UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO MESQUITA FILHO"
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA – DEE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

TESE DE DOUTORADO

Arquitetura para publicação e universalização do acesso a transdutores inteligentes

Alexandre Alves de Lima Ribeiro

Orientador: Prof. Dr. Alexandre César Rodrigues da Silva

Ilha Solteira – SP
Dezembro de 2012

Alexandre Alves de Lima Ribeiro

Arquitetura para publicação e universalização do acesso a transdutores inteligentes

Tese de Doutorado apresentada à Faculdade de Engenharia - UNESP - *Campus* de Ilha Solteira, como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica.
Área de Conhecimento: Automação.

Orientador: Prof. Dr. Alexandre César Rodrigues da Silva

Ilha Solteira – SP
Dezembro de 2012

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação
Serviço Técnico de Biblioteca e Documentação da UNESP - Ilha Solteira.

Ribeiro, Alexandre Alves de Lima.
R484a Arquitetura para publicação e universalização do acesso a transdutores inteligentes / Alexandre Alves de Lima Ribeiro. – Ilha Solteira: [s.n.], 2012.
194 f. : il.

Tese (doutorado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2012

Orientador: Alexandre César Rodrigues da Silva
Inclui bibliografia

1. RESTful. 2. Sistemas embarcados. 3. SOA. 4. Transdutores inteligentes. 5. *Web service*.

unesp

UNIVERSIDADE ESTADUAL PAULISTA

CAMPUS DE ILHA SOLTEIRA

FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

CERTIFICADO DE APROVAÇÃO

TÍTULO: Arquitetura para publicação e universalização do acesso a Transdutores Inteligentes

AUTOR: ALEXANDRE ALVES DE LIMA RIBEIRO

ORIENTADOR: Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovado como parte das exigências para obtenção do Título de DOUTOR EM ENGENHARIA ELÉTRICA , Área: AUTOMAÇÃO, pela Comissão Examinadora:

Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. ERNANDES ROCHA DE OLIVEIRA
Departamento de Matemática / Faculdade de Engenharia de Ilha Solteira

Profa. Dra. ERICA REGINA M D MACHADO
Departamento de Matemática / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. EVANDRO DE ARAUJO JARDINI
Departamento De / Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

Prof. Dr. JOÃO HENRIQUE KLEINSCHMIDT
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas / Universidade Federal do Abc

Data da realização: 14 de dezembro de 2012.

AGRADECIMENTOS

Ao meu orientador, em especial pela liberdade que proporcionou a esta pesquisa.

Aos amigos do LPSSD, pelo essencial apoio aos estudos e ao café.

Ao PPGEE e ao DEE, pelas condições e infraestrutura oferecidas para o desenvolvimento do trabalho.

Ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP), pelo incentivo em horas proporcionado para o desenvolvimento deste trabalho.

A todos que de alguma forma colaboraram com este trabalho.

À minha família, pela paciência e incentivo.

A Deus, pelas oportunidades e dádivas em minha vida.

RESUMO

Neste trabalho é proposta uma arquitetura para publicação e universalização do acesso a transdutores inteligentes. Foram realizadas algumas pesquisas específicas sobre plataformas embarcadas com suporte à pilha de protocolos TCP/IP, arquiteturas orientadas a serviços e tecnologias para o desenvolvimento de aplicações clientes. As pesquisas realizadas contribuíram para a definição da proposta de arquitetura orientada a serviços baseada em transdutores inteligentes RESTful, *gateways* de tradução de padrões e serviços de retaguarda operando em computação em nuvem. Também são apresentadas algumas implementações e resultados obtidos com o desenvolvimento de alguns mecanismos e aplicações sobre a arquitetura. Esta arquitetura possibilita a operação de transdutores inteligentes em alto nível de abstração como serviços.

Palavras-chave: RESTful. Sistemas Embarcados. SOA. Transdutores Inteligentes. *Web Service*.

ABSTRACT

This work proposes an architecture for publication and universal access to smart transducers. Some specific research has been done on embedded platforms with support for the TCP/IP, service-oriented architectures and technologies for the development of client applications. The surveys have contributed to the definition of the proposed service-oriented architecture based on RESTful smart transducers, translation gateways of standards and back-office services operating on cloud computing. Also shown are some implementations and results obtained with the development of some mechanisms and applications over the architecture. This architecture enables the operation of smart transducers at a high level of abstraction as services.

Keywords: Embedded Systems. RESTful. Smart Transducers. SOA. Web Service.

RESUMEN

Este trabajo propone una arquitectura para la publicación y el acceso universal a los transductores inteligentes. Algunas investigación específica se llevaron a cabo en plataformas embebidas con soporte para el protocolo TCP/IP, arquitecturas orientadas a servicios y tecnologías para el desarrollo de aplicaciones cliente. Las encuestas han contribuido a la definición de la propuesta de arquitectura orientada a servicios basada en transductores inteligentes RESTful, *gateways* de traducción de la normas y servicios de *back-office* que operan en la computación en la nube. También se muestran algunas implementaciones y resultados obtenidos con el desarrollo de algunos mecanismos y aplicaciones en la arquitectura. Esta arquitectura permite la operación de los transductores inteligentes a un alto nivel de abstracción como servicios.

Palabras clave: RESTful. Sistemas Embebidos. SOA. Transductores Inteligentes. *Web Service*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AD *Analog-to-Digital*
ADC *Analog-to-Digital Converter*
AIR *Adobe Integrated Runtime*
AJAX *Asynchronous Javascript and* XML
API *Application Programming Interface*
ARM *Advanced* RISC *Machine*
ASCII *American Standard Code for Information Interchange*
B2B *Business-to-Business*
CAN *Controller Area Network*
CGI *Common Gateway Interface*
CiA CAN *in Automation*
CLP Controlador Lógico Programável
COM *Component Object Model*
CORBA *Common Object Request Broker Architecture*
CPLD *Complex Programmable Logic Device*
CPU *Central Processing Unit*
CRUD *Create, Read, Update and Delete*
DC *Direct Current*
DCOM *Distributed* COM
DSP *Digital Signal Processor*
EDS *Embedded Design Suite*
EE *Enterprise Edition*
EPEL *Extra Packages for Enterprise Linux*
Ext3 *Third Extended file system*
FAT *File Allocation Table*
FPGA *Field Programmable Gate Array*
FPU *Float Point Unit*
FTP *File Transfer Protocol*
GA *General Availability*
GCC GNU *Compiler Collection*
GPL *General Public License*
HTTP *HyperText Transfer Protocol*
IDE *Integrated Development Environment*
IDL *Interface Definition Language*
IEEE *Institute of Electrical and Electronics Engineers*
I/O *Input/Output*
I2C *Inter-Integrated Circuit*
ICSP *In-Circuit Serial Programming*
ID *IDentifier*
IoT *Internet of Things*
IP *Internet Protocol*
IPX *Internetwork Packet eXchange*

ISO *International Organization for Standardization*
JAX-RPC *Java* API *for* XML *- based Remote Procedure Call*
JAX-WS *Java* API *for* XML *- based Web Service*
JAXB *Java Architecture for* XML *Binding*
JRE *Java Runtime Environment*
JSON *JavaScript Object Notation*
JVM *Java Virtual Machine*
LCD *Liquid Crystal Display*
LPSSD Laboratório de Processamento de Sinais e Sistemas Digitais
MAC *Media Access Control*
MIPS *Microprocessor without Interlocked Pipeline Stages*
MPFS *Microchip Pic File System*
MVC *Model-view-controller*
NCAP *Network Capable Application Processor*
NTC *Negative Temperature Coefficient*
OASIS *Organization for the Advancement of Structured Information Standards*
OGC *Open Geospatial Consortium*
OLE *Object Linking and Embedding*
OMG *Object Management Group*
OPC *Open Platform Communications*
OPC-UA OPC *Unified Architecture*
OS *Operating System*
OSI *Open Systems Interconnection*
PC *Personal Computer*
PCB *Printed Circuit Board*
PI *Proportional Integral*
PID *Process* ID
PID *Proportional Integral Derivative*
CoAP *Constrained Application Protocol*
RAM *Random Access Memory*
REST *REpresentational State Transfer*
RIA *Rich Internet Application*
RISC *Reduced Instruction Set Computer*
RMI *Remote Method Invocation*
RSS *Really Simple Syndication*
RSSF Rede de Sensores Sem Fio
RTOS *Real-Time* OS
RTU *Remote Terminal Unit*
SAS *Sensor Alert Service*
SCADA *Supervisory Control And Data Acquisition*
SDRAM *Synchronous Dynamic* RAM
SGBD Sistema de Gerenciamento de Banco de Dados
SMTP *Simple Mail Transfer Protocol*
SOA *Service-Oriented Architecture*
SOAP *Simple Object Access Protocol*
SOPC *System On a Programmable Chip*
SOS *Sensor Observation Service*

SPARC *Scalable Processor ARChitecture*
SPS *Sensor Planning Service*
SPX *Sequenced Packet eXchange*
SRAM *Static* RAM
SSH *Secure SHell*
SSW *Semantic Sensor Web*
STIM *Smart* TIM
STWS *Smart Transducers Web Service*
SVG *Scalable Vector Graphics*
SWE *Sensor Web Enablement*
TCP *Transmission Control Protocol*
TEDS *Transducer Electronic Data Sheet*
TI Tecnologia da Informação
TII *Transducer Independent Interface*
TIM *Transducer Interface Module*
TML *Transducer Markup Language*
TJB Transistor de Junção Bipolar
UART *Universal Synchronous Receiver/Transmitter*
UDDI *Universal Description, Discovery and Integration*
URL *Uniform Resource Locator*
USB *Universal Serial Bus*
VHDL VHSIC *Hardware Description Language*
VHSIC *Very-High-Speed Integrated Circuit*
VoIP *Voice over* IP
W3C *World Wide Web Consortium*
WADL *Web Application Description Language*
Web *World Wide Web*
WebGIS *Web Geographic Information System*
WNS *Web Notification Services*
WPF *Windows Presentation Foundation*
WS-I *Web Services Interoperability Organization*
WSDL *Web Service Description Language*
WSN *Wireless Sensor Network*
XML *eXtensible Markup Language*

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO GERAL

1.1 INTRODUÇÃO

Transdutores são dispositivos que convertem energia de uma forma para outra. Um transdutor pode ser um sensor ou um atuador. Sensor é um transdutor que produz algum sinal elétrico, proporcional a uma grandeza física do ambiente em que está submetido. Atuador é um transdutor que por meio de sinais elétricos consegue modificar alguma grandeza física em seu ambiente. Transdutores servem para uma ampla variedade de aplicações de instrumentação e controle de grandezas físicas. Transdutores Inteligentes (*Smart Transducers*) são dispositivos que além de serem transdutores, embarcam alguma capacidade computacional e de comunicação em rede.

Dada à variedade de transdutores e padrões de comunicação encontrados, a interligação e interoperabilidade de transdutores inteligentes, nos mais diversos sistemas, tem se tornado um fator complexo e oneroso para os desenvolvedores e usuários destes dispositivos.

Com o objetivo de diminuir essa complexidade de integração, inúmeros esforços têm sido feitos como, por exemplo, os das especificações IEEE 1451, OGC (*Open Geospatial Consortium*) SWE (*Sensor Web Enablement*) e OPC-UA (*OPC Unified Architecture*).

Em resposta às necessidades da indústria de padronização de interfaces de transdutores, o comitê de tecnologia de sensores da IEEE *Instrumentation and Measurement Society* patrocinou o desenvolvimento de um conjunto de padrões de interfaces para transdutores inteligentes, conhecidos como IEEE 1451 (SONG; LEE, 2008b).

O grupo de trabalho SWE do OGC desenvolve especificações de interface para interoperabilidade e codificação de metadados, para permitir a integração em tempo real de redes de sensores Web heterogêneos, em infraestruturas de informação (OGC, 2011).

A OPC-UA é uma arquitetura independente de plataforma, através do qual diferentes tipos de sistemas e dispositivos podem se comunicar, enviando mensagens entre clientes e servidores de diversos tipos de redes (OPC, 2009). Para isso, a OPC-UA faz uso de protocolos

e padrões como TCP (*Transmission Control Protocol*), XML (*eXtensible Markup Language*), SOAP (*Simple Object Access Protocol*) etc.

No passado a comunicação entre redes de computadores também esbarrava em uma série de dificuldades, muitas delas oriundas da falta de padrões entre as interfaces e protocolos de rede. Um dos esforços muito bem sucedido para a resolução desse problema foi o modelo OSI (*Open Systems Interconnection*) de arquitetura de redes em camadas, proposta em 1983 pela ISO (*International Standards Organization*). Em decorrência desse modelo, surgiram diversas implementações das camadas OSI, muitas delas proprietárias, embora seguissem o modelo de referência OSI (TANENBAUM, 2003).

O que a história provou foi que implementações baseadas em padrões abertos, como a Ethernet (IEEE 802.3), ao se difundirem superaram, ou eliminaram, as demais implementações como, por exemplo, a Token Ring da IBM. A história se repetiu na popularização do conjunto de protocolos de rede TCP/IP (*Transmission Control Protocol / Interner Protocol*), sobre o IPX/SPX[1] da Novell[2] e o NetBEUI[3] empregados pela Microsoft e IBM (EVANS, 2003).

Alguns degraus acima no modelo OSI, mais precisamente no topo da pilha, surgiu no início da década de 90 um protocolo de aplicação muito simples chamado *HyperText Transfer Protocol* (HTTP). Embora simples, ou devido a sua simplicidade, esse protocolo foi um marco histórico para a comunicação em rede, pois foi o alicerce para a *World Wide Web*, ou simplesmente Web. Com a popularização desse protocolo aberto foi possível a troca de textos, até então ASCII (*American Standard Code for Information Interchange*), de forma simples e universal.

Outro fato que a história apresenta é a preferência pelo padrão popular, mesmo que este não seja o mais eficiente. O próprio protocolo HTTP é um exemplo dessa preferência. O protocolo FTP (*File Transfer Profocol*) é mais específico e eficiente para a transferência de arquivos binários, no entanto, por vários motivos, dentre eles a simplicidade e popularidade, o HTTP é amplamente empregado para a transferência de arquivos, mesmo sendo menos eficiente que o FTP. Outro exemplo que evidencia a preferência de padrão popular são as aplicações VoIP (*Voice over Internet Protocol*). Embora o protocolo IP (*Interner Protocol*) não seja projetado para a transferência de áudio, por não ser determinístico em sua rota e latência,

1 Pilha de protocolos proprietários da Novell, operam nas camadas de rede e transporte.
2 Empresa de TI que liderou o mercado de sistemas operacionais de rede, durante a década e 80 até meados da década de 90, com seu sistema Netware (TORRES, 2001).
3 Protocolo proprietário da Microsoft, opera nas camadas de rede e transporte.

é empregado para esse fim por ser o protocolo de rede mais popular do mundo.

Embora o HTTP já tivesse universalizado o meio de transferência de mensagens na Web, os padrões para essa transferência ainda vinculavam forma e conteúdo, como o HTML. Atualmente o *World Wide Web Consortium* (W3C) recomenda o emprego de XML para a estruturação e transferência de dados na rede mundial (W3C, 2008).

Esta recomendação tem sido largamente empregada, proporcionando uma revolução na organização da informação contida na Web, pois o conteúdo tem se desvinculado da forma, permitindo que diversas fontes de informação publiquem seus conteúdos na rede, deixando livre para cada aplicação decidir como e de que forma vão republicar ou consumir esses conteúdos. Outra recomendação da W3C, a WSDL (*Web Service Description Language*), também propõe a descrever em XML o conteúdo e serviços disponíveis na Web.

A W3C também especifica o protocolo SOAP que opera sobre o HTTP, utilizando XML e WSDL para a transferência de informações estruturadas de forma descentralizada e distribuída (W3C, 2007).

Embora existam diversos tipos de serviços Web, neste trabalho será utilizado o termo *Web Service* para referir-se especificamente a dispositivos que provêm conteúdo e serviços Web em XML, sobre o protocolo SOAP. Na seção 3.3 são apresentados alguns elementos, especificações e padrões de *Web Services*.

A seguir são apresentados alguns termos que se relacionam, em maior ou menor grau, com o trabalho desenvolvido:

- Sensores Inteligente (*Smart Sensors*) – Embora adote apenas a palavra *sensores*, este termo também é empregado para atuadores, sendo frequentemente encontrado como sinônimo do termo Trandutores Inteligente, já definido no início desta seção;
- *Sensor Web* – Termo empregado mais frequentemente em aplicações de monitoramento ambiental com RSSFs (Redes de Sensores Sem Fio) (DELIN; JACKSON, 2001). Os trabalhos do OGC SWE adotam esta terminologia (OGC, 2007);
- Sensores Virtuais (*Virtual Sensors*) – São aplicações em software que agregam informações de diversos sensores ou adiciona características ou comportamentos não existentes nos sensores reais, como, por exemplo, tolerância a falhas ou orientação a serviços (ZEQIANG et al., 2011);
- RSSFs (WSNs – *Wireless Sensor Networks*) – São redes de sensores sem fio. Entre

algumas de suas características mais comuns estão o baixo custo por nó, o baixo consumo de energia e as restrições de hardware (AKYILDIZ et al., 2002);

- Internet das Coisas (IoT – *Internet of Things*) – É um conceito baseado em uma Internet com dispositivos (coisas) sempre conectados, com endereços válidos e identificação única, expandindo em tamanho e diversidade a Internet. Assim, serviços e inteligência poderão ser adicionados a esta Internet expandida, fundindo o mundo digital ao mundo físico (COETZEE; EKSTEEN, 2011).

Neste trabalho é adotado o termo Transdutores Inteligentes, em uma arquitetura com elementos que estendem suas funcionalidades. O meio físico da rede (com ou sem fio) não é uma característica preponderante para os elementos da arquitetura proposta, apresentada no capítulo 5.

1.2 OBJETIVOS GERAIS

O objetivo deste trabalho foi desenvolver uma arquitetura para publicação e universalização do acesso a transdutores inteligentes, sobre padrões e protocolos abertos, como o IP, HTTP e XML, permitindo que o desenvolvimento e integração de sistemas de instrumentação e controle possam abstrair detalhes físicos e tecnológicos dos transdutores e de suas redes.

Neste trabalho buscou-se dominar tecnologias como a de plataformas embarcadas, arquiteturas orientadas a serviços (SOA - *Service-Oriented Architecture*) e aplicações distribuídas, contribuindo para a publicação e universalização do acesso a transdutores inteligentes, de forma semelhante ao que o HTTP fez pela Web e o XML pela informação.

1.3 JUSTIFICATIVAS E MOTIVAÇÃO

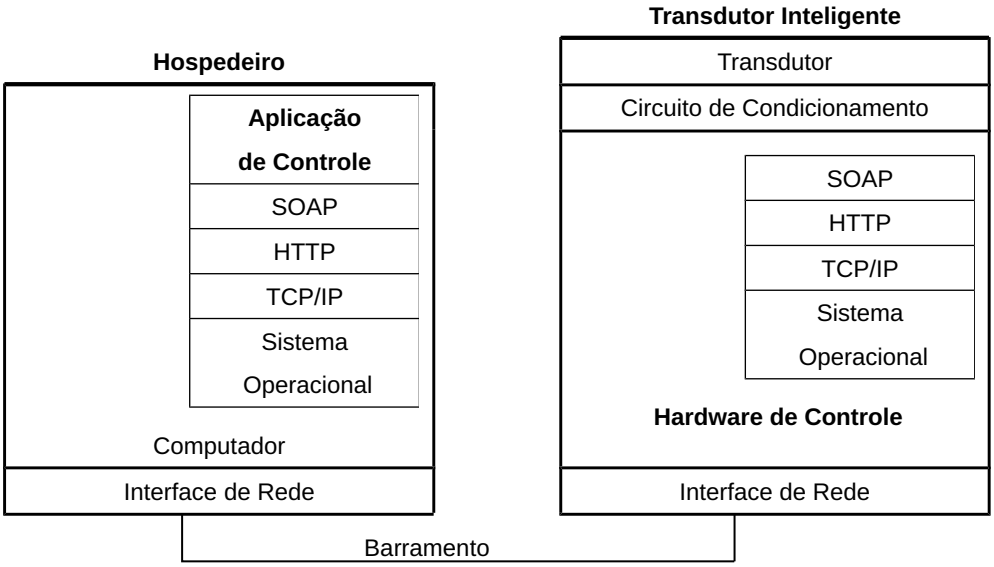
O emprego do protocolo IP e do padrão Ethernet sobre barramentos de campo, como proposto por Girerd et al. (2000), não é novidade. Sensores Web também têm sido propostos nos padrões SWE do OGC (OGC, 2007).

