D. M.; CAURIN, G. A. P. Scara robot controller using real time linux. In: *In Proceedings of the 2007 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. [S.l.: s.n.], 2007. 85, 96

47 NILSSON, K. *Industrial Robot Programming*. Tese (Doutorado) — Department of Automatic Control – Lund University, 1996. 85, 89

48 ASATO, O.; KATO, E.; AL., R. I. et. Analysis of open cnc architecture for machine tools. *Journal of the Brazilian Society of Mechanical Sciences*, 2002. Acesso em: 22/01/2008. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-73862002000300009&lng=en&nrm=iso>. 87

49 BRUYNINCKX, H.; SOETENS, P. The OROCOS project. Acesso em: 22/01/2008. 2007. Disponível em: <http://people.mech.kuleuven.be-/~psoetens/orocos/devel/rtt/latest/doc-xml/orocos-overview.html>. Acesso em: 18/04/2006. 87, 91

50 COLOMBO, D.; DALLEFRATE, D.; TOSATTI, L. M. Pc based control systems for compliance control and intuitive programming of industrial robots. In: *VDI-Wissensforum et al.: ISR 2006 – ROBOTIK 2006 – Proceedings of the Joint Conference on Robotics*. [S.l.: s.n.], 2006. 87, 88

51 SOUSA, M. de. Matplc-the truly open automation controller. In: *IECON 02 – IEEE 2002 28th Annual Conference of the Industrial Electronics Society*. [S.l.: s.n.], 2002. p. 2278–2283. ISBN: 0-7803-7474-6. 88

52 KUKA ROBOTER GMBH. *KR C1, KR C2 – Robot Sensor Interface (RSI)*. [S.l.], 2006. Release 1.0 (BETA). 89

53 OLSSON, M. *Simulation and execution of autonomous robot systems*. Tese (Doutorado) — Department of Mechanical Engineering – Lund University, 2002. 90

54 PONT, F.; SIEGWART, R. A real-time software framework for indoor navigation. In: *In Proceedings of the IEEE/RSJ Intenational Conference on Intelligent Robots and Systems - IROS'2005*. [S.l.: s.n.], 2005. 90

55 BRUYNICKX, H. Open robot control software: the orocos project. In: *In Proceedings of IEEE International Conference on Robotics and Automation (ICRA) 2001*. [S.l.: s.n.], 2001. v. 3, p. 2523–2528. 91

56 AARNO, D. *Control of a PUMA 560 Using Linux Real-Time Application Interface (RTAI)*. 2004. Http://www.nada.kth.se/ bishop/rtcontrol.pdf. Acesso em: 22/01/2008. 92

57 AARNO, D. *Evaluation of Real-Time Linux Derivatives for Use in Robotic Control*. 2004. Acesso em: 22/01/2008. Disponível em: <http://www.nada-.kth.se/˜bishop/rtos.p>. 92

58 MIGNOT, G. L. *Monolithic kernels and micro-kernels*. 2005. On-line. Acesso em: 22/01/2008. Disponível em: <http://kilobug.free.fr/hurd/presen/abstract/html/node2.html>. 95

59 AROCA, R. V. *Análise de Sistemas Operacionais de Tempo Real para Aplicações de Robótica e Automação*. Dissertação (Mestrado) — Escola de Engenharia de São Carlos da Universidade de São Paulo, 2008. 97

60 BRUYNICKX, H. *The Orocos Component Builder's Manual*. 2007. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.orocos.org/stable-/documentation/rtt/v1.2.x/doc-xml/orocos-components-manual.html>. 103

61 KUPPENS, H. *Realtime Linux (RTAI) - Radboud University Experiment #3 LinuX RealTime - LXRT*. 2006. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.cs.ru.nl/lab/rtai/experiments/3.LXRT/Experiment-3-.html>. 109

62 WILLIAMS, C. *Linux Scheduler Latency*. March 2002. On-line. Disponível em: <http://www.linuxfordevices.com/files/article027/rh-rtpaper.pdf>. 109