Este trabalho investiga plataformas embarcadas com suporte a TCP/IP, arquiteturas orientadas a serviços e aplicações distribuídas para melhor compreender os elementos que possam compor uma arquitetura para a publicação e universalização do acesso a transdutores

inteligentes.

Na Figura 1 é apresentada uma hipótese de camadas de abstração que permitam a publicação de um transdutor inteligente.

Figura 1: Camadas de abstração para a publicação de transdutores inteligentes.



Fonte: Elaborada pelo autor.

Nesta hipótese um transdutor inteligente necessita de um hardware de controle que embarque um sistema operacional, como o uClinux[4] ou o TinyOS[5], a pilha de protocolos TCP/IP e aplicações que respondam em HTTP ou SOAP.

Ainda como hipótese, transdutores inteligentes podem apenas publicar as informações e recursos como em um *Web Service.*

Desta forma, um transdutor inteligente não necessita realizar controles específicos do processo em que estiver inserido. Esta função computacional pode ficar a cargo de qualquer máquina da rede que hospede uma aplicação de controle do processo. Para tanto é suficiente a aplicação de controle acessar os transdutores da rede pertencentes ao processo.

Com esta estratégia é possível minimizar a necessidade computacional dos transdutores inteligentes e flexibilizar a integração deles em um processo de controle.

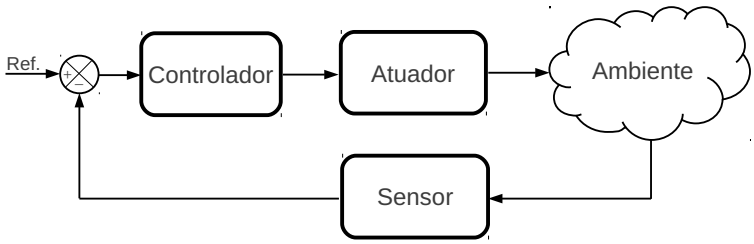
Na Figura 2 ilustra-se um caso típico de controle onde o sensor monitora certa grandeza física no ambiente e o atuador interfere nesta grandeza. O controlador recebe

4 Sistema operacional embarcado baseado em *kernel* Linux e compilado para diversas famílias de microcontroladores (uC).

5 Sistema operacional embarcado projetado especificamente para WSNs.

informações do sensor e decide como atuar em função de um valor de referência.

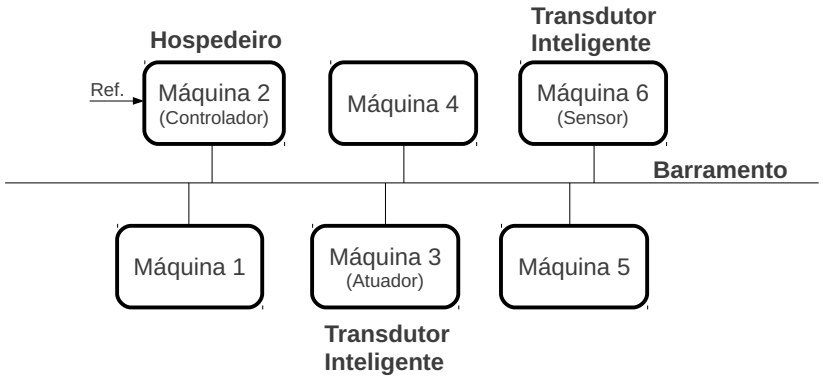
Figura 2: Caso típico de controle.



Fonte: Elaborada pelo autor.

Na Figura 3 é apresentada uma possível distribuição dos elementos do sistema apresentado na Figura 2, em uma rede de transdutores inteligentes.

Figura 3: Rede de transdutores inteligentes.



Fonte: Elaborada pelo autor.

Nesta arquitetura, um transdutor inteligente (máquina 6 - *sensor*) necessita publicar as informações de suas medições de forma que o hospedeiro (máquina 2 - *controlador*) possa acessar essas informações. O hospedeiro pode ser qualquer máquina da rede. Da mesma forma, outro transdutor inteligente (máquina 3 - *atuador*) deve permitir que o controlador acesse seus recursos, a fim de interferir na grandeza física controlada.

Nota-se que bastará aos transdutores inteligentes publicarem seus recursos para que uma aplicação na rede possa utilizá-los, independente de linguagem computacional ou sistema operacional.

Nesta arquitetura, quando comparada com o padrão IEEE 1451, o NCAP (*Network Capable Application Processor*) deveria publicar os recursos dos TIMs (*Transducer Interface*

Module) no barramento de usuário, onde seriam inseridas aplicações de controle que acessassem os NCAPs para compor a aplicação.

Esta estratégia pode ser comparada à de arquiteturas computacionais MIMD (*Multiple Instruction Multiple Data*) (STALLINGS, 2000), onde apenas uma máquina da rede atua como mestre e as demais como escravos. Neste caso a máquina hospedeira com a aplicação de controle seria o mestre e os transdutores inteligentes os escravos.

Salienta-se que problemas de latência de rede, sincronização, granularidade do recurso e escalabilidade, também são questões a serem trabalhadas. É de se esperar que aplicações em um nó qualquer da rede, como o controlador, fracamente acoplada aos transdutores, tenha menor eficiência do que se estivessem fortemente acopladas em hardware dedicado, no próprio transdutor inteligente como, por exemplo, em um FPGA (*Field Programmable Gate Array*) ou DSP (*Digital Signal Processor*). No entanto, visto a abundante oferta de capacidade de processamento em redes de computadores, como na computação em nuvem, torna-se possível explorar essa capacidade em alto nível de abstração, por meio de padrões de comunicação bem definidos. Desta forma pode-se racionalizar o uso de recursos computacionais nos transdutores inteligentes.

Computação em nuvem ou c*loud computing* refere-se ao rápido desenvolvimento e integração de diversos recursos computacionais baseados em TCP/IP (CHUNYE et al., 2010). Buyya et al. (2009) apontam que a computação em nuvem é uma evolução da computação em *cluster (cluster computing)* e da computação em grade *(grid computing)*.

A publicação padronizada de recursos também simplifica o trabalho de desenvolvedores e integradores de sistemas SCADA[6] (*Supervisory Control And Data Acquisition*), pois será suficiente a estes acessar os recursos publicados, como em um *Web Service.* Cabe observar que o trabalho de desenvolver aplicações de controle, como o controlador do exemplo apresentado, continuará exigindo profissionais de engenharia de instrumentação e controle. No entanto, o desenvolvimento e integração de sistemas SCADA poderá ser realizado por profissionais de TI (Tecnologia da Informação), sem, necessariamente, possuírem conhecimentos específicos de instrumentação e controle. Será suficiente a esses profissionais acessarem os recursos publicados pelos transdutores inteligentes, em padrões universais como o HTTP e XML.

6 Sistema de monitoramento (supervisão) e controle de processos industriais.

1.4 TRABALHOS RELACIONADOS

Os grupos de trabalho do IEEE 1451 estão atualmente divididos em diversos comitês. O IEEE 1451.0 define um conjunto de comandos e operações comuns e a TEDS (*Transducer Electronic Data Sheet*) da família de padrões. A TEDS é uma descrição eletrônica gravada na memória interna dos transdutores, ou do NCAP, com informações de identificação, calibração, escalas de medição, unidades de medida etc. (IEEE, 2007).

O IEEE 1451.1 define um modelo de objeto comum, denominado NCAP, que descreve o comportamento e as interfaces de comunicação em rede dos transdutores inteligentes (IEEE, 2000). O NCAP opera como um *gateway*[7] de rede, de um lado comunica-se com transdutores por meio de barramentos de campo e de outro com outros NCAPs ou máquinas da rede, por meio de um barramento de usuários.

Os demais comitês do IEEE 1451 não são referenciados neste trabalho por tratarem de barramentos de campo, enquanto a proposta de arquitetura desenvolvida e apresentada no capítulo 5, refere-se ao barramento de usuários.

O grupo de trabalho SWE do OGC (OGC SWE) desenvolve especificações de interface para interoperabilidade e codificação de metadados, para permitir a integração em tempo real de redes de sensores Web heterogêneos, em infraestruturas de informação (OGC, 2011). As propostas recentes de extensão do padrão IEEE 1451.1, para suportar requisições de serviços Web, convergem com muitos dos requisitos do OGC SWE. Este por sua vez já adotava em suas propostas de arquitetura, a interoperabilidade dos sensores Web com o padrão IEEE 1451 (OGC, 2007).

O projeto Open1451 (SONG, 2004) fornece um repositório para implementações, exemplos e aplicações do padrão IEEE 1451. Neste repositório encontram-se códigos de implementações em linguagem Java e C++ do modelo de objetos descrito no padrão IEEE 1451.1, no entanto este projeto não está muito ativo a alguns anos.

1.5 ORGANIZAÇÃO DO TEXTO

Neste capítulo 1 foram introduzidas as motivações, objetivos e justificativas deste trabalho, contextualizando com a proposta.

No capítulo 2 são explorados recursos de diferentes plataformas embarcadas com

7 Dispositivo que permite interligar diferentes redes de dados.

suporte a TCP/IP, que podem embarcar transdutores inteligentes.

No capítulo 3 são investigados mecanismos e padrões de arquiteturas orientadas a serviços que podem servir de *middleware* entre os transdutores inteligentes e as aplicações distribuídas que os acessam.

No capítulo 4 são investigadas algumas tecnologias, linguagens de programação e recursos de comunicação distribuída que possam servir para o desenvolvimento das aplicações clientes da arquitetura proposta.

No capítulo 5 é apresentada um modelo de arquitetura para publicação e universalização do acesso a transdutores inteligentes.

No capítulo 6 são apresentadas algumas implementações de elementos da arquitetura proposta e estudos de casos de aplicações sobre a arquitetura.

Nas seções seguintes são apresentadas, respectivamente, as conclusões gerais deste trabalho, os trabalhos publicados e aceitos e as referências utilizadas.

CAPÍTULO 2

PLATAFORMAS EMBARCADAS COM SUPORTE A TCP/IP

2.1 INTRODUÇÃO

Sistemas embarcados são encontrados em uma infinidade de aplicações das mais diversas áreas como: aparelhos eletrônicos de consumo, eletrodomésticos, equipamentos médicos hospitalares, equipamentos industriais, equipamentos agropecuários etc., apresentando-se também em uma infinidade de configurações de hardware e software para cada aplicação encontrada.

Dentre algumas das plataformas de hardware para sistemas embarcados, uma das mais populares são as microcontroladas, que embarcam no mesmo *chip* diversos periféricos de entrada e saída de dados e componentes do sistema. Microcontroladores são em geral de baixo custo, versáteis e apresentam capacidade restrita de processamento, se comparados a outras plataformas de hardware.

Processadores digitais de sinais (DSPs) também são muito empregados, principalmente quando a aplicação exige maior capacidade de processamento, ou mesmo repostas em tempo real.

Processadores de propósito geral, como os de arquiteturas: i386, ARM, AVR, PowerPC, SPARC, MIPS etc., normalmente são encontrados em plataformas de hardware sob sistemas operacionais embarcados, como: TinyOS, uClinux, Microsoft Windows Embedded, eCos, FreeRTOS etc., embora também sejam adotados em soluções *standalone* (OLIVEIRA; ANDRADE, 2006).

Os dispositivos reconfiguráveis como CPLDs (*Complex Programmable Logic Device*) e FPGAs completam a lista das principais soluções de hardware para sistemas embarcados, sendo que em algumas soluções com FPGAs de maior capacidade lógica, são encontrados *cores*[8] de processadores de propósito geral, sintetizados em alguma linguagem de descrição de hardware, como a VHDL *(*VHSIC *Hardware Description Language)*, que também permite o

8 Núcleo de processamento de dados.

emprego de sistemas operacionais embarcados.

Para a comunicação entre sistemas embarcados e outros dispositivos são adotados os mais diversos padrões de interfaces de comunicação da indústria, como: RS232, RS485, I2C, USB, CAN, Ethernet, Bluetooth, ZigBee etc., cada um deles mais adequado a uma dada aplicação do que os outros.

Para esta pesquisa teve-se acessa a quatro plataformas embarcadas com suporte a TCP/IP:

- PME-10 – é uma placa microcontrolada de 8 bits (PIC 18F8720 da Microchip (2004)) com interfaces Ethernet e RS232 (2EI, 2007);
- DB-DP11115 – é uma placa microcontrolada de 16 bits (PIC 24FJ256 da Microchip (2009)) com interface Ethernet, RS232 e USB, além de alguns dispositivos de I/O (SURE, 2010);
- DE2 – é uma placa de prototipação com um FPGA (Cyclone II 2C35F672C6 da Altera (2008)) como principal unidade de hardware reconfigurável, um CPLD, memórias SDRAM, SRAM e Flash, e diversos periféricos como interfaces Ethernet, RS232 e USB (ALTERA, 2006);
- NGW100 – é um *gateway* embarcado com um microcontrolador de 32 bits (AT32AP7000 da Atmel (2009)) de arquitetura AVR32, com conjunto de instruções RISC (*Reduced Instruction Set Computer*) e DSP, que caracterizam este microcontrolador também como um DSP, com diversas interfaces de comunicação como duas Ethernets, I2C, USB e RS232 (AVRFREAKS, 2009).

2.1.1 Objetivos específicos

O objetivo das pesquisas desenvolvidas neste capítulo foi de familiarizar-se com o desenvolvimento de aplicações sobre plataformas embarcadas com comunicação em rede, visando evoluir um degrau em outro objetivo maior, o de definir uma arquitetura para publicação e universalização do acesso a transdutores inteligentes.

Para tanto, foi proposto investigar dentre as diversas plataformas de hardware para sistemas embarcados, as que suportassem ao menos a pilha de protocolos TCP/IP ou sistemas operacionais embarcados. Dentre as várias opções disponíveis, desejou-se compreender as principais diferenças quanto à capacidade computacional, conectividade, suporte a protocolos

como TCP/IP e HTTP, sistemas operacionais embarcados, serviços embarcados como servidor Web e *Web Services*, custo, dimensões e consumo de energia.

2.1.2 Justificativas

Embora seja esperada uma disparidade computacional entre dispositivos como o PIC18F8720 (MICROCHIP, 2004) e o AT32AP7000 (ATMEL, 2009), que pode sugerir resultados previsíveis em suas comparações, foi exatamente esta diferença de recursos disponíveis para o desenvolvedor de aplicações que enriqueceu a pesquisa, frente ao objetivo primário de familiarizar-se com o desenvolvimento de aplicações sobre plataformas embarcadas.

A escolha da comunicação sobre a pilha de protocolos TCP/IP, que neste trabalho foi conseguido com o protocolo de aplicação HTTP, vem ao encontro do objetivo maior relatado anteriormente. Embora seja conhecida a dificuldade do não determinismo apresentado pelo protocolo IP, este é o protocolo adotado na Internet, rede a qual se pretende prover acesso aos transdutores inteligentes.

Quando as pesquisas deste capítulo foram realizadas utilizou-se apenas as plataformas embarcadas PME-10, DE2 e NGW100. Com as conclusões deste capítulo decidiu-se empregar a plataforma NGW100, de maior capacidade computacional, para as pesquisas do capítulo seguinte. Entretanto, na sequência da pesquisa nos capítulos 5 e 6, decidiu-se empregar plataformas mais restritas, transferindo demandas computacionais para elementos em software na rede de computadores. Para tanto, seria possível a adoção da placa PME-10, se esta ainda estivesse operacional na ocasião em que se desenvolvia os estudos de caso apresentados no capítulo 6.

Com a indisponibilidade da placa PME-10, se adquiriu a placa DB-DP11115 da Sure Electronics, também de arquitetura microcontrolada e com interface Ethernet. Com a plataforma DB-DP11115, se repetiu os procedimentos realizados com as demais plataformas embarcadas, incluindo os resultados obtidos neste capítulo.

2.1.3 Trabalhos relacionados

Plataformas de hardware como as empregadas nesta pesquisa são utilizadas para as mais diversas aplicações. Merhi, Elgamel e Bayoumi (2009) da Universidade da Louisiana,

em Lafayette, desenvolveram um sistema colaborativo tolerante a falhas, de localização de alvos para redes de sensores sem fio. Este sistema foi desenvolvido sobre a plataforma de hardware MICAz[9], com o sistema operacional embarcado TinyOS e uma placa microcontrolada com o PIC 18F8720, o mesmo microcontrolador da plataforma PME-10.

Kuczenski, LeDuc e Messner (2005) da Universidade Carnegie Mellon desenvolveram uma plataforma de software sobre a licença GPL (*General Public License*) (FSF, 2007) para o desenvolvimento de aplicações de instrumentação e controle com microcontroladores da família PIC 18F.

Joshi, Dakhole e Zode (2009) do Colégio de Engenharia Yashwantrao Chavan de Nagpur, implementaram um servidor Web sobre um FPGA, com o processador embarcado NIOS II. Este trabalho tem semelhança com a aplicação desenvolvida, sobre a plataforma DE2, apresentada neste capítulo. No entanto, a aplicação desenvolvida, além de permitir familiarizar-se com o desenvolvimento de aplicações sobre plataformas embarcadas com comunicação em rede, como previsto nos objetivos, também estende o trabalho citado ao comparar esta plataforma embarcada com outras duas de arquiteturas distintas.

Savochkin et al. (2008) da Universidade Técnica Nacional de Sevastopol, desenvolveram um sistema de monitoramento e controle de ambientes baseado na plataforma NGW100, sobre o microcontrolador AT32AP7000 e um sistema operacional embarcado Linux.

Rossi et al. (2009), da Universidade Estadual Paulista, implementaram um NCAP e um STIM (*Smart* TIM) no padrão IEEE 1451. O NCAP foi implementado em Java sobre uma plataforma PC e parte em uma plataforma baseada em um FPGA da família FLEX da Altera. O módulo STIM com interface TII (*Transducer Independent Interface*) foi implementado em uma plataforma baseada em um FPGA ACEX da Altera. O NCAP acessava o barramento de usuário por interface Ethernet e o barramento de campo por interface TII. A interface TII do NCAP foi implementada em uma plataforma baseada em um FPGA FLEX da Altera, interligado a plataforma PC por uma porta paralela.

2.1.4 Organização do capítulo

Na seção 2.2 deste capítulo apresentam-se as plataformas embarcadas empregadas nesta pesquisa.

9 Plataforma da Moog Crossbow baseada em microcontrolador Atmel da família megaAVR.

Na seção 2.3 são descritos os procedimentos e ferramentas adotadas para desenvolvimento de aplicações para estas plataformas.

Na seção 2.4 são apresentados os resultados obtidos de diversas medições realizadas.

Na seção 2.5 são relatadas as conclusões deste capítulo.

Na seção 2.6 são elencadas algumas propostas de trabalhos futuros relacionadas às plataformas de hardware.

2.2 PLATAFORMAS EMBARCADAS EMPREGADAS

A seguir são apresentadas as plataformas embarcadas com suporte a TCP/IP disponíveis e empregadas nas pesquisas deste capítulo. Na sequência dos trabalhos, algumas destas plataformas foram utilizadas nas pesquisas desenvolvidas nos demais capítulos.

2.2.1 PME-10

A PME-10 é uma placa microcontrolada de 8 bits com interfaces Ethernet e RS232. Na Figura 4 apresenta-se uma imagem desta placa e na Tabela 1 relacionam-se algumas de suas características.

Figura 4: Placa PME-10.

Fonte: Elaborada pelo autor.

Tabela 1: Características da placa PME-10.

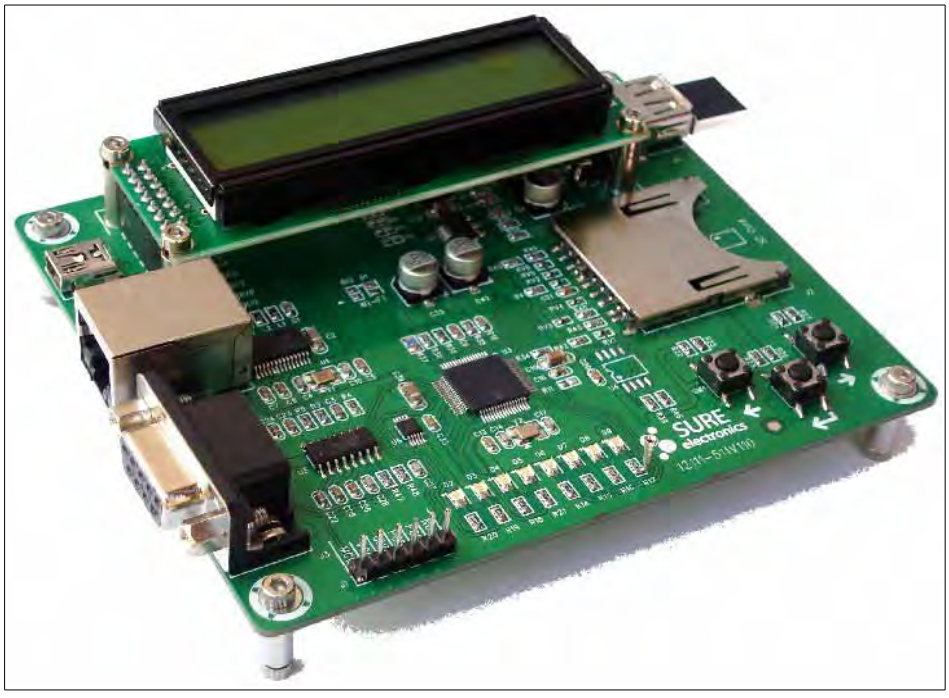
Característica	Descrição
Chip principal	Microcontrolador PIC18F8720 da Microchip
Clock	20 MHz
SRAM	3840 B
Flash	128 kB
Ethernet	10 Mbps
Dimensões	50 mm x 60 mm x 30 mm
Modelo	PME-10
Fabricante	2EI Engenharia

Fonte: 2EI Engenharia (2007).

2.2.2 DB-DP11115

A DB-DP11115 é uma placa microcontrolada de 16 bits com interfaces Ethernet, RS232 e USB, além de alguns dispositivos de I/O e transdutores *onboard*. Na Figura 5 apresenta-se uma imagem desta placa e na Tabela 2 relacionam-se algumas de suas características.

Figura 5: DB-DP11115.



Fonte: Elaborada pelo autor.

Tabela 2: Características da placa DB-DP11115.

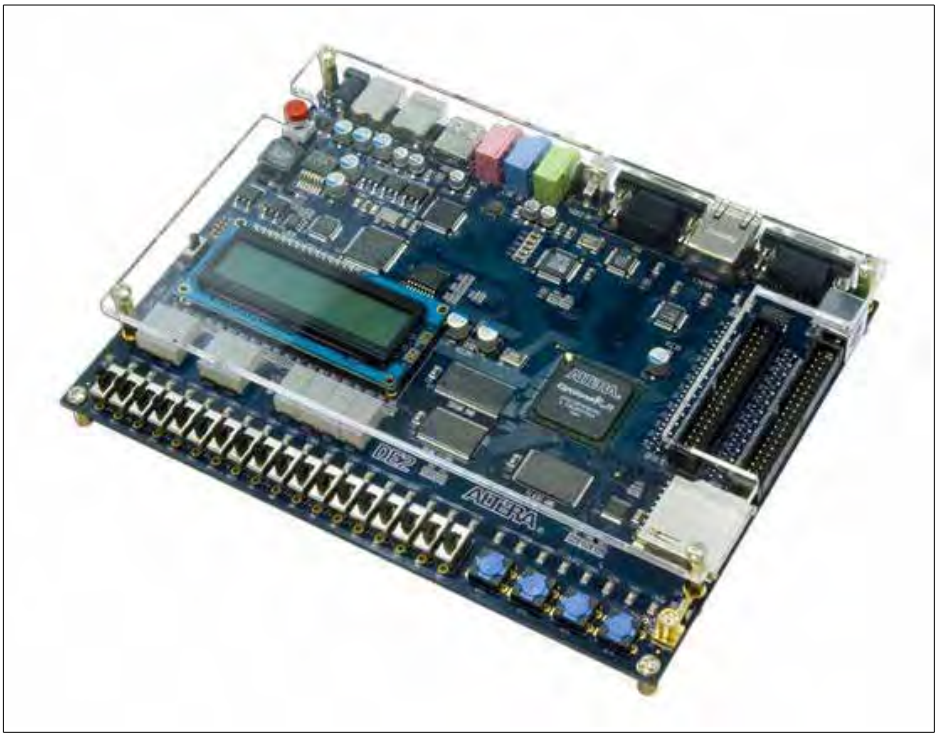
Característica	Descrição
Chip principal	Microcontrolador PIC24FJ256GB106 da Microchip
Clock	20 MHz
SRAM	16 kB
Flash	256 kB
Ethernet	10 Mbps
Dimensões	120.6 mm x 96.7 mm x 24.7 mm
Modelo	DB-DP11115
Fabricante	Sure Electronics

Fonte: Sure Electronics (2010).

2.2.3 DE2

A DE2 é uma placa de prototipação com um FPGA como principal unidade de hardware reconfigurável, um CPLD, memórias SDRAM, SRAM e Flash e diversos periféricos como interfaces Ethernet, RS232 e USB. Na Figura 6 apresenta-se uma imagem desta placa e na Tabela 3 relacionam-se algumas de suas características.

Figura 6: Placa DE2.



Fonte: Altera Corporation (2011).

Tabela 3: Características da placa DE2.

Característica	Descrição
Chip principal	FPGA Cyclone II 2C35F672C6 da Altera
Clock	50 MHz
SRAM	512 kB
SDRAM	8 MB
Flash	4 MB
Ethernet	100 Mbps
Dimensões	152 mm x 203 mm x 37 mm
Modelo	DE2
Fabricante	Altera

Fonte: Altera Corporation (2006).

2.2.4 NGW100

A placa NGW100 é um *gateway* embarcado com um microcontrolador de 32 bits de arquitetura AVR32, com conjunto de instruções RISC e DSP, que caracterizam este microcontrolador também como um DSP, com diversas interfaces de comunicação como duas Ethernets, I2C, USB e RS232. Na Figura 7 apresenta-se uma imagem desta placa e na Tabela 4 relacionam-se algumas de suas características.

Figura 7: Placa NGW100.

Fonte: Elaborada pelo autor.

Tabela 4: Características da placa NGW100.

Característica	Descrição
Chip principal	Microcontrolador AT32AP7000 da Atmel
Clock	150 MHz
SRAM	32 kB
SDRAM	32 MB
Flash	16 MB
Ethernet	100 Mbps
Dimensões	100 mm x 127 mm x 20 mm
Modelo	NGW100
Fabricante	Atmel

Fonte: AVRFreaks Wiki (2009).

2.3 PROCEDIMENTOS EXPERIMENTAIS

A aplicação escolhida para ser desenvolvida nas diversas plataformas embarcadas foi o clássico método de ordenação *Bubble Sort* (CORMEN; LEISERSON; RIVEST, 2001). Esta escolha se deu tanto pela simplicidade de implementação, quanto pela sua complexidade computacional $O(n^2)$. Como um dos objetivos foi comparar a capacidade de processamento nas plataformas embarcadas, a escolha de um método de maior complexidade computacional mostrou-se mais adequado que um método mais eficiente.

A comunicação com a aplicação desenvolvida, para cada uma das plataformas, foi realizada pelo protocolo HTTP, sobre TCP/IP, pela interface Ethernet de cada placa. A entrada de dados para a aplicação embarcada se deu pela composição da URL (*Uniform Resource Locator*) do método GET, do protocolo HTTP, utilizado para as requisições das aplicações clientes e a saída de dados pelo retorno do protocolo HTTP à requisição cliente.

A seguir são apresentadas as ferramentas empregadas, assim como os ambientes configurados para o desenvolvimento das aplicações.

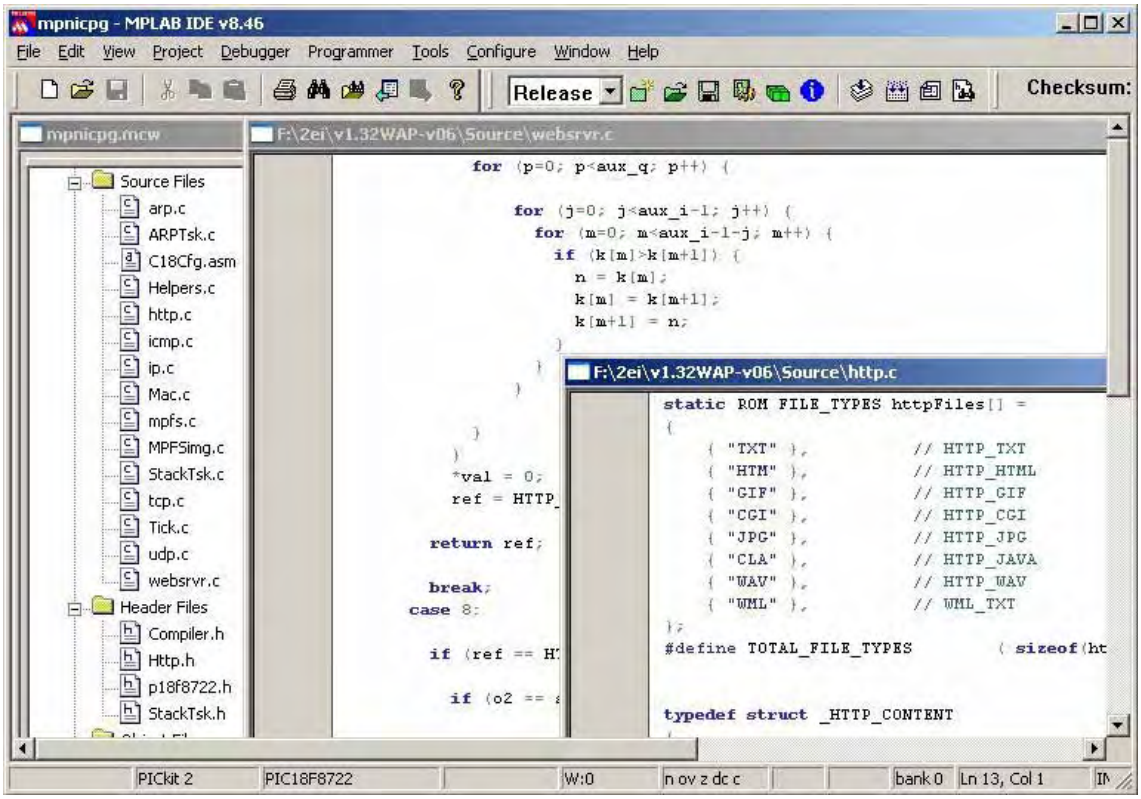
2.3.1 PME-10

Para o desenvolvimento da aplicação na plataforma PME-10 foi utilizado o IDE (*Integrated Development Environment*) MPLAB 8.46 da Microchip, fabricante do microcontrolador utilizado na placa, partindo do projeto v1.32WAP de demonstração,

fornecido pela 2EI Engenharia, fabricante da placa. Este IDE permite ao desenvolvedor escrever sua aplicação na linguagem de programação C e compilá-la para o conjunto de instruções específicas do microcontrolador alvo. Para tanto basta especificar na ferramenta o microcontrolador alvo, que neste caso foi o PIC18F8720.

Na Figura 8 apresenta-se a interface do IDE MPLAB com parte do código C do método *Bubble Sort* implementado. Como pode ser observada a sintaxe é idêntica a do desenvolvimento para arquiteturas convencionais, exceto pelas restrições de tipos de dados e uso de memória que a arquitetura do microcontrolador impõe.

Figura 8: Interface da IDE MPLAB 8.46 da Microchip.



Fonte: Elaborada pelo autor.

Para a programação da comunicação com a interface Ethernet, foi utilizada uma implementação da pilha TCP/IP fornecida pela Microchip, conforme relação de arquivos apresentados na Tabela 5, que além da camada MAC (*Media Access Control*) e os protocolos IP, TCP e HTTP, também fornece a implementação de um micro servidor Web.

Os arquivos relacionados na Tabela 5 podem ser editados para configurar, entre outras coisas, o endereço MAC, o endereço IP, a porta TCP em que o microcontrolador responderá

às requisições HTTP e as extensões de arquivos suportadas pelo servidor Web etc.

Tabela 5: Arquivos da pilha TCP/IP da Microchip.

Arquivo	Descrição
arp.c	Implementa o protocolo ARP da camada de enlace
icmp.c	Implementa o protocolo ICMP que responde ao *ping*
mac.c	Implementa o endereço MAC
ip.c	Implementa o protocolo IP da camada de rede
tcp.c	Implementa o protocolo TCP da camada de transporte
udp.c	Implementa o protocolo UDP da camada de transporte
http.c	Implementa o protocolo HTTP da camada de aplicação
websrvr.c	Implementa um micro servidor Web

Fonte: Elaborada pelo autor.

Para o armazenamento de arquivos neste servidor Web a Microchip implementa o sistema de arquivos MPFS (*Microchip Pic File System*). Os arquivos das páginas Web a serem armazenadas neste servidor devem ser convertidas para este formato de sistema de arquivos (2EI, 2007), o que pode ser feito pelo aplicativo *MPFS.exe* com a linha de comando a seguir:

```
mpfs [pasta com os arquivos] MPFSImg.c /c
```

Compilando a aplicação e descarregando-a no microcontrolador já será possível acessar páginas estáticas no microcontrolador por meio de um cliente HTTP, como um navegador Web. Para descarregar a aplicação compilada no MPLAB para o microcontrolador foi utilizado um programador USB PICkit 2, interligado à interface ICSP (*In-Circuit Serial Programming*) da placa PME-10.

Para desenvolver páginas dinâmicas que interajam com aplicações no microcontrolador, o servidor Web suporta páginas CGI[10] (*Common Gateway Interface*), que devem ser armazenadas com a extensão *.cgi*. Nestas páginas podem ser colocadas variáveis precedidas pelo caractere *%* (de %00 à %99) que invoca o método *HTTPGetVar()* do servidor Web.

Os principais métodos utilizados no servidor Web foram o *void HTTPExecCmd(unsigned char** argv, unsigned char argc)*, que é invocado quando uma

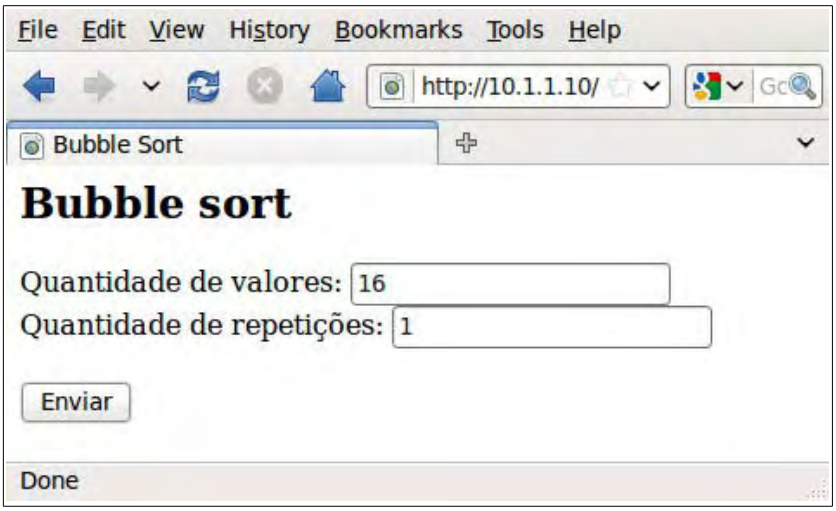
10 Tecnologia que permite gerar páginas Web dinamicamente por meio de execução de *scripts* no servidor Web.

aplicação cliente faz uma requisição HTTP para este servidor, *argv* recebe os argumentos enviados pela aplicação cliente e *argc* registra a quantidade de argumentos recebidos e o *WORD HTTPGetVar(unsigned char var, WORD ref, unsigned char* val)* que é disparado toda vez que for encontrada em uma página CGI uma variável, como descrito anteriormente. Neste instante a aplicação do microcontrolador é invocada e pode retornar o dado que desejar neste ponto da página. O parâmetro *var* identifica a variável invocada, *ref* é um parâmetro de controle para o retorno de dados e *val* retorna um byte, ou caractere, por vez para a página CGI, enquanto não for atribuído *HTTP_END_OF_VAR* para *ref*.

Assim, foi possível implementar a entrada e saída de dados da aplicação de ordenação, por meio de uma página Web hospedada no microcontrolador.

Na Figura 9 pode ser observada a interface Web da aplicação hospedada no microcontrolador.

Figura 9: Interface Web da aplicação hospedada no microcontrolador.

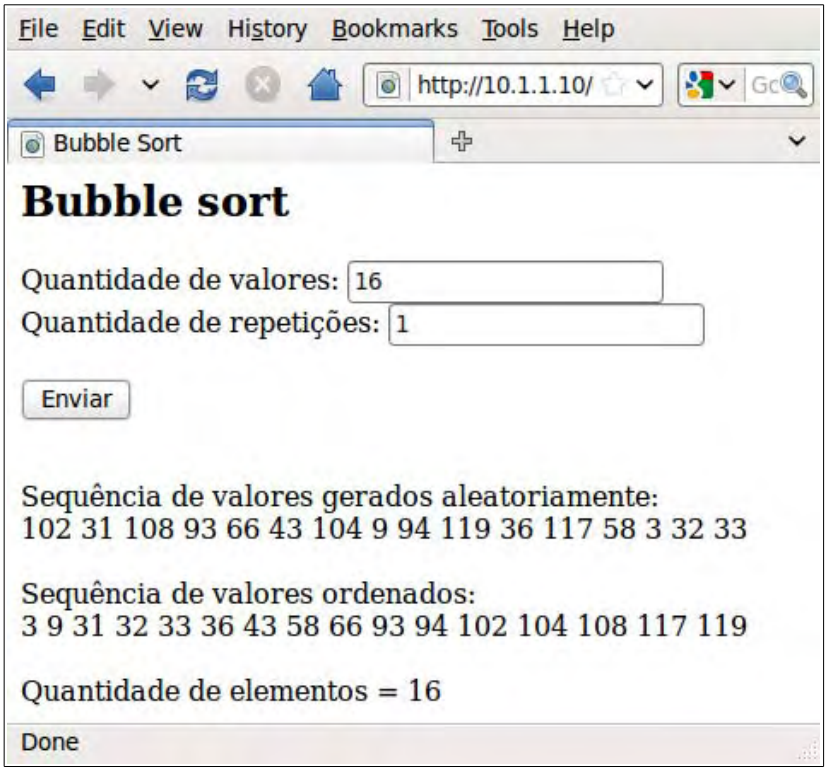


Fonte: Elaborada pelo autor.

O primeiro parâmetro apresentado é a quantidade de valores que devem ser gerados aleatoriamente pela aplicação, para que na sequência sejam ordenados pelo método *Bubble Sort*. O segundo parâmetro define a quantidade de vezes que o procedimento de gerar e ordenar valores deve ser realizado. Esta repetição foi adotada para que fosse possível obter tempos maiores de processamento com uma quantidade reduzida de valores gerados, visto que a reduzida memória SRAM do microcontrolador foi um limitante para a quantidade de valores.

Na Figura 10 apresenta-se o resultado dos valores gerados e ordenados pelo microcontrolador para os parâmetros de entrada 16 e 1, para as quantidades de valores e repetições, respectivamente.

Figura 10: Resultado de valores gerados e ordenados para entrada 16 e 1.