63 MANTEGAZZA, P. *DIAPM RTAI for Linux: WHYs, WHATs and HOWs*. 1999. Https://www.rtai.org/index.php?module=documents&JAS DocumentManager op=downloadFile&JAS File id=31. Acesso em: 22/01/2008. 110

64 AROCA, R.; TAVARES, D. M.; CAURIN, G. A. P. Mechatronics real time linux - a rtos live cd based on linux. In: *In Proceedings of the Workshop de Sistemas Operacionais - WSO 2007*. [S.l.: s.n.], 2007. 120

65 GODOY, A. P. de. *Desenvolvimento de uma distribuição LiveUSB para ensino de sistemas de tempo real. Iniciação Científica. Instituto Federal de Educação, Ciência e Tecnologia.* 2010. 120

66 SCHMIDT, D. C. *Overview of ACE*. September 2006. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.cs.wustl.edu/~schmidt/ACE-overview.html>. 132

67 REAL-TIME Monitoring in Mission-based Wireless ad-hoc Sensor Actor Networks. November 2006. On-line. Presented on Sensor networks - Quo Vadis? Paving the Road to Real Business Applications 23-Nov-2006, IST Event, Helsinki, Finland. Disponível em: <http://www.cobis-online.de-/ist2006/>. 132

68 BHATTACHARYYA, D. *Introduction to VisiBroker CORBA Development with JBuilder*. 2008. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.devb.com/borcon/3168.html>. 132

69 ATIS Committee T1A1. *ATIS Telecom Glossary*. 2007. On-line. Acesso em: 12/09/2010. Disponível em: <http://www.atis.org/tg2k/>. 132, 133, 134, 136

70 JOHNSON, R. E. Documenting frameworks using patterns. In: *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*. [s.n.], 1992. p. 63–72. Disponível em: <citeseer.ist.psu.edu/johnson92documenting.html>. 133

71 jGuru.com. *Tutorials & Code Camps – Introduction to CORBA*. [S.l.], 1999. Acesso em: 22/01/2008. Disponível em: <http://java.sun.com/developer-/onlineTraining/corba/corba.html>. 133

72 OBJECT MANAGEMENT GROUP. *CORBA BASICS*. 140 Kendrick Street, Building A Suite 300, Needham, MA 02494, U.S.A., September 2007. Acesso em: 22/01/2008. Disponível em: <http://www.omg.org/gettingstarted-/corbafaq.htm>. 133

73 JAPANESE ECONOMY DIVISION. *Trends in the Japanese Robotics Industry*. March 2006. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.jetro.go.jp/en/market/report/pdf/2006_15_c.pdf>. 134

74 GUNTHER, N. *UNIX Load Average Part 1: How It Works*. February 2003. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.teamquest-.com/resources/gunther/display/5/index.htm>. 134

75 RESTREPO, T. *C++ Namespaces*. April 2002. On-line. Acesso em: 22/01/2008. Disponível em: <http://www-h.eng.cam.ac.uk/help/tpl-/languages/C++/namespaces.html>. 134

76 The Software Engineering Institute. *Object Request Broker – Software Technology Roadmap*. [S.l.], 2007. Acesso em: 22/01/2008. Disponível em: <http://www.sei.cmu.edu/str/descriptions% -/orb.html>. 134

77 TERRIS, A. Military avionics retrofit programs-engineering and management considerations. In: *Proceedings of the IEEE 1990 National Conference, NAECON 1990*. [S.l.: s.n.], 1990. p. 1012–1017. 135

78 YODAIKEN, V.; BARABANOV, M. *Real-Time Linux*. 2007. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.soe.ucsc.edu/~sbrandt-/courses/Winter00/290S/rtlinux.pdf>. 135

79 Wind River. *Wind River Acquires Hard Real-Time Linux Technology from FSMLabs*. February 2007. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.windriver.com/announces/rtlinux/>. 136