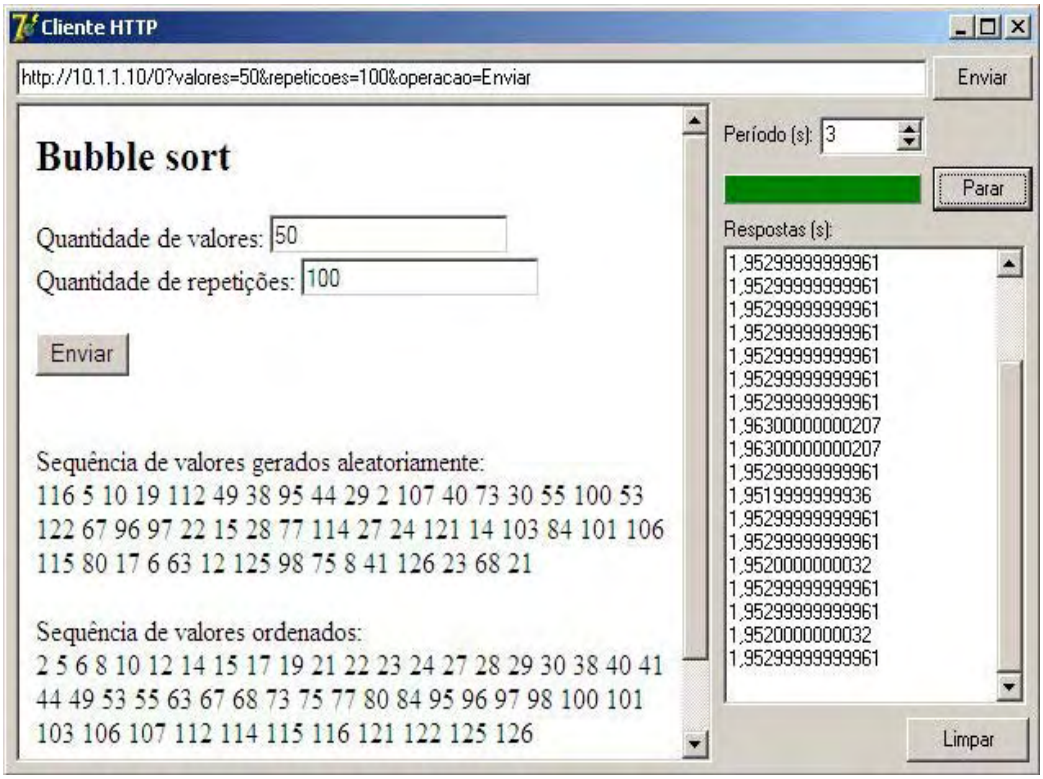
Fonte: Elaborada pelo autor.

Para automatizar o processo de requisição às aplicações embarcadas e medir os tempos de respostas, foi desenvolvida uma aplicação cliente, cuja interface é apresentada na Figura 11, que efetua requisições HTTP em quantidade e períodos previamente configurados e registra os tempos de respostas. Os dados das medições realizadas, assim como sua análise, são apresentados na seção 2.4 deste capítulo.

Inicialmente, tentou-se desenvolver esta aplicação cliente em Java, com conexões HTTP nesta tecnologia. No entanto, surgiram algumas dificuldades no tratamento das respostas HTTP, das diferentes plataformas de hardware empregadas.

Alternativamente e para melhor compreender o que estava ocorrendo, utilizou-se também o IDE Delphi com o componente WebBrowser, para o desenvolvimento da aplicação cliente. Constatou-se neste caso, que o componente WebBrowser já apresentava os tratamentos necessários para a execução dos testes.

Figura 11: Interface da aplicação Cliente HTTP desenvolvida.



Fonte: Elaborada pelo autor.

2.3.2 DB-DP11115

Para o desenvolvimento da aplicação na plataforma DB-DP-11115 foi utilizado o IDE MPLAB X da Microchip, fabricante do microcontrolador utilizado na placa, partindo do projeto PIC24 Web Server Demo(USB) V1.0 de demonstração, fornecido pela Sure Electronics, fabricante da placa. Este IDE permite ao desenvolvedor escrever sua aplicação na linguagem de programação C e compilá-la para o conjunto de instruções específicas do microcontrolador alvo. Para tanto basta especificar na ferramenta o microcontrolador alvo, que neste caso foi o PIC24FJ256GB106.

A aplicação desenvolvida foi semelhante a anterior, desenvolvida para a plataforma PME-10. Para a programação da comunicação com a interface Ethernet, também foi utilizada a pilha TCP/IP fornecida pela Microchip.

Visualmente, a interface da aplicação de ordenação desenvolvida para essa plataforma é idêntica as desenvolvidas para a PME-10, apresentadas nas Figuras 9 e 10.

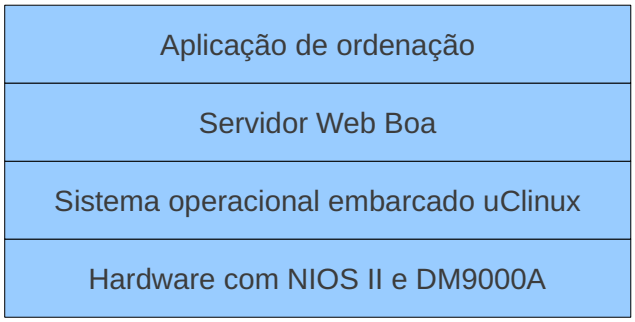
A aplicação Cliente HTTP, apresentada na seção anterior, também foi utilizada para consumir e medir os tempos de respostas da aplicação de ordenação implantada na plataforma

DB-DP11115. Os dados das medições realizadas, assim como sua análise, são apresentados na seção 2.4 deste capítulo.

2.3.3 DE2

A plataforma DE2, comparada com a PME-10 e a DB-DP11115, apresenta características, recursos e ambiente de desenvolvimento significativamente diferentes. Antes de iniciar o desenvolvimento da aplicação em software, para ser embarcada no FPGA, foi necessário desenvolver o próprio hardware, composto por um processador e um controlador Ethernet. Além disso, deve-se embarcar o sistema operacional uClinux neste hardware e o servidor Web Boa[11] sobre o sistema operacional uClinux, conforme as camadas apresentadas na Figura 12, para somente então ter-se o ambiente necessário para a execução da aplicação de ordenação desenvolvida também na linguagem C.

Figura 12: Camadas da plataforma embarcada na DE2.



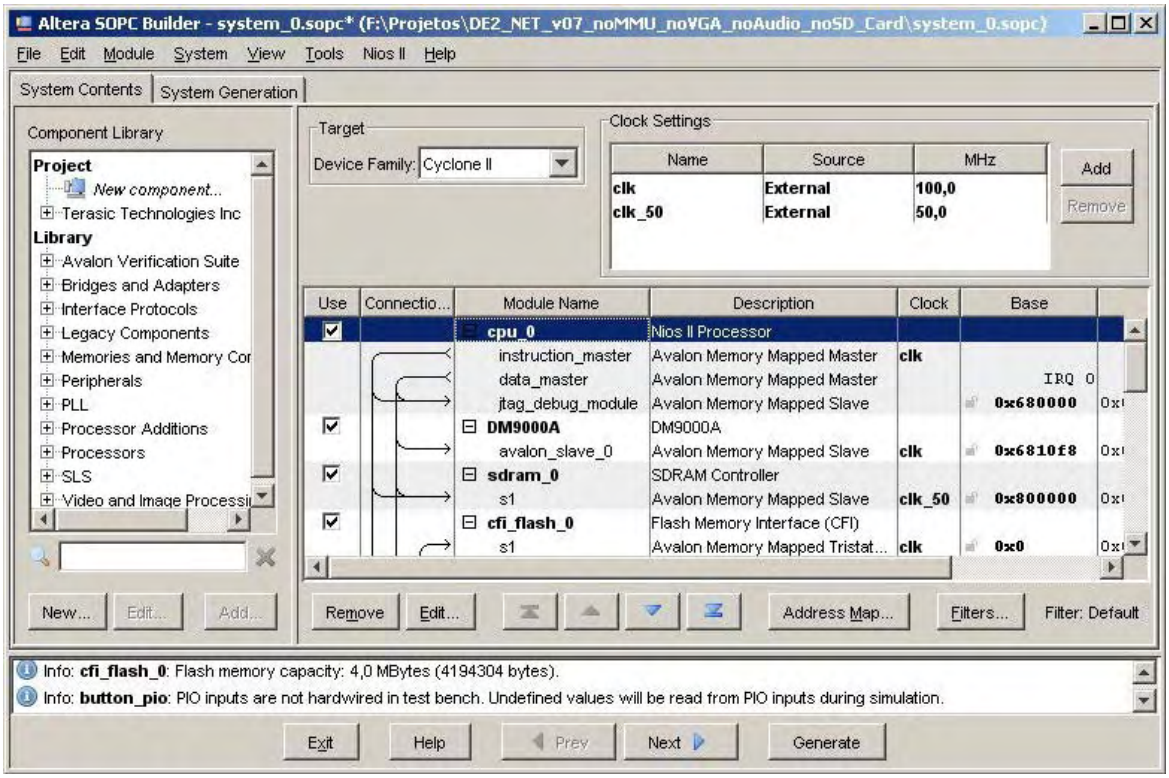
Fonte: Elaborada pelo autor.

A arquitetura do hardware embarcado no FPGA da DE2 foi desenvolvida com a ferramenta proprietária SOPC Builder 9.1 da Altera, integrada ao IDE Quartus II 9.1 também da Altera, que permite a composição de uma arquitetura computacional em torno do *core* do processador NIOS II. Além de blocos essenciais como controladores de memórias e barramentos, foi incluído nesta arquitetura o módulo do controlador Ethernet DM9000A, fornecido pela Terasic[12].

Na Figura 13 apresenta-se a interface da ferramenta SOPC Builder, com os módulos *cpu_0* (processador NIOS II), *DM9000A* (controlador Ethernet), *sdram_0* (controlador da SDRAM) etc., selecionados para um dispositivo da família Cyclone II, existente na DE2.

11 Pequeno servidor Web de código aberto.
12 Empresa de desenvolvimento de ferramentas de síntese, placas e soluções usando FPGAs da Altera.

Figura 13: Interface do SOPC Builder com alguns dos módulos selecionados.



Fonte: Elaborada pelo autor.

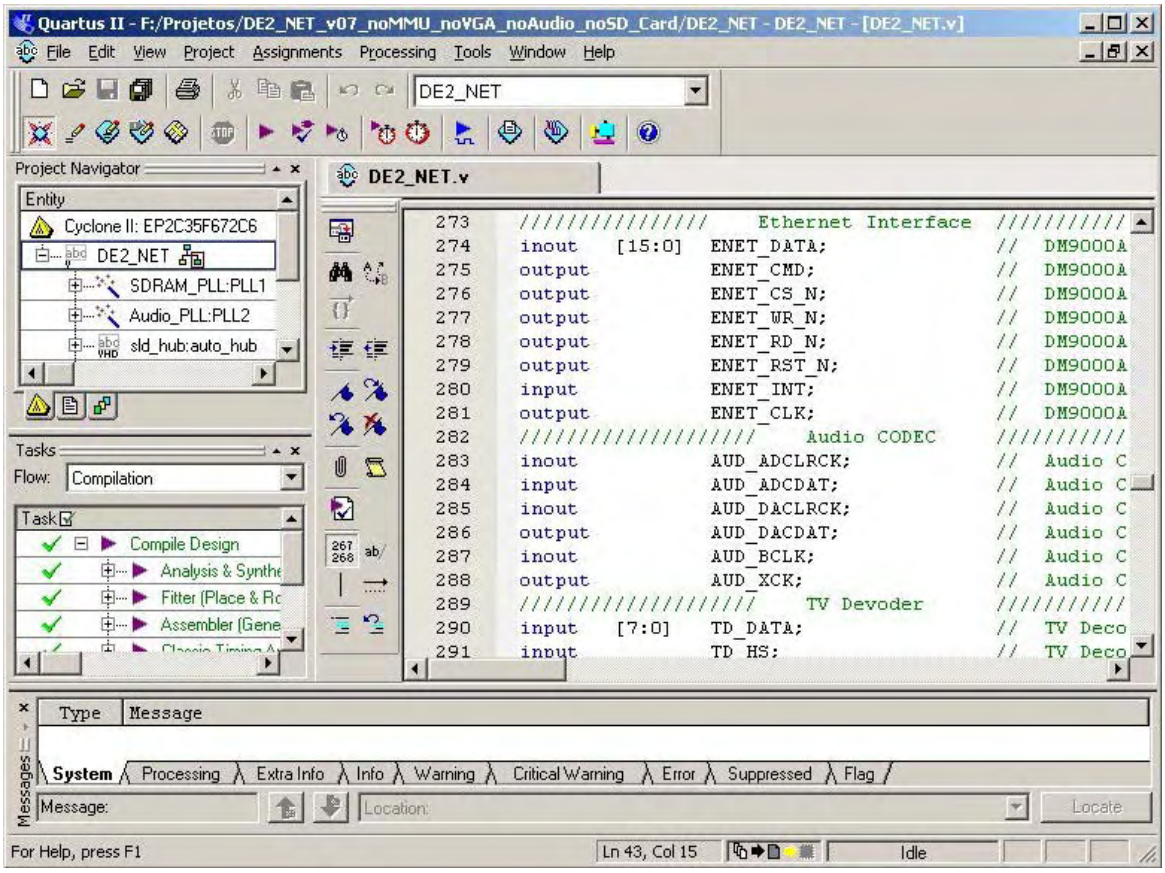
Para a elaboração deste projeto, partiu-se do projeto DE2_NET, que é um dos projetos de demonstração da placa DE2. As alterações realizadas foram apenas para remover módulos que não foram necessários nesta pesquisa, como os módulos do controlador de áudio e vídeo.

Depois de gerada a arquitetura computacional pelo SOPC Builder, foi necessário remover as referências aos módulos não utilizados no código Verilog no Quartus II, apresentado na Figura 14, para somente então compilar e sintetizar o hardware que foi descarregado na FPGA Cyclone II da DE2, concluindo assim a primeira camada desta plataforma.

Enquanto a primeira camada (hardware) foi desenvolvida com as ferramentas proprietárias relatadas anteriormente, as demais camadas (software) foram desenvolvidas com ferramentas livres e de código aberto. Para tanto, foi necessário configurar um cenário de desenvolvimento cruzado específico para a plataforma alvo.

Desenvolvimento cruzado refere-se ao emprego de ferramentas de desenvolvimento sobre sistemas diferentes dos que serão descarregadas e executadas as aplicações. Geralmente as plataformas do sistema alvo apresentam restrita capacidade computacional.

Figura 14: Quartus II com algumas declarações de I/O.



Fonte: Elaborada pelo autor.

Para configurar o cenário de desenvolvimento cruzado foi adotada a distribuição Linux CentOS 5.4, virtualizada sobre um sistema operacional hospedeiro. A ferramenta de virtualização utilizada foi o VirtualBox 3.1.8 e o sistema operacional hospedeiro o Ubuntu 9.10.

Escolheu-se a distribuição CentOS pois trata-se de uma recompilação livre da distribuição Linux comercial RedHat que apresenta mais referências e diversidade de ferramentas, para o ambiente de desenvolvimento do sistema operacional embarcado uClinux. Uma outra opção seria utilizar a distribuição Linux Fedora, que também é livre, no entanto, apresenta recursos mais novos e menos estáveis do que os encontrados nas distribuições RedHat e CentOS.

A virtualização do sistema operacional adotado foi necessária, pois o cenário de desenvolvimento para a plataforma alvo, exige configurações de ambiente muito específicas, o que comprometeria o funcionamento das demais aplicações do sistema operacional hospedeiro.

Uma alternativa à virtualização é reservar uma máquina exclusivamente para o cenário de desenvolvimento, no entanto, preferiu-se desenvolver em um sistema operacional virtualizado, preservando assim o sistema hospedeiro e economizando no uso de uma máquina a mais.

Instalado o CentOS no disco virtual criado pelo VirtualBox, foi necessário instalar os complementos contidos na imagem *VboxGuestAdditions_3.1.6.iso* para ter acesso o sistema de arquivos do sistema operacional hospedeiro. Para tanto, foram executado os comandos a seguir:

```
1: yum install gcc kernel-devel
2: export KERN_DIR=/usr/src/kernels/2.6.18-194.8.1.el5-i686
3: ./VBoxLinuxAdditions-x86.run
4: mount -t vboxsf <compartilhamento> <montagem>
```

A primeira linha utiliza o aplicativo *yum* para instalar o compilador *gcc* e os fontes do *kernel* do CentOS. A segunda linha de comando declara temporariamente a variável de ambiente *KERN_DIR* e na terceira linha o arquivo *VboxLinuxAdditions-x86.run* é executado. Antes disso ele precisa ser copiado do local de montagem da imagem *VboxGuestAdditions_3. 1.6.iso* para uma pasta do sistema de arquivos local e, por fim, o nome do compartilhamento criado no VirtualBox com uma pasta do sistema hospedeiro pode ser montado em uma pasta no sistema operacional virtualizado. Este compartilhamento facilita a troca de arquivos entre os dois sistemas operacionais, visto que o sistemas de arquivos de ambos estão no mesmo disco. No entanto, não é necessário caso utilize-se de outro meio de transferência de arquivos entre os sistemas.

Para instalar no CentOS o compilador GCC 3.4.6 para o NIOS II (nios2-linux-gcc) foi utilizado o pacote *nios2gcc-20080203.tar.bz2*, disponível em *http://www.niosftp.com/pub/gnutools/nios2gcc-20080203.tar.bz2*, com as linhas de comando a seguir:

```
1: tar jxf nios2gcc-20080203.tar.bz2
2: export PATH=$PATH:/opt/nios2/bin/
```

O aplicativo *tar* descomprime o pacote *nios2gcc-20080203.tar.bz2*. Sugere-se descomprimir este arquivo na raiz do sistema de arquivos, pois ele gera a pasta /opt/nios/bin

onde estão os executáveis do compilador GCC para o NIOS II. A segunda linha de comando sugere-se incluí-la em algum *script* de inicialização, como o *.bash_profile*, para que esta alteração no PATH do sistema seja permanente.

Antes de instalar os arquivos da distribuição do uClinux no CentOS foi necessário adicionar as referências ao repositório do EPEL (*Extra Packages for Enterprise Linux*). O primeiro comando descrito a seguir faz o download do pacote do *epel* e o segundo comando utiliza o aplicativo *rpm* para instalá-lo. Na última linha de comando é utilizado o aplicativo *yum* para instalar os pacotes *git-all*, localizado no repositório EPEL, *ncurses-devel* e *byacc*, assim como as respectivas dependências resolvidas pelo *yum*.

```
1: wget http://download.fedora.redhat.com/pub/epel/5/i386/epel-
release-5-3.noarch.rpm
2: rpm -Uvh epel-release-5-3.noarch.rpm
3: yum install git-all ncurses-devel byacc
```

Caso não se tenha instalado o pacote *gcc*, como nos procedimentos descritos anteriormente para a instalação do *VboxLinuxAdditions-x86.run*, é necessário instalá-lo para compilar a distribuição uClinux, com os módulos que serão escolhidos conforme descrição a seguir, salientando que dependendo dos módulos selecionados para o uClinux, pode ser necessário a instalação de outros pacotes no CentOS.

Para instalar a distribuição uClinux 4.0 no CentOS, foi utilizado o pacote *nios2-linux-20090929.tar,* disponível em *http://www.niosftp.com/pub/linux/nios2-linux-20090929.tar*, utilizando os comandos a seguir:

```
1: tar xf nios2-linux-20090929.tar
2: cd nios2-linux
3: ./checkout
```

Na primeira linha de comando o aplicativo *tar* descomprime o arquivo *nios2-linux-20090929.tar* em uma pasta denominada *nios2-linux*, dentro de onde for executado. Sugere-se utilizar também a pasta /*opt* onde foi instalado o compilador *nios2-linux-gcc*. A segunda linha de comando apenas acessa a pasta *nios2-linux* e a última linha de comando executa o *script checkout*, que utiliza o aplicativo *git* do pacote *git-all*, para conferir e distribuir o sistema de

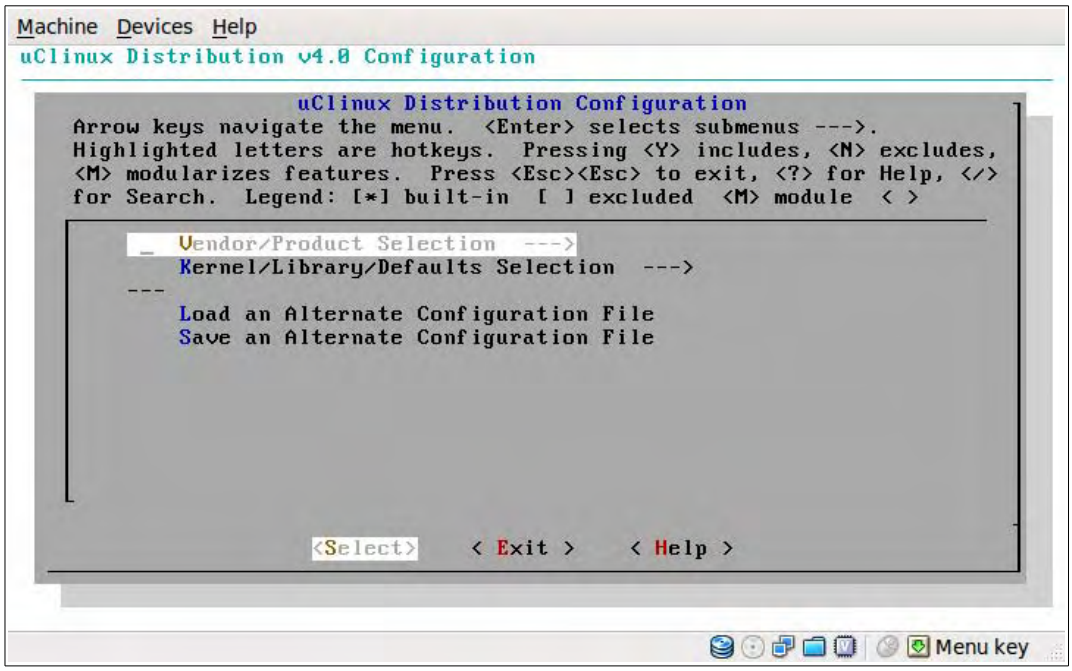
arquivos da distribuição uClinux na pasta *nios2-linux*.

Para selecionar os módulos e compilar a distribuição uClinux foram utilizados os comandos a seguir:

```
1: cd uClinix-dist
2: make menuconfig
3: make vendor_hwselect SYSPTF=</caminho_arquivo/arquivo.ptf>
4: make
```

A primeira linha de comando apenas acessa a pasta */opt/nios2-linux/uClinux-dist*, considerando que estava-se na pasta */opt/nios2-linux*. O comando da segunda linha abre uma interface gráfica para configuração dos módulos da distribuição uClinux, apresentada na Figura 15.

Figura 15: Interface de configuração de módulos da distribuição uClinux.

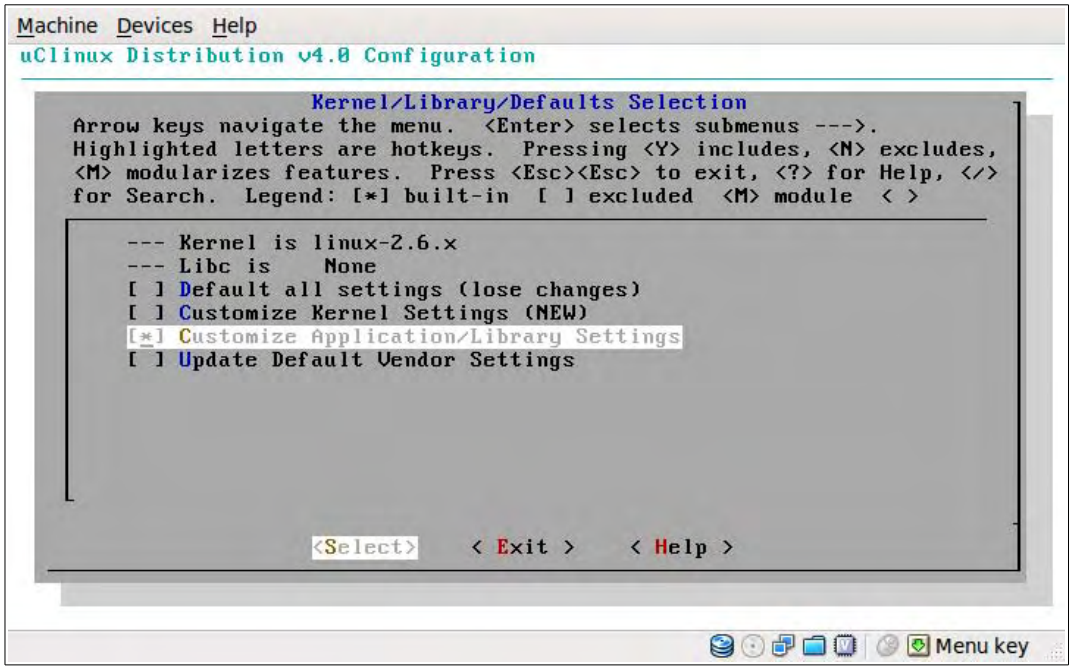


Fonte: Elaborada pelo autor.

Esta distribuição vem por padrão com a Altera e NIOS II selecionados como fabricante e produto, respectivamente, entretanto podem ser alterados no primeiro item da interface. No segundo item da interface tem-se a opção de personalizar outras configurações da distribuição, como pode ser observado na interface apresentada na Figura 16. Embora esta

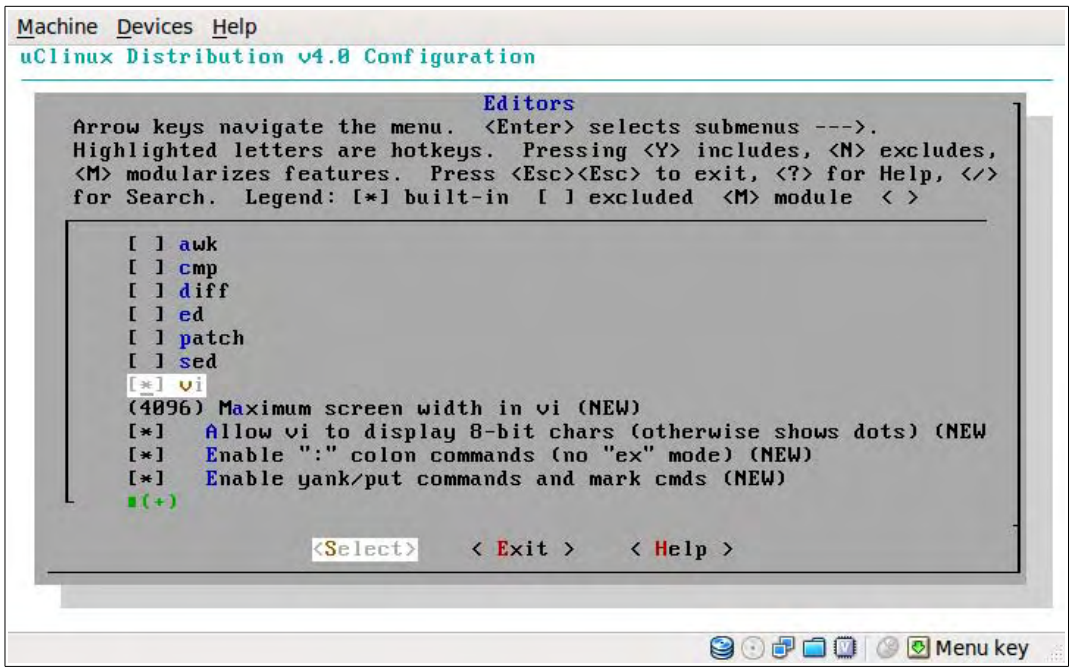
distribuição já apresente por padrão a seleção de módulos de suporte a rede TCP/IP e o servidor Web Boa, selecionou-se a opção *Customize Application* apenas para incluir o editor *vi*, como pode ser observado na interface apresentada na Figura 17.

Figura 16: Demais opções de personalização da distribuição.



Fonte: Elaborada pelo autor.

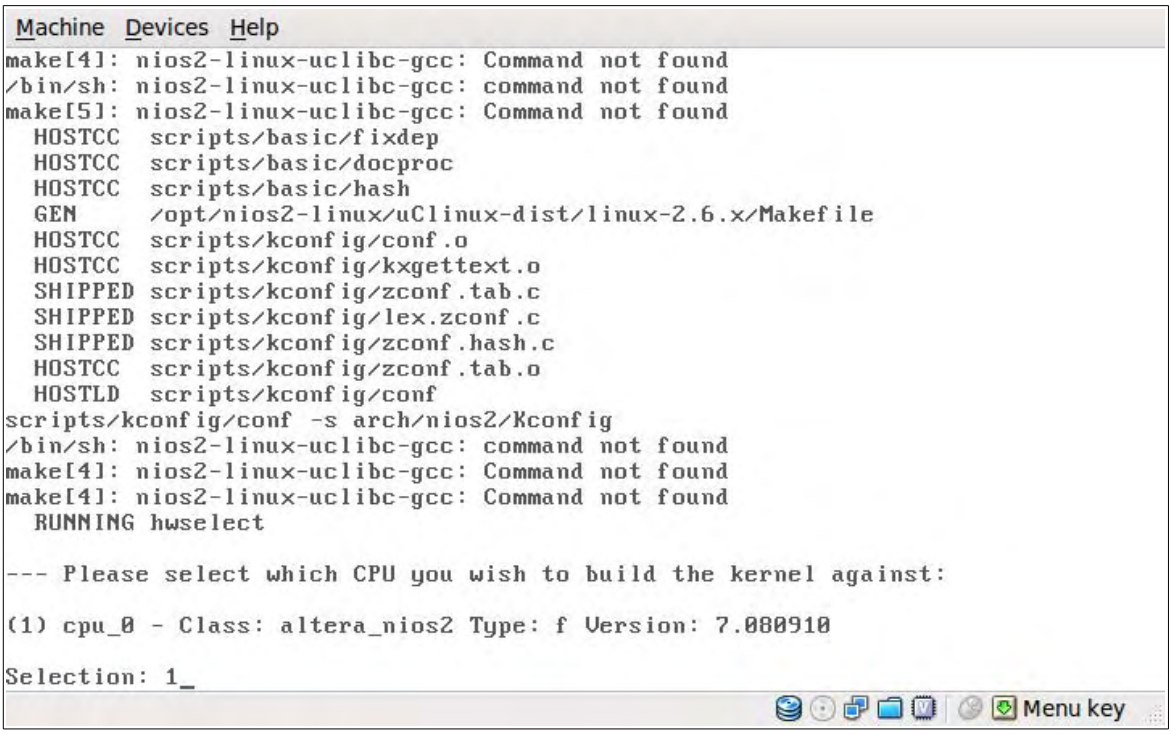
Figura 17: Escolha do editor *vi*.



Fonte: Elaborada pelo autor.

A terceira linha de código configura detalhes do hardware, especificando o caminho do arquivo *.ptf* gerado pela síntese do hardware pelo Quartus II. Na sequência da execução deve ser informado a CPU que o sistema deve utilizar. Como pode ser observado na Figura 18, a única opção é a *cpu_0* que foi o único processador (NIOS II) selecionado no SOPC Builder. Deve-se ainda selecionar a memória do hardware em que o *kernel* da distribuição deve ser executado. Como pode ser observado na Figura 19, a opção escolhida foi a memória SDRAM, módulo *sdram_0* selecionado no SOPC Builder.

Figura 18: Escolha da CPU.

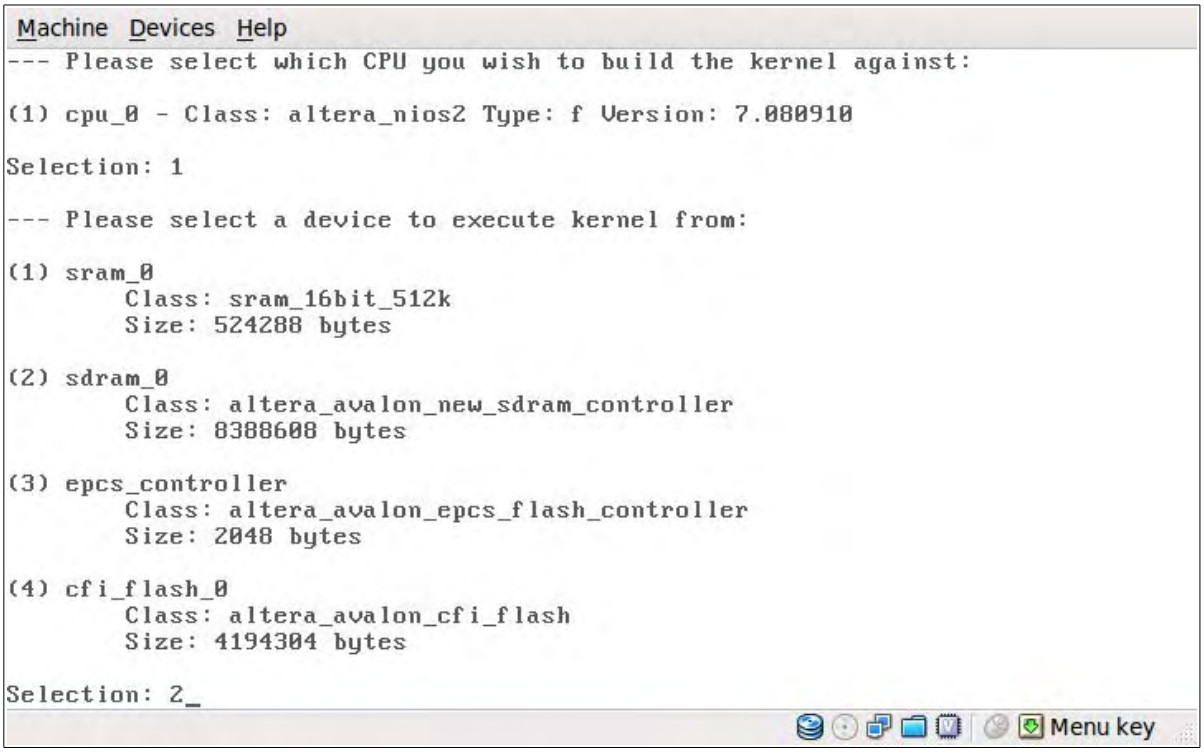


Fonte: Elaborada pelo autor.

Por fim, o comando *make* da última linha compila a distribuição uClinux, gerando na pasta *../uClinux-dist/images/* o arquivo de imagem *zImage* da distribuição deste sistema operacional.

Os procedimentos realizados até este ponto permitiram a experiência de compor e compilar uma distribuição Linux, no entanto, para a continuidade dos trabalhos foi necessário carregar a imagem da distribuição na plataforma DE2.

Figura 19: Escolha da memória de execução.



Fonte: Elaborada pelo autor.

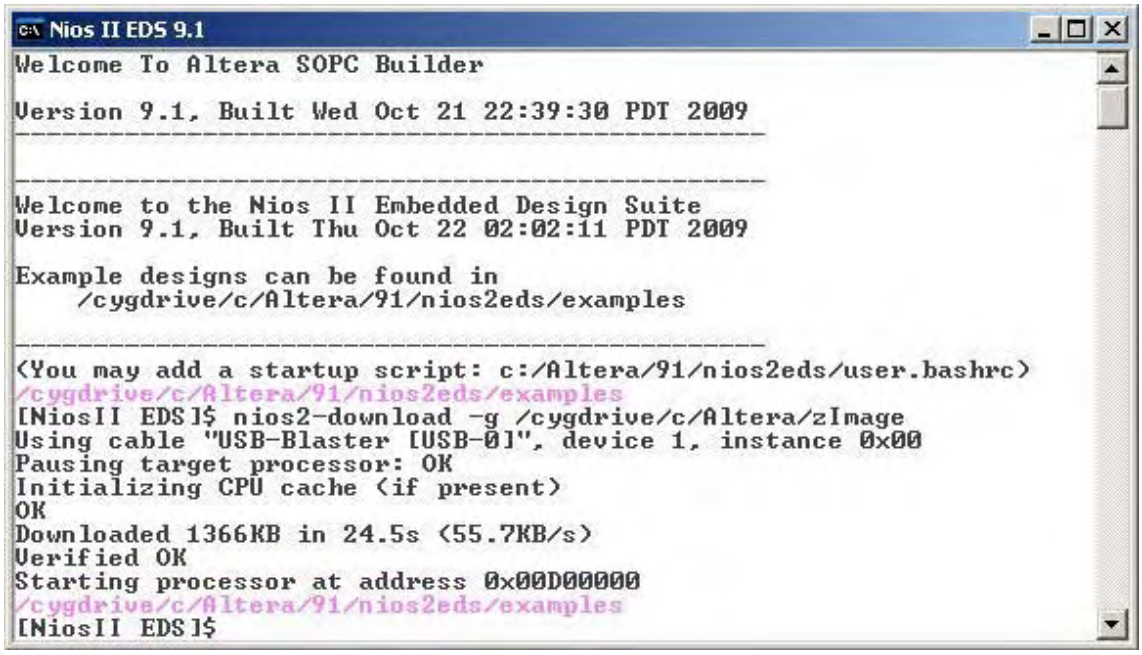
Para carregar a imagem *zImage* no hardware desenvolvido e gravado no FPGA, foi utilizado o console da ferramenta NIOS II EDS (*Embedded Design Suite*) 9.1 da Altera, apresentado na Figura 20, com os comandos a seguir:

```
1: nios2-download -g </caminho_arquivo/zImage>
2: nios2-terminal
```

O NIOS II EDS é um pacote de ferramentas gratuitas da Altera, no entanto, necessita do IDE Quartus II para funcionar. As distribuições utilizadas do IDE Quartus II, SOPC Builder e NIOS II EDS foram sobre sistema operacional Windows.

O NIOS II EDS carrega a imagem do sistema operacional uClinux na memória SDRAM do hardware pela interface USB Blaster da placa DE2. O parâmetro *-g* faz com que o processador entre em execução imediatamente após o carregamento da imagem e o comando *nios2-terminal* abre um terminal em linha de comando pelo NIOS II EDS via a interface USB Blaster. Tem-se, então, acesso ao sistema operacional em execução no FPGA, como apresentado na Figura 21.

Figura 20: Carregamento da *zImage* pelo console do NIOS II EDS.



Fonte: Elaborada pelo autor.

Figura 21: Terminal em linha de comando com o uClinux.

Fonte: Elaborada pelo autor.

Pelo terminal do NIOS II EDS as configurações de rede e inicialização de serviços do uClinux pedem ser feitas com os comandos a seguir:

```
1: ifconfig eth0 hw ether <endereço_MAC>
2: ifconfig eth0 <endereço_IP> netmask <máscara_de_rede>
```

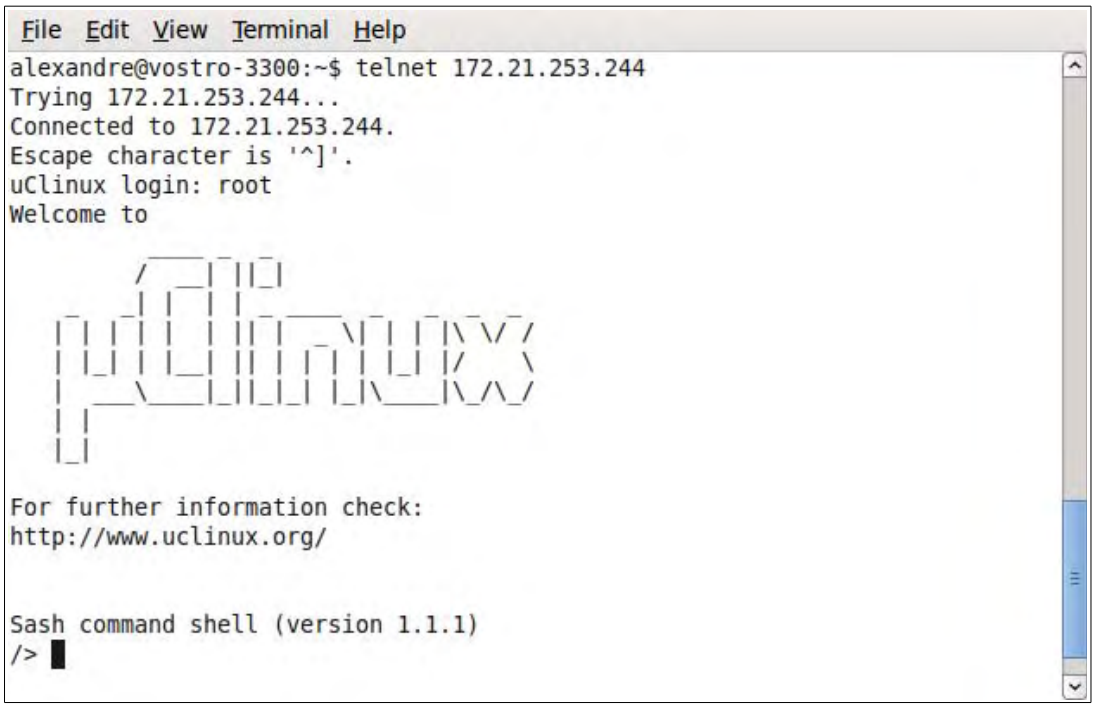
```
3: inetd &
4: boa -d &
```

A primeira linha de comando define o endereço MAC da interface Ethernet. A segunda linha de comando define o endereço e a máscara do protocolo IP e a terceira linha inicia em *background* o *inetd*, que é um gerenciador de serviços que inicia o servidor FTP *(ftpd)* e o servidor Telnet *(telnetd)* e por fim a última linha de comando inicia em *background* o servidor Web Boa.