80 SCHMIDT, D. C. *Overview of the ACE+TAO Project*. October 2006. On-line. Acesso em: 22/01/2008. Disponível em: <http://www.cs.wustl.edu-/~schmidt/TAO-overview.html>. 137

81 ROTHER, M.; SHOOK, J. *Aprendendo a Enxergar*. 1.3. ed. [S.l.]: Lean Institute Brasil, 2003. 137

82 NAZARENO, R. R.; RENTES, A. F.; SILVA, A. L. Implantado técnicas e conceitos da produção enxuta integradas à dimensão de análise de custos. In: *Encontro Nacional de Engenharia de Produção (ENEGEP 2001)*. [s.n.], 2001. Acesso em: 22/01/2008. Disponível em: <http://www.numa.org.br/gmo-/arquivos/artigo.doc>. 137

83 LAPLANTE, P. A. *REAL-TIME SYSTEMS DESIGN AND ANALYSIS*. [S.l.]: A JOHN WILEY & SONS, INC., PUBLICATION, 2004. 137

84 The Sherrill Group. *Multithreaded Programming :: Improving Performance through Threads (pthreads Tutorial)*. 2002. On-line. Acesso em: 22/01/2008. Disponível em: <http://randu.org/tutorials/>. 137

Glossário

ACE — O *ADAPTIVE Communication Environment* (ACE) é um *framework* de software disponível gratuitamente, *open-source*, orientado a objeto (OO) que implementa padrões fundamentais (*core*) para software de comunicação concorrente. O ACE oferece um rico conjunto de abstrações C++ (*wrapper facades*) e componentes que desempenham tarefas de comunicação de software comuns a uma variada gama de SOs [66].

Ator — Um ator representa um recurso mais nobre quando comparado a um sensor. É geralmente dotado de uma unidade de processamento melhor, maior poder de transmissão, tempo de vida de bateria mais longo e mobilidade [67].

Common Object Request Broker Architecture — O *Common Object Request Broker Architecture* (CORBA) é uma coleção de especificações desenvolvidas e padronizadas pelo *Object Management Group* (OMG). O *framework* CORBA padrão permite que fabricantes de software individuais desenvolvam produtos que se adequem às especificações da OMG. Vários componentes de uma implementação CORBA são definidos pelo padrão OMG; entretanto, a maioria dos fabricantes aumentam estes componentes básicos para oferecer uma solução completa. O padrão CORBA define como os objetos são compartilhados e interagem em um ambiente distribuído, assim como o protocolo de comunicação pelo qual eles fazem isso [68].

Daemon — Abreviatura para *disk and execution monitor*. Um procedimento que é invocado sem ser explicitamente chamado sempre que uma alteração, adição ou remoção ou outro evento ocorra [69].

eXtended Markup Language — uma especificação ou versão mais elaborada da linguagem *Standard Generalized Markup Language* (SGML), que permite a desenvolvedores web a criação de *tags* personalizadas para funcionalidade adicional [69].

Firewall — Um computador ou equipamento que (a) age como uma interface entre duas rede (e.g., a Internet e uma rede privada, respectivamente), e (b) regula o tráfego entre aquelas redes para o propósito de proteger a rede interna de ataques eletrônicos originados a partir da rede externa [69].

Framework — Um *framework* consiste no projeto reutilizável de um programa, ou parte de um programa, expresso como um conjunto de classes. Por definição, um *framework* é um projeto orientado a objeto (embora não precise ser implementado em uma linguagem orientada a objeto). O reaproveitamento em larga escala de bibliotecas orientadas a objeto requerem o uso de *frameworks*. Este fornece um contexto para os componentes na biblioteca serem reutilizados [70].

IDL CORBA — A *Interface Definition Language* da OMG suporta a especificação de objetos, mas não como eles são implementados. Assim, em IDL não existe maneira de declarar o estado de um objeto e algoritmos. a implementação de um objeto CORBA é realizada em uma linguagem de programação padrão (e.g. Java, C++, etc). Uma interface especifica o contrato entre o código usando o objeto e o código implementando o objeto. Os clientes depenem apenas de sua interface [71].