Embora inicialmente estas configurações e inicializações tenham sido feitas como descrito, posteriormente foram inseridas no *script* de inicialização */opt/nios2-linux/uClinux-dist/vendors/Altera/nios2/rc* da distribuição uClinux, que após compilada e carregada a distribuição, será o arquivo */etc/rc* do sistema em operação.

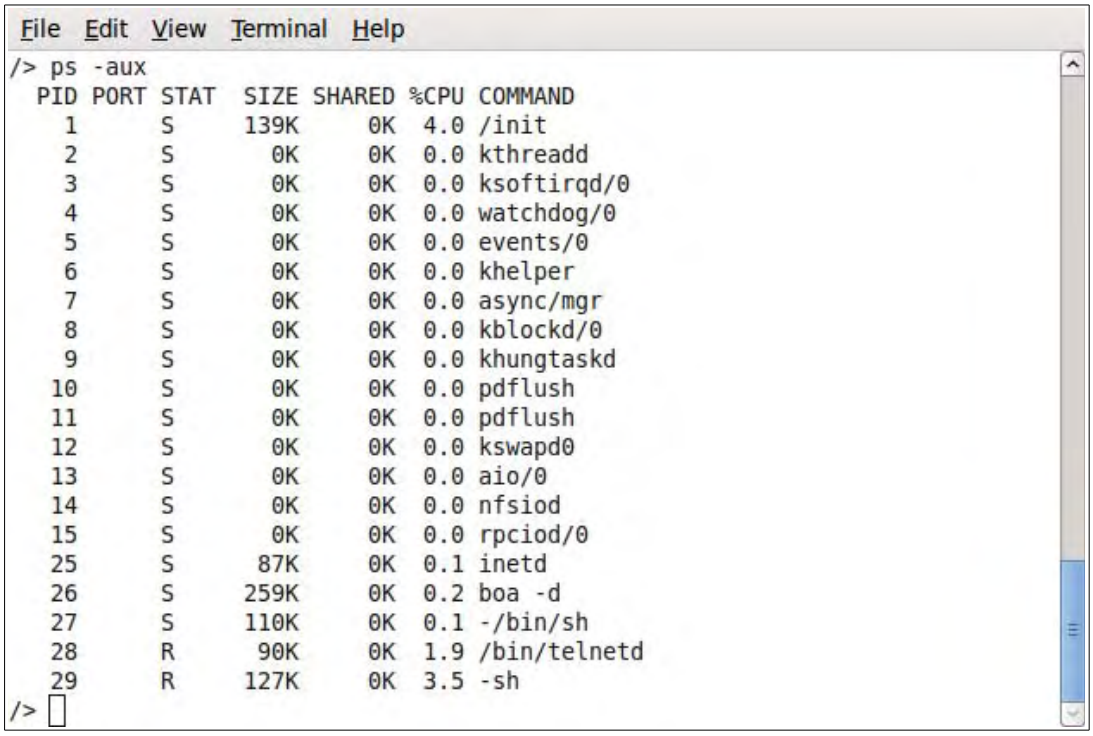
Nesta distribuição do uClinux, por padrão, o usuário *root* não possui senha. Para acessar um terminal via Telnet é possível conectar-se com o usuário *root* deixando a senha sem preencher. Na Figura 22 apresenta-se um terminal acessado via Telnet e na Figura 23 pode ser visto a lista de processos em execução no uClinux, onde pode ser observado os processos 26 e 28 do servidor Web Boa e do servidor Telnet, respectivamente.

Figura 22: Acesso ao uClinux via terminal Telnet.



Fonte: Elaborada pelo autor.

Figura 23: Lista de processos em execução no uClinux.



Fonte: Elaborada pelo autor.

Até esta etapa de desenvolvimento, tem-se em operação as três primeiras camadas apresentadas na Figura 12, restando apenas a aplicação de ordenação para ser hospedada no servidor Web Boa. Esta aplicação também foi uma página CGI desenvolvida em C e compilada pelo *nios2-linux-gcc* instalado no CentOS, com a linha de comando a seguir:

```
nios2-linux-gcc arquivo_fonte.c -o arquivo_cgi -elf2flt
```

Uma das diferenças significativas desta aplicação, comparada com a desenvolvida para a primeira plataforma, foi a detecção dos dados da requisição GET. Utilizando a biblioteca *cgi_header.c* é possível recuperar os valores dos parâmetros recebidos, pelos nomes dos parâmetros enviados na requisição e a resposta pode ser realizada diretamente pelo método de saída de dados *printf()*, simplificando significativamente a entrada e saída de dados da aplicação.

Adicionalmente, percebeu-se menor restrição de tipos de dados e uso da memória, que se deu pela maior quantidade na SDRAM nesta arquitetura. Além disso, o arquivo executável da aplicação, uma vez compilado no CentOS, pode ser transferido por FTP (*File Transfer*

Protocol) para a pasta de hospedagem do servidor Boa, */home/httpd*, sem a necessidade de conversão de sistemas de arquivos, como foi feito com o MPFS na implementação da primeira plataforma.

Os arquivos CGI compilados devem ser hospedados na pasta */home/httpd/cgi-bin* e concedido permissão de execução, o que pode ser feito pelo linha de comando:

```
chmod 744 arquivo.cgi
```

Visualmente, a interface da aplicação de ordenação desenvolvida para essa plataforma é idêntica as desenvolvidas para a PME-10, apresentadas nas Figuras 9 e 10.

A aplicação Cliente HTTP, apresentada na seção 2.3.1, também foi utilizada para consumir e medir os tempos de respostas da aplicação de ordenação implantada na plataforma DE2. Os dados das medições realizadas, assim como sua análise, são apresentados na seção 2.4 deste capítulo.

2.3.4 NGW100

A plataforma NGW100 apresenta arquitetura AVR32 já definida pelo microcontrolador AT32AP7000 e é fornecida pelo fabricante com uma distribuição Linux previamente instalada na memória Flash, com servidores Telnet, FTP, SSH (*Secure SHell*), Web etc., que são carregados na inicialização. O servidor Web também é o Boa e o *kernel* da distribuição Linux é o 2.6.18, o mesmo da distribuição uClinux utilizada na segunda plataforma.

Comparando esta plataforma com as camadas apresentadas na Figura 12, pode se notar que o hardware é fixo e as camadas intermediárias já se encontram implantadas, embora seja possível desenvolve-las assim como feito para a plataforma anterior. No entanto, com o objetivo de testar a capacidade de processamento desta plataforma, para compará-la com as demais, partiu-se para a quarta camada, ou seja, o desenvolvimento da aplicação de ordenação compilada para a arquitetura AVR32.

Para instalar e configurar o cenário de desenvolvimento cruzado da plataforma NGW100, optou-se em utilizar como sistema operacional virtualizado a distribuição Linux Ubuntu 9.04, pois o fabricante da plataforma apresenta ferramentas para esta distribuição, além de ser a preferência do autor.

Após instalar o Ubuntu 9.04 em outro disco de imagem do VirtualBox foi necessário instalar os complementos contidos na imagem *VboxGuestAdditions_3.1.6 .iso*, como feito no CentOS, exceto que no Ubuntu já vem instalado os fontes do *kernel*, não sendo necessária a instalação do pacote *kernel-devel* nem a criação da variável de ambiente *KERN_DIR*.

O pacote de ferramentas utilizado foi o *avr32_gnu_toolchain_2.4.2_ubuntu_904.zip*, disponível em *http://www.atmel.com/dyn/resources/prod_documents/dl_avr32_gnu_toolchain_2.4.2_ubuntu_904.zip*. Para descomprimi-lo e instalá-lo foram utilizadas as linhas de comando a seguir:

```
1: sudo unzip avr32_gnu_toolchain_2.4.2_ubuntu_904.zip
2: sudo dpkg -i *.deb
3: sudo apt-get -f install
4: sudo dpkg -i *.deb
```

A primeira linha de comando descomprime os arquivos na pasta em que estiver. A segunda linha de comando tenta instalar os pacotes *.deb* descomprimidos, mas é abortada apresentando a lista de dependências requeridas. A terceira linha instala as dependências listadas anteriormente e, por fim, a última linha efetiva a instalação dos pacotes *.deb* abortados anteriormente.

Além do AVR32 GNU Toolchain 2.4.2 instalado, o que já permite compilar aplicações *standalone* para o microcontrolador AVR32, para compilar executáveis para o *kernel* Linux embarcado sobre o microcontrolador AVR32 foi necessário instalar também o pacote de ferramentas *buildroot-avr32-v2.3.0.tar.bz2*, disponível em *http://www.atmel.no/buildroot/source/buildroot-avr32-v2.3.0.tar.bz2*. Para descomprimi-lo e instalá-lo foram utilizadas as linhas de comando a seguir:

```
1: sudo tar -jvxf buildroot-avr32-v2.3.0.tar.bz2
2: cd buildroot-avr32-v2.3.0
3: sudo make atngw100_defconfig
4: sudo make
5: export PATH=$PATH:/opt/buildroot-avr32-v2.3.0/build_avr32/st
aging_dir/bin/
```

A primeira linha de comando descomprime o pacote na pasta *buildroot-avr32-v2.3.0*. Sugere-se colocá-la na pasta */opt*. A segunda linha de comando acessa a pasta descomprimida, a terceira define a plataforma alvo e a linha seguinte compila as ferramentas do pacote para o ambiente existente. Por fim, a última linha adiciona, ao *path* do sistema, os binários deste *Buildroot*. Sugere-se deixar essa última linha de comando em algum *script* de inicialização, como o */etc/enviroment*, para que estes binários estejam sempre disponíveis no ambiente. Na implementação, utilizou-se o *avr32-linux-gcc* para compilar a aplicação de ordenação desenvolvida para essa plataforma, com a linha de comando a seguir:

```
avr32-linux-gcc arquivo.c -o arquivo.cgi
```

Uma das diferenças desta aplicação, comparada com as desenvolvidas anteriormente, foi a detecção dos dados da requisição GET, que puderam ser feitas diretamente pelos parâmetros passados ao método *int main(int argc, char *argv[])*, sendo *argc* a quantidade de parâmetros passados pela requisição GET e **argv[]* um ponteiro para um vetor com os parâmetros recebidos. Visualmente, a interface da aplicação de ordenação desenvolvida para essa plataforma também é idêntica as desenvolvidas para a PME-10, apresentada nas Figuras 9 e 10.

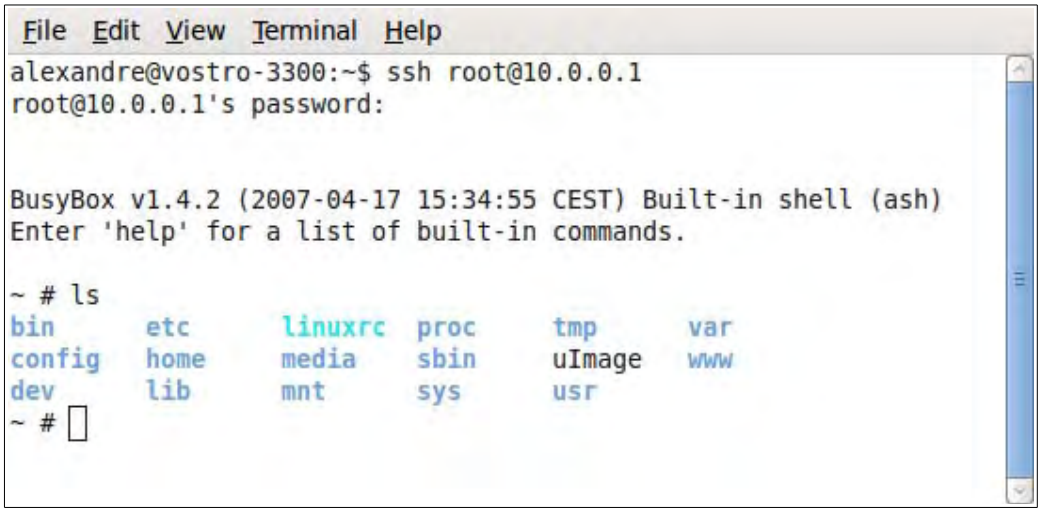
Assim como nas aplicações com as duas primeiras plataformas, a aplicação Cliente HTTP também foi utilizada para consumir e medir os tempos de respostas desta aplicação de ordenação. Os dados dessas medições, assim como sua análise, são apresentados na seção 2.4 deste capítulo.

Nesta plataforma a pasta de hospedagem do servidor Web Boa está em */www* e os CGIs em */www/cgi-bin*, a senha padrão do usuário *root* é *roota*, que pode ser usada para acessar um terminal SSH, como mostrado na Figura 24, onde o comando *ls* é usado para listar as pastas e arquivos da raiz do sistema operacional.

Na Figura 25 apresenta-se a listagem dos processos em execução no sistema operacional embarcado na NGW100. Observa-se que o servidor Web Boa (*httpd*) está em execução sob o PID[13] (*Process ID)* 317, enquanto o servidor Telnet (*telnetd*) executa com o PID 237.

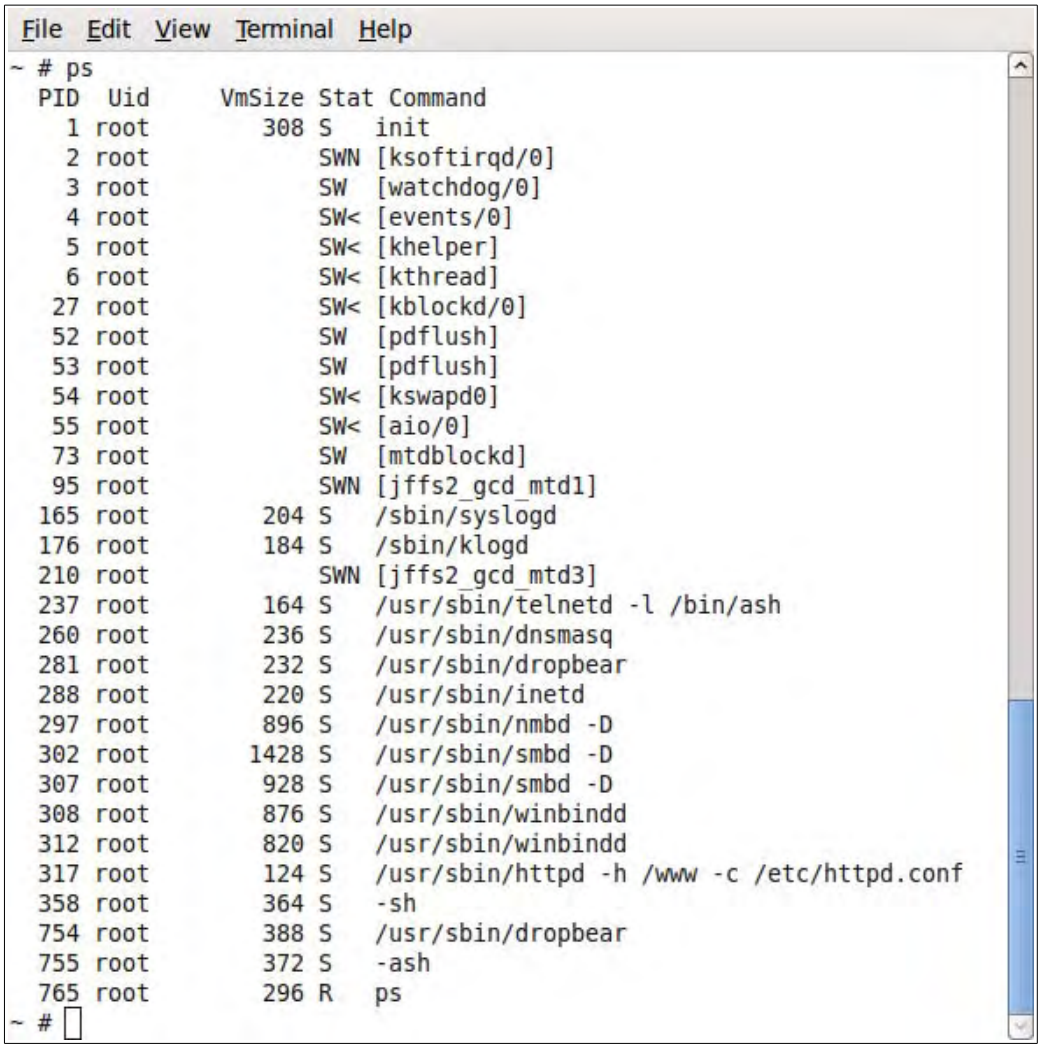
13 Número que identifica o processo em execução no sistema operacional.

Figura 24: Acesso à NGW100 via terminal SSH.



Fonte: Elaborada pelo autor.

Figura 25: Lista de processos em execução no sistema operacional embarcado.



Fonte: Elaborada pelo autor.

2.4 RESULTADOS

Como citado anteriormente, a aplicação de ordenação empregando o algoritmo *Bubble Sort* desenvolvida para cada uma das plataformas descritas, foi testada quanto ao tempo de resposta para a ordenação em duas situações, a primeira considerando constante a quantidade de elementos (100). Dado a restrição de memória da primeira plataforma optou-se por variar a quantidade de repetições. Já na segunda situação considera-se apenas uma única operação de ordenação e varia-se a quantidade de elementos, o que inviabilizou a utilização da plataforma PME-10 por exceder significativamente aos cem elementos.

Nas Tabelas 6 e 7 são apresentadas as médias das medições para as duas situações, respectivamente, sendo que a aplicação cliente realizou dez medições para cada variação de repetições e elementos.

Tabela 6: Tempo de resposta em segundos para ordenação de cem elementos.

Nº repetições \ Plataforma	**PME-10**	**DB-DP11115**	**DE2**	**NGW100**	**PC**
1	0.22	0,21	0.2	0.04	0.01
10	0.85	0,30	0.23	0.04	0.01
50	3.61	0,68	0.37	0.1	0.01
100	7.08	1,15	0.54	0.16	0.01
150	10.54	1,63	0.71	0.23	0.02
250	-	2,57	1.06	0.36	0.02
500	-	4,94	1.91	0.69	0.02
750	-	7,31	2.77	1.02	0.03
1000	-	9,67	3.63	1.35	0.03
1500	-	14,41	5.37	2.02	0.04

Fonte: Elaborada pelo autor.

Adicionalmente, como também pode ser observado nas Tabelas 6 e 7, foram feitos os mesmos testes em uma arquitetura PC *multicore*, não embarcada, com as características apresentadas na Tabela 8.

Na máquina PC também foi instalado o mesmo servidor Web Boa utilizado nas plataformas embarcadas com sistema operacional, neste caso, sobre o sistema operacional Ubuntu 10.04. Assim, a mesma aplicação CGI utilizada na plataforma NGW100 pode ser reutilizada na platarma PC, para tanto, foi necessário recompilá-la para o sistema operacional em questão.

Tabela 7: Tempo de resposta em segundos para uma única operação de ordenação.

Nº elementos \ Plataforma	DB-DP11115	DE2	NGW100	PC
100	0,21	0.2	0.03	0.01
250	0,30	0.35	0.05	0.01
500	0,57	0.64	0.09	0.01
750	0,99	1.05	0.17	0.02
1000	1,57	1.63	0.26	0.02
1250	2,31	2.36	0.38	0.03
1500	3,2	3.2	0.53	0.03
1750	4,36	4.17	0.69	0.03
2000	5,61	5.22	0.89	0.03
2500	-	-	1.35	0.04
3000	-	-	1.92	0.06
4000	-	-	3.38	0.09
5000	-	-	5.3	0.14
6000	-	-	7.69	0.19
8000	-	-	-	0.33
10000	-	-	-	0.51
15000	-	-	-	1.13
20000	-	-	-	1.98
25000	-	-	-	3.12
30000	-	-	-	4.46

Fonte: Elaborada pelo autor.

Tabela 8: Características do PC utilizado.

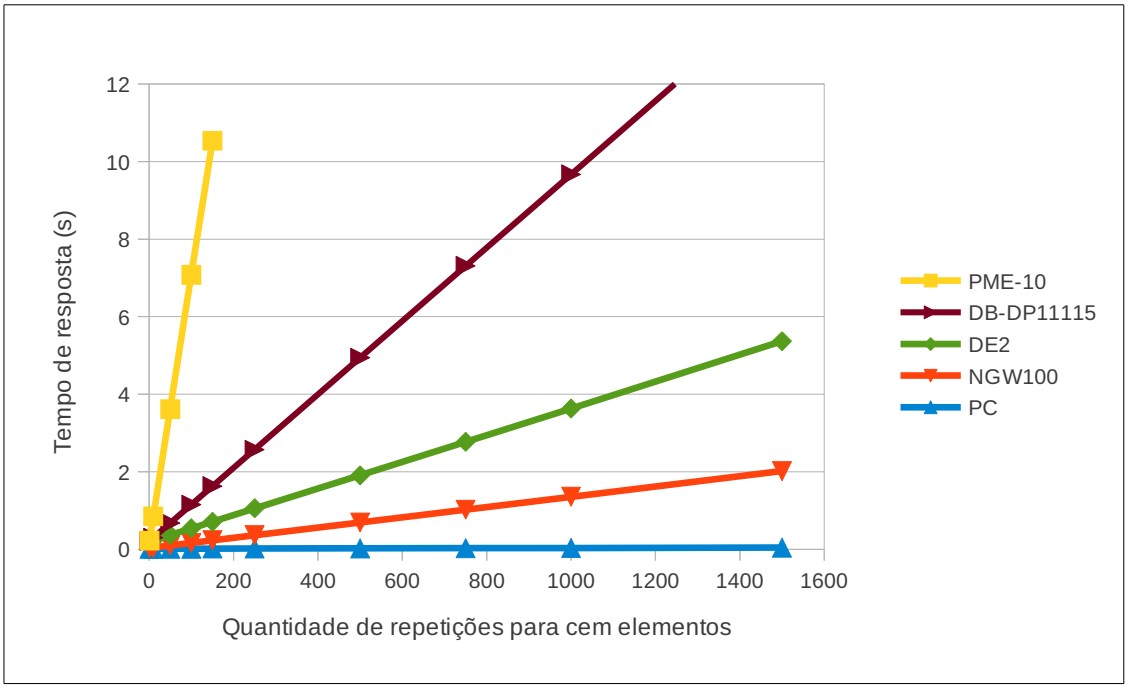
Característica	Descrição
Processador	Core i5 430m da Intel
Clock	2.26 GHz
SRAM	3 MB
SDRAM	3 GB
Ethernet	1 Gbps
Modelo	Vostro-3300
Fabricante	Dell

Fonte: Elaborada pelo autor.

Na Figura 26 apresenta-se o gráfico produzido com os dados da Tabela 6. Como pode

ser observado o tempo de resposta de cada uma das plataformas é linear em função das repetições, o que era de se esperar visto que o tempo de ordenação é constante para a quantidade fixa de cem elementos, de acordo com o método *Bubble Sort*.

Figura 26: Gráfico do tempo de resposta pela quantidade de repetições.



Fonte: Elaborada pelo autor.

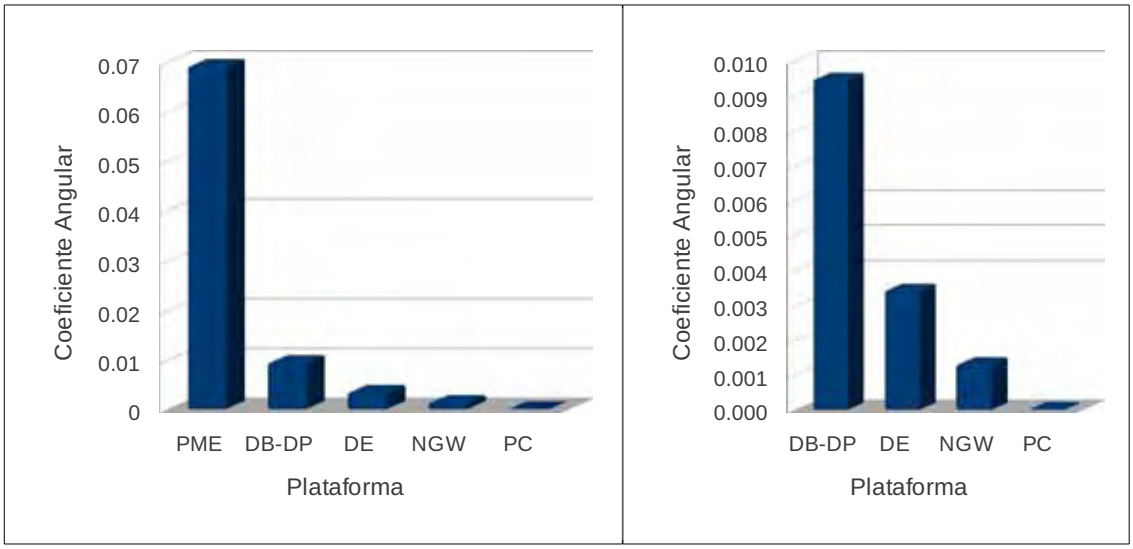
Na Tabela 9 apresenta-se os coeficientes angulares das regressões lineares de cada uma das curvas, permitindo observar de forma mais clara a diferença em magnitude do tempo de resposta de cada plataforma. Estas informações também são apresentadas no gráfico de barras da Figura 27 (a). Nota-se que o coeficiente angular para a plataforma PME-10 é significativamente maior, motivando sua exclusão no gráfico da Figura 27 (b) para melhor comparação em escala com as demais plataformas. Desta forma pode ser observado que a plataforma PC continua apresentando coeficiente angular desprezível comparado com as plataformas embarcadas. Ratifica-se que o objetivo desta pesquisa refere-se ao desenvolvimento de aplicações sobre plataformas embarcadas.

Tabela 9: Coeficientes angulares das curvas de tempo de resposta pelas repetições.

	PME-10	**DB-DP11115**	**DE2**	**NGW100**	**PC**
Coeficiente angular	0,069209	0,009473	0,003443	0,001321	0,000021

Fonte: Elaborada pelo autor.

Figura 27: Coeficientes angulares do tempo de resposta médio pelas repetições.

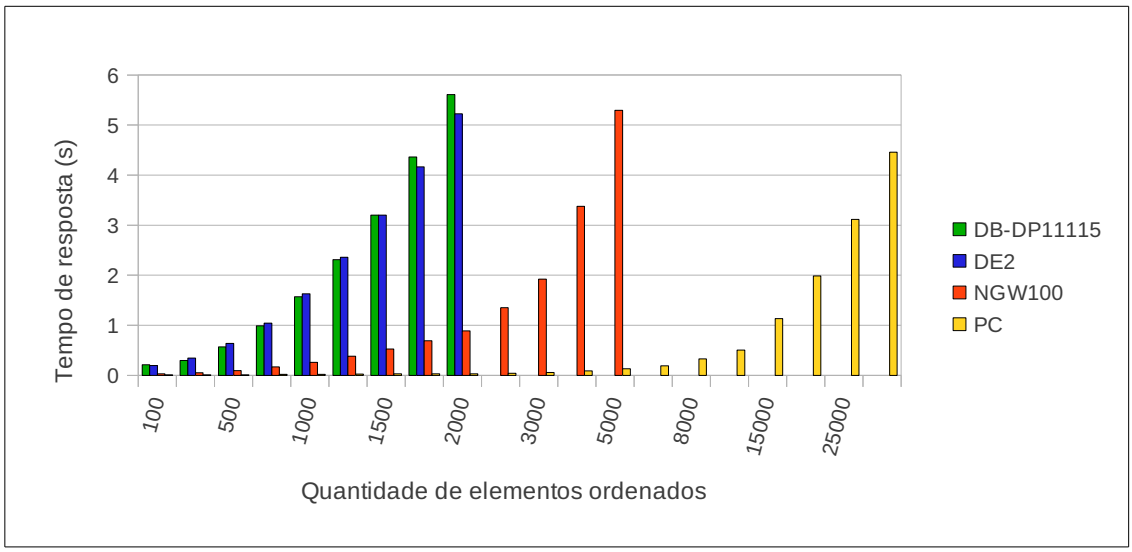


(a) (b)

Fonte: Elaborada pelo autor.

Na Figura 28 apresenta-se o gráfico produzido com os dados das Tabelas 7, onde pode ser observado o comportamento quadrático dos tempos de respostas, provenientes da complexidade $O(n^2)$ do método *Bubble Sort*.

Figura 28: Gráfico do tempo de resposta pela quantidade de elementos.



Fonte: Elaborada pelo autor.

Assim como nos gráficos das Figuras 26 e 27, também pode ser observada a diferença computacional das plataformas, para a ordenação de dois mil elementos.

Enquanto as plataformas DB-DP11115 e DE2 superam os cinco segundos, a

plataforma NGW100 não atinge um segundo e a plataforma PC ainda apresenta tempos de respostas desprezíveis.

Também foi realizado nestas plataformas um teste simples de capacidade de transferência de dados pela interface Ethernet de cada uma delas. Cada uma das plataformas foi conectada diretamente à outra máquina PC que hospedava a aplicação Cliente HTTP, por meio de um cabo *crossover*, evitando-se assim o compartilhamento do barramento de rede, com tráfego e colisões de pacotes.

Para se chegar a uma taxa de transferência comparativa entre as plataformas, hospedou-se um arquivo de 57,6 kB em cada uma das plataformas e realizou-se o download pela aplicação Cliente HTTP, obtendo-se a média dos tempos de respostas. Calculou-se então a taxa de transferência média, conforme os dados apresentados na Tabela 10.

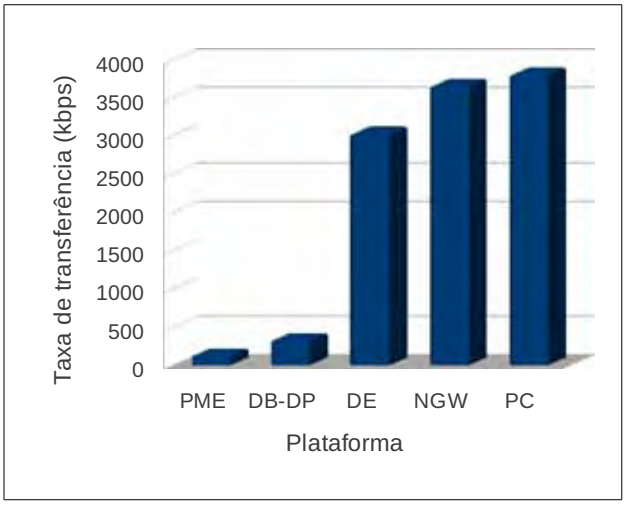
Tabela 10: Taxa de transferência média das plataformas.

	PME-10	**DB-DP11115**	**DE2**	**NGW100**	**PC**
Tx. transferência	130,1 kbps	333,19 kbps	3,03 Mbps	3,66 Mbps	3,80 Mbps

Fonte: Elaborada pelo autor.

Na Figura 29 apresenta-se o gráfico de barras da taxa de transferência obtida por cada plataforma. Como pode ser observado, as taxas de transferências de dados das plataformas DE2, NGW100 e PC apresentaram-se muito próximas, enquanto a plataforma PME-10 e DB-DP11115 apresentaram taxas de transferências de dados bem inferiores as demais plataformas.

Figura 29: Taxa de transferência média obtida por cada plataforma.



Fonte: Elaborada pelo autor.

Outra medição realizada foi a da potência consumida durante operação de ordenação para cada uma das plataformas, conforme dados da Tabela 11. A obtenção da potência instantânea consumida foi realizada de forma indireta, pelo monitoramento da corrente da fonte de alimentação, multiplicada pela tensão da fonte.

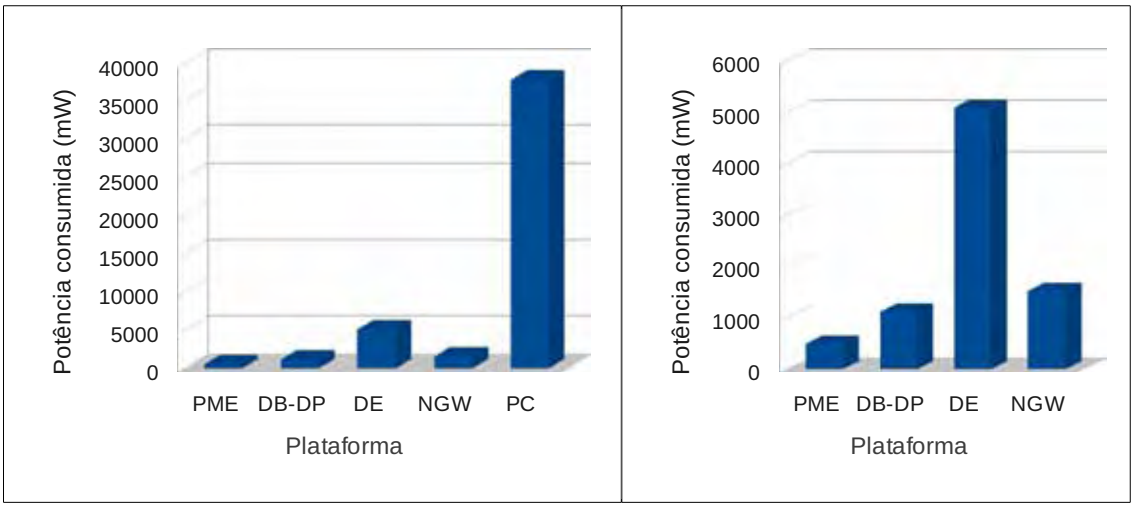
Tabela 11: Potência consumida em operação.

	PME-10	**DB-DP11115**	**DE2**	**NGW100**	**PC**
Potência	495 mW	1115 mW	5085 mW	1521 mW	37,8 W

Fonte: Elaborada pelo autor.

Na Figura 30 (a) apresenta-se o gráfico de barras da potência instantânea consumida por cada plataforma. Nota-se que a plataforma PC apresenta o maior consumo entre as plataformas. Na Figura 30 (b) são comparadas apenas as plataformas embarcadas, onde a DE2 apresenta maior consumo comparada com as demais.

Figura 30: Gráfico da potência consumida em operação por cada plataforma.



(a) (b)

Fonte: Elaborada pelo autor.

Dentre as plataformas analisadas, observa-se que a plataforma DE2 é a com maior quantidade de periférios na placa de circuito impresso (PCB - *Printed Circuit Board*). As plataformas PME-10 e DB-DP11115 possuem elementos no próprio *chip* do microcontrolador, como as memória SRAM e Flash. Estas características justificam parcialmente as diferenças de consumo de potência entre as plataformas.

Nas medições das potências consumidas, chegou-se a aferir a potência em operação e em espera de cada plataforma, no entanto a diferença percebida foi desprezível, exceto para a plataforma PC. Conclui-se com estas medições que as plataformas embarcadas analisadas não apresentam mecanismos de gerenciamento de potência dinâmica.

Na Tabela 12 apresentam-se os custos estimados em dólares para a aquisição de cada uma das plataformas. Os valores encontrados em Reais foram convertidos para Dólares considerando a taxa de câmbio de 1,75 R$/U$ e para as cotações realizadas no exterior foi considerado uma alíquota de importação de 60% sobre o preço base.

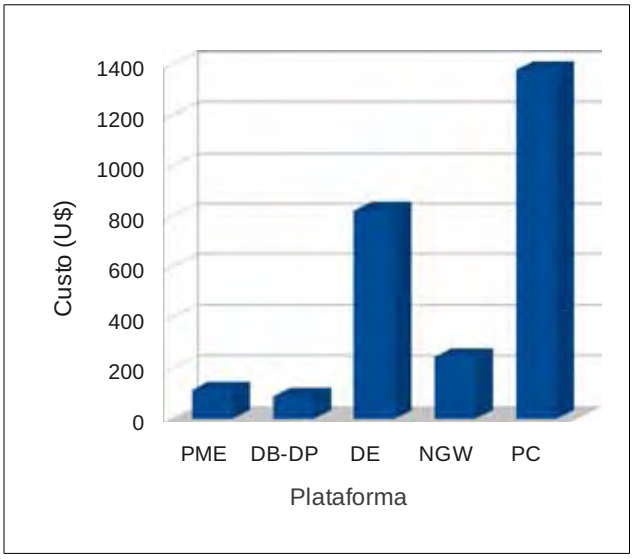
Tabela 12: Custo estimado de aquisição de cada uma das plataformas.

Custo	**PME-10***	**DB-DP11115****	**DE2*****	**NGW100*****	**PC******
Preço	U$ 108,00	U$ 50,00	U$ 495,00	U$ 134,00	U$ 1.315,00
Frete	U$ 7,00	U$ 10,00	U$ 32,00	U$ 32,00	U$ 64,00
Tarifa importação	-	U$ 30,00	U$ 297,00	U$ 80,00	-
Total	U$ 115,00	U$ 90,00	U$ 824,00	U$ 246,00	U$ 1.379,00

Fonte: * 2EI Engenharia (2010), ** Sure Electronics (2012), *** DigiKey (2010), **** Dell (2010).

No gráfico de barras da Figura 31 apresentam-se os custos totais estimados de cada uma das plataformas, onde a DE2 mostra-se como a de maior custo dentre as plataformas embarcadas desta pesquisa.

Figura 31: Gráfico do custo estimado total de cada plataforma.



Fonte: Elaborada pelo autor.

Outra análise realizada para as plataformas embarcadas estudadas foi o custo do *chip* principal e dos *chips* necessários para se confeccionar uma placa semelhante. Na Tabela 13 apresentam-se os valores de cada dispositivo.

Tabela 13: Custo estimado do *chip* principal e *chips* dependentes.

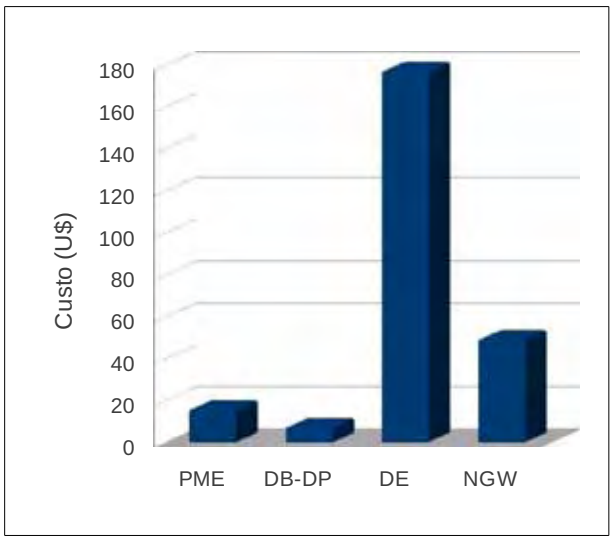
Dispositivo	**PME-10***	**DB-DP11115****	**DE2***	**NGW100***
Chip principal	U$ 16,00	U$ 7,50	U$ 150,00	U$ 22,00
16 MB SDRAM	-	-	U$ 3,00	U$ 3,00
4 MB Flash	-	-	U$ 1,50	U$ 1,50
CPLD MAX II	-	-	U$ 23,00	U$ 23,00
Total	U$ 16,00	U$ 7,50	U$ 177,50	U$ 49,50

Fonte: * DigiKey (2010), ** DigiKey (2012).

Observa-se que a memória SDRAM, a memória Flash e um CPLD são necessários somente nas plataformas DE2 e NGW100, onde foram considerados os mesmos dispositivos. Os demais elementos como PCB, terminais, osciladores, *driver* e conector Ethernet etc., não foram considerados por entender que seriam necessários nas quatro plataformas.

Na Figura 32 apresenta-se o gráfico de barras dos custos estimados totais do *chip* principal e dos *chips* associados a cada uma das plataformas pesquisadas, onde observa-se o maior custo apresentado pela plataforma DE2.