Industry Packs — Em um PC industrial, são representados por placas menores com funções específicas, mantidas pelas placas *carrier* [6].

Internet Inter-ORB Protocol — Internet Inter-ORB Protocol (IIOP) é um protocolo que torna possível a comunicação entre programas distribuídos escritos em linguagens de programação diferentes pela Internet. O IIOP é uma parte crítica de um padrão industrial estratégico, o Common Object Request Broker Architecture (CORBA). Por meio do protocolo IIOP, um programa baseado em CORBA de qualquer fabricante, em quase qualquer computador, sistema operacional, linguagem de programação e rede, pode interoperar com um programa baseado em CORBA do mesmo fabricante ou de outro, em quase qualquer computador, sistema operacional, linguagem de programação e rede [72].

Jitter — *Jitter* consiste na variação abrupta e indesejada de uma ou mais características de um sinal, como o intervalo entre pulsos sucessivos, a amplitude de ciclos sucessivos, ou a freqüência ou fase de ciclos sucessivos [69].

Keiretsu — Exemplos de *keiretsus* são encontrados em linhas de produção especificamente projetadas para um determinado fabricante e envolvem muitos outros sistemas além de robôs. A adaptação de uma linha de produção existente à tecnologia de um fabricante distinto é um procedimento trabalhoso e caro. Considera-se essa uma característica desejável quando a aplicação é bem definida e não são esperadas mudanças. Por exemplo, a FANUC possui

presença intensa na Nissan, enquanto a Kawasaki Heavy Industries e Nachi comercializam a maioria de seus produtos para a Toyota. Uma vez que uma companhia assegurou um fabricante automobilístico como cliente, torna-se mais fácil comercializar com o "*keiretsu*" (família de companhias) daquele fabricante [73].

Kernel — Um módulo de um programa que forma uma entidade lógica ou representa uma entidade funcional. Nota: a porção de código mais vulnerável em um sistema operacional seguro é um caso especial de um *kernel* [69].

Load average — O *load average* é uma métrica embarcada que aparece na saída de outros comandos UNIX como o $uptime$ e o $procinfo$. É uma métrica dependente de tempo comumente usada para observar o consumo de recursos de sistema. O *load average* tenta medir o número de processos ativos em um determinado momento. *Load averages* altos geralmente significam que o sistema está sendo usado pesadamente e o tempo de resposta é proporcionalmente lento. Um *load average bloqueante* desejável deveria estar abaixo de três (...) [74].

Namespace — A idéia de *namespaces* é comum em linguagens computacionais. Um *namespace* é como um contexto o qual determina o significado de um símbolo (uma boa analogia seria um espaço onde nomes são armazenados). Algumas linguagens simples possuem apenas um *namespace* - todos os nomes de função, nomes de variáveis, etc pertencem ao mesmo contexto. Outras linguagens possuem vários *namespaces* (um para nomes de variáveis, outro para nomes de função, etc) tornando possível ter uma variável e uma função com o mesmo nome, mas o número de papel destes *namespaces* são fixos [75].

Object Request Broker — Um *Object Request Broker* (ORB) é uma tecnologia de *middleware* que gerencia a comunicação e troca de dados entre objetos. ORBs promovem interoperabilidade entre sistemas de objetos distribuídos pois permitem que usuários construam sistemas pela junção de objetos (provenientes de diferentes fabricantes) que se comunicam entre si via ORB. Os detalhes de implementação do ORB geralmente não são importantes para desenvolvedores construindo sistemas distribuídos. Esta forma de encobrimento de informação melhora a manutenção do sistema, considerando que os detalhes de comunicação do objeto são escondidos dos desenvolvedores e isolados no ORB [76].

Off-the-shelf — Equipamento fabricado e disponível para consumo. Adaptado a partir de [69].

Ontologia — Em ciência da computação, uma ontologia é uma organização

rigorosa e exaustiva de algum domínio de conhecimento que é geralmente hierárquico e contém todas as entidades relevantes e suas relações[4].

Open Source Software* e *Free Software — A principal diferença entre esses conceitos é basicamente semântica. A escolha de um deles (i.e. "*open source*" ou "*free software*" deixa-se claro quais são os objetivos previstos pelo desenvolvedor. O movimento *open source* suporta os direitos do desenvolvedor de software considerando a cobrança por qualquer tipo de serviço relacionado ao software e a distribuição de código fonte e binário ao público, prática a qual o autor é compelido a realizar sempre. Já o movimento baseado em *free software* suporta os direitos individuais do usuário final, considerando que organizações possam cobrar por quaisquer serviços relacionados a software e, quanto a distribuição de código fonte e binário, o autor pode escolher não realizá-la se assim for seu desejo[5].

Retrofit — O conceito de *retrofit* aplicado a este projeto foi adaptado a partir do conceito apresentado por Terris [77], e aplicado para o contexto de robótica. Este representa uma tentativa de estender o tempo de vida útil de um sistema complexo e caro por meio de modificações no mesmo. Estas modificações podem envolver o reforço da estrutura e melhoria em muitos dos seus sistemas visando ganhos em precisão e confiabilidade. Embora a adição de novas funcionalidades seja importante, esse é tido como um objetivo secundário considerando a dificuldade de substituição de módulos fechados (*black-boxes*) em uma relação um-para-um. Pode-se dizer que a principal motivação para o início de um procedimento de *retrofit* seja uma tentativa de aumento em desempenho funcional equiparável a sistemas novos.

RTLinux — RTLinux (ou Real Time Linux) é uma extensão do Linux para sistemas operacionais de tempo real, originalmente desenvolvido por Yodaiken e Barabanov [78] no Instituto de Tecnologia e Mineração do Novo México (*New Mexico Institute of Mining and Technology*). O RTLinux foi disponibilizado em duas versões: uma suportada pela comunidade e livre e outra comercial denominada RTLinux Pro. O RTLinux suporta operação de tempo real *hard* (determinística) por meio de controle de interrupção entre hardware e o sistema operacional. Interrupções necessárias para o processamento determinístico são processadas pelo núcleo de tempo real, enquanto outras interrupções são enviadas para o sistema operacional de tempo não real. O sistema operacional

[4]Definição obtida em Wordnet - *A lexical database for English*. `http://wordnetweb.princeton.edu` Acesso em: 16/12/2010.

[5]Para maiores detalhes sobre esses conceitos, acesse `http://www.opensource.org/docs/definition.php` e `http://www.gnu.org/philosophy/free-sw.html` (Acesso em: 05/05/2009).

(Linux) executa como uma *thread* de baixa prioridade.Em 20 de Fevereiro de 2007 a divisão de software embarcado da FSMLabs[6], responsável pelo RTLinux, foi comprada pela *Wind River Systems* [79]. Desde então, o futuro do RTLinux tornou-se incerto. A versão GPL do site ainda existe mas novas atualização não foram feitas.

Segurança — O conceito de segurança possui conotação dupla quando consideramos o vocábulo em inglês referente a um ambiente industrial. Segundo a *European environment information and observation network* (Eionet), segurança (*safety*) refere-se a medidas ou técnicas implementadas para reduzir o risco de ferimentos, perda e perigo a pessoas, propriedade ou o ambiente em qualquer instalação ou lugar envolvendo a manufatura, produção e processamento de bens ou produtos. Segurança (*security*) também pode significar medidas, técnicas ou projetos implementados para proteger de danos ou restringir o acesso a qualquer aparato, maquinário ou construção, posicionado ou conectado para uso[7].

Shell script — Um programa de computador relativamente curto que desempenha uma tarefa específica é um *script*. É normalmente usado para automatizar funções complexas ou avançadas dentro de um sistema. Um *shell script* é um *script* que é interpretado por um interpretador de comandos, i.e., um utilitário de software que lê uma entrada especificando uma operação e então, pode realizar uma operação diretamente ou simplesmente controlá-la por meio de outros aplicativos em um sistema operacional (SO) (adaptado a partir de [69]).

Socket — Em redes de computadores, um *socket* Internet é uma interface de comunicação bidirecional entre processos em uma rede baseada em Internet Protocol (IP), como a Internet. Esta interface pode ser construída com base em uma conexão virtual ou em uma estrutura sem conexão dependendo do protocolo de transporte utilizado. *Sockets* Internet constituem um mecanismo para entrega de dados de chegada para um processo ou *thread*, baseada em uma combinação de endereço IP e número de porta.

***State Chart XML* (SCXML)** — *State Chart XML (SCXML) ou State Machine Notation for Control Abstraction*, oferece um ambiente de execução baseado em máquina de estados genérica, baseada em Call Control XML (CCXML) e tabelas de estado de Harel. Dessa forma, seria possível alcançar uma representação gráfica de máquinas de estado, associada a uma representação em XML es-

[6]Disponível em: `http://www.fsmlabs.com/` (Acesso em: 25/07/2007).
[7]Fonte: `http://www.eionet.europa.eu/gemet/` (acesso em 24/09/2010)

crito. [8].

TAO — *The ACE ORB* é um *middleware* CORBA que permite que clientes invoquem operações em objetos distribuídos sem levar em conta a localização do objeto, linguagem de programação, plataforma de SO, protocolos de comunicação e hardware. O TAO foi projetado para permitir alto desempenho e qualidade de serviço (e.g. controle de latência, vazão e *jitter* fim a fim) de tempo real para aplicações distribuídas [80].

Taxa IPO — Taxa de interpolação, taxa IPO ou *interpolation rate* representam valores típicos que variam entre 5-15 ms para robôs industriais [3].

Tempo de ciclo — Segundo Rother e Shook [81] e Nazareno et al. [82], o tempo de ciclo é o tempo necessário à conclusão de todo um processo. A sua análise é normalmente executada através do acompanhamento do trabalho das atividades exigidas no processo. As estimativas de tempo devem ser somadas separadamente, a fim de se obter os tempos de processamento e os tempos de não processamento.

Tempo real *firm* — Aquele no qual a perda de alguns poucos *deadlines* não levariam a uma total falha, mas perder uma quantidade elevada pode levar a uma falha de sistema total e catastrófica [83].

Tempo real *hard* — Sistemas nos quais uma única falha em atingir restrições quanto aos tempos de resposta levam a uma falha de sistema completa e catastrófica são chamados de sistemas de tempo real *hard* [83].

Tempo real *soft* — Sistema de tempo real *soft* é aquele no qual o desempenho é degradado mas não inutilizado pela falha em atingir restrições quanto ao tempo de resposta, como no caso de um caixa eletrônico ou um processador de texto [83].

Thread — *Thread*, ou linha de execução em português, é uma forma de um processo executar duas ou mais tarefas de forma simultânea ou não . *Threads* de um programa não são considerados processos completos, mas pequenas porções do processo executando de maneira concorrente (ou em paralelo) [84]. O suporte à *thread* é fornecido pelo próprio sistema operacional (SO), no caso de uma *thread* em nível de *kernel* (*Kernel-Level Thread* ou KLT), ou implementada através de uma biblioteca baseada em uma determinada linguagem (*thread* de nível de usuário ou *User-Level Thread*, ULT).

***Web Ontology Language* (OWL)** — O OWL foi uma das tecnologias fundamentais para o desenvolvimento da web semântica, desenvolvida para inserir

[8]Para maiores informações: `http://www.w3.org/TR/scxml/`. Acesso em: 15/09/2010

significado no conteúdo da web e assim, permitir que computadores o analisem. [9].

[9]Maiores informações podem ser obtidas em: http://www.w3.org/TR/owl-features/. Acesso em: 15/09/2010.