Figura 32: Gráfico do custo estimado total do *chip* principal e *chips* dependentes.



Fonte: Elaborada pelo autor.

Com os resultados aqui apresentados foi possível comparar as plataformas estudadas quanto a capacidade de processamento, analisada pelos tempos de respostas, tanto pelas repetições das ordenações, quanto pelas variações das quantidades de elementos que foram ordenados. Foi possível também comparar a capacidade de transferência de dados, os consumos de energia, custo das plataformas da forma que foram estudadas e custo de seus principais dispositivos, imaginando que em um emprego efetivo em escala de alguma dessas plataformas, elas deverão ser produzidas e não adquiridas na forma que foram.

2.5 CONCLUSÕES

Com a pesquisa apresentada neste capítulo foi possível familiarizar-se com o desenvolvimento de aplicações para algumas plataformas embarcadas, com comunicação em rede sobre TCP/IP, pelas interfaces Ethernets. Embora o resultado funcional das aplicações tenha sido o mesmo, foi possível perceber que as limitações da plataforma PME-10 torna o desenvolvimento com comunicação em rede muito mais complexo que nas demais plataformas. Esta complexidade de desenvolvimento e a reduzida capacidade computacional apresentada pela plataforma PME-10, inviabiliza seu uso em transdutores inteligentes diretamente em nível de serviços. Neste propósito, as plataformas com sistemas operacionais embarcados mostraram-se interessantes, motivando outras pesquisas para a exploração da comunicação orientada a serviços.

Comparando as plataformas DE2 e NGW100, para o mesmo nível de recursos de softwares apresentado pelos sistemas operacionais embarcados, a NGW100 mostrou-se mais eficiente por apresentar maior capacidade computacional para a aplicação desenvolvida, com menor custo de aquisição, tanto da plataforma, quanto dos *chips* separadamente. No entanto, a DE2 é uma plataforma com hardware reconfigurável, que pode oferecer recursos de hardware para o desenvolvimento de lógicas dedicadas, que superam significativamente o desempenho de qualquer aplicação executada por uma unidade de processamento sequencial de instruções, entretanto, se tal tipo de recurso não for necessário para a continuidade dos trabalhos, em direção a aplicações orientadas a serviços, esta plataforma poderá continuar sendo subutilizada, sendo realmente mais adequado o emprego de uma plataforma com arquitetura fixa, como a NGW100.

No LPSSD[14] (Laboratório de Processamento de Sinais e Sistemas Digitais) existem

14 Laboratório de pesquisa da Unesp-FEIS.

outras pesquisas sobre a plataforma DE2 que melhor exploram seus recursos, como a capacidade de reconfiguração de hardware.

Apesar da plataforma PME-10 não ter despertado interesse para a continuidade das pesquisas, em direção ao desenvolvimento orientado a serviços, onde foi escolhida a NGW100, a mesma pode ser uma boa opção para o desenvolvimento de TIMs, pois não exigem maior capacidade computacional.

Além das plataformas PME-10 e DB-DP11115 terem apresentado os menores custos dentre as plataformas pesquisadas, são aplataformas que apresentam conversores ADs (*Analog-to-Digital*), essenciais para o condicionamento de sinais analógicos de sensores.

Outra possibilidade é transferir parte das implementações que demandam maior memória e processamento para máquinas externas na rede, ou em computação em nuvem, o que pode viabilizar a utilização de plataformas mais restritas como a PME-10 e a DB-DP11115 em implementações plenas de transdutores inteligentes, a custos bem menores que os das demais plataformas estudadas, o que é proposto no capítulo 5 e realizado no capítulo 6 com a plataforma DB-DP11115.

2.6 PROPOSTAS DE TRABALHOS FUTUROS

Ao término dos estudos apresentados neste capítulo, algumas dúvidas e motivações sugerem frentes de pesquisas futuras, como as que seguem:

- Avaliação das plataformas DE2 e NGW100 operando sobre protocolos de comunicação orientados a serviços;
- Avaliação das plataformas DE2 e NGW100 operando em barramento de campo;
- Projeto de plataforma própria baseada na arquitetura da DE2 ou NGW100;
- Desenvolvimento de um NCAP sobre as plataformas DE2 ou NGW100;
- Desenvolvimento de um TIM sobre as plataformas PME-10 ou DB-DP11115;
- Transferência de implementações para computação em nuvem.

CAPÍTULO 3

ARQUITETURAS ORIENTADAS A SERVIÇOS SOBRE PLATAFORMAS EMBARCADAS

3.1 INTRODUÇÃO

Sistemas de informação corporativos são muitas vezes distribuídos e heterogêneos por diversas razões, dentre elas os sistemas legados que já se mostraram eficientes por anos, ou pela fusão de corporações de diferentes culturas e modelos de negócio, pelo avanço do B2B[15] (*Business-to-Business*), ou ainda por questões de infraestrutura.

Neste contexto, embora muitas vezes deseja-se remodelar e reconstruir totalmente um sistema de informação grande e complexo, centralizando os recursos e homogeneizando as tecnologias, esta pode não ser a melhor solução em termos de flexibilidade, escalabilidade e, principalmente, custos.

As arquiteturas orientadas a serviços (SOAs) têm se mostrado um emergente paradigma para integração e manutenção de sistemas de informação distribuídos, complexos e heterogêneos, não que a heterogeneidade das tecnologias seja um requisito, ou mesmo desejada. A questão é que como tudo é modelado como serviços e o acesso ao serviço só depende do protocolo de comunicação de rede adotado, a tecnologia em que a aplicação está implementada não é relevante nesta arquitetura.

As SOAs são independentes de plataformas e linguagens de programação, necessitando somente interfaces bem definidas nos serviços e um protocolo padrão de comunicação em rede, de preferência aberto, para a interoperabilidade de serviços. A SOA não é uma tecnologia, é apenas um conceito, paradigma ou forma de se modelar a arquitetura de um sistema de software.

3.1.1 Objetivos

Um dos objetivos específicos deste capítulo foi compreender os mecanismos orientados a serviços e as possibilidades de emprego em plataformas embarcadas, para

15 Tecnologia de transações comerciais entre empresas utilizando redes de computadores.

avançar no objetivo geral de definir uma arquitetura para publicação e universalização do acesso a transdutores inteligentes.

Especificamente, investigou-se nesta pesquisa padrões e tecnologias de arquiteturas orientadas a serviços, com o objetivo de empregá-las sobre plataformas embarcadas, ou em serviços distribuídos, para prover acesso em alto nível de abstração a redes de transdutores inteligentes, realizando testes de conectividade e acesso.

3.1.2 Justificativas

Assim como a SOA mostra-se poderosa na interoperabilidade de sistemas distribuídos complexos, heterogêneos e legados, a possibilidade de empregá-la em redes de transdutores inteligentes pode ser muito proveitosa. Aliado aos objetivos gerais deste trabalho, a SOA torna-se, no mínimo, algo a ser investigado.

3.1.3 Trabalhos relacionados

Song e Lee (2008a) propuseram o emprego de *Web Services* para estender o padrão IEEE 1451, elevando-o ao nível de serviço, o que denominaram de STWS (*Smart Transducers Web Service*). Os códigos do STWS implementados por Song e Lee foram postados no repositório do projeto Open1451.

Barfield (COOK; BARFIELD, 2006) projetaram e implementaram um pequeno servidor de arquivos com *Web Services* e RMI e realizaram comparações entre as duas implementações.

Li et al. (2009) descrevem uma abordagem com *Web Services* no padrão JAX-WS (*Java API for XML - based Web Service*) empregados em um sistema de tempo real de uma mineradora de carvão, que possui diversos sensores e atuadores distribuídos.

3.1.4 Organização do capítulo

Na seção 3.2 deste capítulo são apresentadas algumas definições de SOA, suas principais características e algumas tecnologias possíveis de serem adotadas para a sua implementação.

Na seção 3.3 detalha-se um pouco mais a arquitetura de *Web Service*, seus principais

elementos e alguns de seus padrões.

Na seção 3.4 estão agrupados os procedimentos experimentais realizados tanto na plataforma PC, com implementação de um *Web Service* e um sistema RESTful, como na plataforma embarcada, com diversas implementações e tentativas de instalação de aplicações que possam prover suporte ao desenvolvimento de serviços.

Na seção 3.5 são relatadas as conclusões deste capítulo.

Na seção 3.6 são elencadas algumas propostas de trabalhos futuros relacionados ao tema.

3.2 ARQUITETURAS ORIENTADAS A SERVIÇOS

O termo SOA não é novo, Schulte e Natis (1996) descrevem que uma arquitetura orientada a serviços é um estilo de computação multicamadas que ajuda as organizações a compartilharem dados e lógica entre várias aplicações e modos de utilização. Natis (2003) relata que o principal benefício de SOA é a oportunidade para aumentar o desenvolvimento, a implantação, a manutenção e a extensão de aplicativos de negócios.

O modelo de referência da OASIS (*Organization for the Advancement of Structured Information Standards*) apresenta a SOA como um paradigma para organizar e utilizar capacidades distribuídas que podem estar sob o controle de diferentes domínios proprietários (OASIS, 2006).

A Oracle (2008) aponta a SOA como uma abordagem arquitetônica que facilita a criação de serviços de negócios interoperáveis e flexivelmente acoplados, para fácil compartilhamento dentro das empresas e entre elas. Cita ainda que o conceito de SOA extrai seu verdadeiro valor da reutilização e agilidade proporcionadas ao desenvolvi-mento de sistemas em software.

3.2.1 Características de SOA

Várias são as definições de SOA, dentre a maioria delas percebe-se características relevantes como (JOSUTTIS, 2008; SOUZA, 2006):

- alto nível de abstração – os elementos de SOA operam em alto nível de abstração, ficando mais próximos do modelo de negócio e independentes das plataformas de TI, que podem ser heterogêneas;

- heterogeneidade de plataformas e linguagens – como a comunicação entre os elementos de SOA se dá por troca de mensagens de rede, por um protocolo padrão adotado e uma interface bem definida, a plataforma de hardware e software que roda a aplicação não é relevante, assim como a linguagem de programação adotada, visto que o que garante a interoperabilidade é o protocolo de rede adotado e as interfaces de troca de mensagens definidas;
- interoperabilidade – uma vez implantado um caso de uso, como um serviço do sistema, ele é facilmente interoperável com outros casos de uso deste, ou de outros sistemas de SOA, desde que adotem o mesmo protocolo de rede e as respectivas interfaces dos serviços;
- flexibilidade – neste cenário, um serviço pode ser facilmente substituído ou estendido, sem que outras partes do sistema necessitem de qualquer alteração. Desde que as interfaces originais sejam mantidas, novos recursos podem ser adicionados, sem que os casos de uso que dependem do serviço original percebam qualquer mudança;
- escalabilidade – a flexibilidade presente nos serviços pode ser explorada com técnicas de computação de alto desempenho, a fim de manter os serviços escaláveis, com possibilidade de tratamento fragmentado dos serviços com maior demanda, consequentemente, todo o sistema será escalável;
- tolerância à falhas – como em qualquer sistema distribuído, é necessária uma atenção especial com a integração dos serviços e o *frontend*, com mecanismos de tolerância à falhas como: detecção de falhas, mascaramento de falhas, recuperação de falhas, redundância etc. É preciso assumir que sistemas complexos e distribuídos falham e, sempre que possível, o sistema necessita de mecanismos para continuar operando;
- fraco acoplamento – é uma característica do acoplamento em alto nível de abstração dos elementos de um sistema distribuído. O fraco acoplamento também contribui para a flexibilidade do sistema, em contrapartida, se comparado com sistemas fortemente acoplados, como na computação de alto desempenho, o fraco acoplamento pode prejudicar o desempenho computacional;
- composição – novos serviços do sistema podem ser compostos a partir de serviços já existentes, semelhante às pilhas de abstração encontradas em modelos de software organizados em camadas, ou às heranças de classes do modelo orientado a objetos;
- reusabilidade – muito além do encapsulamento e reuso de código da programação

orientada a objetos, a SOA é o estado da arte em reusabilidade, pois os demais elementos do sistema apenas acessam o serviço e utilizam-se de seus recursos, sem nenhuma transferência de código, apenas mensagens de dados são trafegadas na rede;

- granularidade – a granularidade de SOA pode variar de níveis de procedimentos a casos de uso completos, no entanto, a fim de manter maior flexibilidade ao sistema, procura-se granular os serviços pelo menos em métodos dos casos de uso, o que pode ser considerado uma alta granularidade;
- ubiquidade – a localização geográfica do serviço não é relevante, ao mesmo tempo o serviço está disponível em qualquer ponto do sistema. Esta onipresença faz com que os serviços sejam ubíquos.

3.2.2 Tecnologias para SOA

Embora a SOA seja independente de tecnologia, quando se deseja implementar um serviço, inevitavelmente, um padrão e tecnologia devem ser escolhidos e adotados. A seguir estão elencadas algumas tecnologias encontradas, possíveis de serem empregadas em SOA:

- CORBA (*Common Object Request Broker Architecture*) – CORBA é uma arquitetura de desenvolvimento de software distribuído especificada e mantida pelo OMG (*Object Management Group*), utiliza a IDL (*Interface Definition Language*) para desenvolvimento das interfaces de comunicação entre os elementos do sistema distribuído. A IDL originalmente realizava o mapeamento dos métodos da interface para métodos escritos em C. Atualmente existem especificações da IDL para mapeamento em C++, Smalltalk, Java etc. (OMG, 2009);
- Java RMI (*Remote Method Invocation*) – Java RMI é uma tecnologia que permite a construção de aplicações distribuídas em Java em que os métodos de objetos remotos podem ser chamados pelas máquinas virtuais presentes em diferentes hospedeiros. Para isso a Java RMI realiza a serialização de objetos para empacotar e desempacotar parâmetros, suportando conceitos de orientação a objetos, como o polimorfismo (ORACLE, 2011a);
- COM (*Component Object Model*) – COM é uma tecnologia baseada nos sistemas operacionais da Microsoft que permite a comunicação de componentes de software. Via APIs (*Application Programming Interfaces*) do Microsoft Windows é possível

adotar uma variedade de linguagens de programação como: C++, C#, Visual Basic, Object Pascal etc. A família de tecnologias COM da Microsoft inclui as tecnologias: COM+, DCOM (*Distributed* COM) e ActiveX (MICROSOFT, 2011a);

- .NET Remoting – .NET Remoting é um *framework*[16] da Microsoft, baseado na tecnologia .NET, que permite o desenvolvimento de aplicações com objetos distribuídos, apresenta diversos recursos para gerenciamento e manutenção de objetos remotos. Pode ser utilizada codificação binária para troca de mensagens quando o desempenho é crítico, ou mensagens em XML sobre SOAP quando a interoperabilidade entre objetos é mais relevante (OBERMEYER; HAWKINS, 2001);
- *Web Service* – *Web Service* é uma arquitetura em software projetado para suportar interoperabilidade entre máquinas em rede, adota diversas tecnologias livres e padronizadas como: XML, SOAP e WSDL. Tem uma interface descrita em formato processado pela máquina hospedeira, especificamente o WSDL. Outros sistemas interagem com o *Web Service* da maneira prescrita, usando mensagens SOAP, tipicamente transmitida usando HTTP, com serialização XML em conjunto com outras normas relacionadas à Web (W3C, 2004). Assim, *Web Services* garantem a interoperabilidade de serviços com mensagens XML sobre SOAP, como o .NET Remoting, independente da linguagem de programação ou plataforma de hardware;
- REST (*REpresentational State Transfer*) – REST é uma arquitetura proposta por Fielding (FIELDING, 2000) para sistemas hipermídia distribuídos, utilizando apenas recursos do HTTP, sendo assim uma arquitetura mais leve e simples que as citadas anteriormente e ainda independente da linguagem de programação ou plataforma de hardware. Um exemplo clássico de aplicação baseada em REST é uma aplicação CRUD (*Create, Read, Update and Delete*), implementada com os métodos POST, GET, PUT e DELETE do HTTP, respectivamente. Sistemas baseados em REST são frequentemente denominados de RESTful;
- JSON (*JavaScript Object Notation*) – JSON é um formato de estruturação de dados que tem se apresentado como uma alternativa ao XML (JSON, 2011). Dentre as vantagens do formato JSON sobre XML está a notória redução na quantidade de dados das estruturas de organização e a reinvindicada simplicidade oferecida aos *parsers* JSON, comparado aos *parsers* XML. A reduzida quantidade de dados utilizados para

16 Conjunto de classes que agilizam e simplificam o desenvolvimento em uma dada linguagem de programação.

representar a mesma informação é uma vantagem relevante, principalmente em sistemas em que a transferência de dados em rede é crítica. O emprego de JSON em sistemas RESTful tem se popularizado, principalmente em sistemas que prezam pela simplicidade e racionalização dos recursos computacionais;

- CoAP (Constrained Application Protocol) – CoAP é um protocolo de transferência Web especializado para dispositivos embarcados restritos, em especial quanto ao consumo de energia elétrica. Este protocolo emprega a arquitetura REST para a transferência de dados com o protocolo de aplicação HTTP, sobre o protocolo de transporte UDP, com confiabilidade opcional, baixo *overhead* de cabeçalho, baixa complexidade de análise e trocas de menssagens assíncronas (SHELBY et al., 2012). CoAP tem sido empregado principalmente em RSSFs, onde o consumo de energia é um fator crítico.

3.3 *WEB SERVICES*

Dentre as tecnologias relatadas anteriormente, escolheu-se aprofundar as pesquisas em *Web Services*. Além da independência de linguagem de programação e plataforma de hardware, influenciou também na decisão de escolha a popularidade desta tecnologia em aplicações Web, que é o propósito dos serviços de sensores. Vale observar que sistemas RESTful tem se popularizado também na Web. Ademais, os serviços descritos na seção 3.1.3, *Trabalhos relacionados*, relatam a utilização desta tecnologia.

A seguir são descritos os principais elementos de um *Web Service*, assim como algumas de suas especificações.

3.3.1 Elementos e especificações de *Web Services*

Um *Web Service* é composto por várias tecnologias relacionadas, por isso são brevemente relatadas a seguir as tecnologias XML, SOAP, WSDL e UDDI (*Universal Description, Discovery and Integration*), embora existam muitas outras. Conforme salientado por Denning (2004), há também muitas maneiras de visualizar essas tecnologias, assim como existem muitas maneiras de construir e usar *Web Services*.

Embora o HTTP já tivesse universalizado o meio de transferência de mensagens na Web, os padrões para essa transferência ainda vinculavam forma e conteúdo, como o HTML.

Atualmente a W3C especifica e recomenda o emprego de XML para a estruturação e transferência de dados na rede mundial. Esta recomendação tem sido largamente empregada, proporcionando uma revolução na organização da informação contida na Web, pois o conteúdo tem se desvinculado da forma, permitindo que diversas fontes de informação apenas as postem na rede, deixando livre para cada aplicação decidir como e de que forma publicar a informação (W3C, 2008).

Os *Web Services* utilizam o SOAP, que é um protocolo de serviço baseado em XML e também especificado pela W3C, que permite a troca de dados entre os clientes e o *Web Service*. O SOAP pode operar sobre diversos protocolos de aplicação como o SMTP (*Simple Mail Transfer Protocol*), o FTP e o HTTP, sendo este último mais comumente empregado (W3C, 2007).

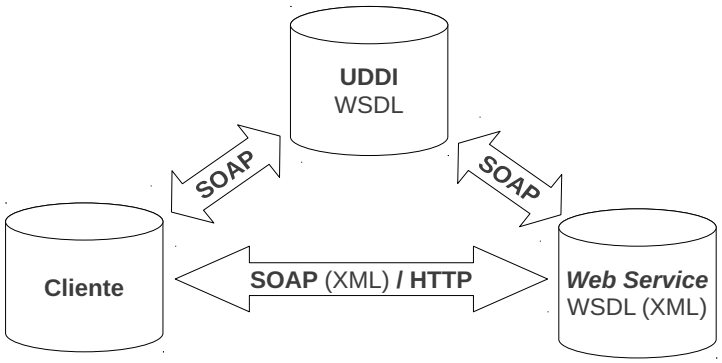
Outra especificação da W3C é a WSDL, que descreve, também em XML, interfaces de métodos disponíveis em um *Web Service*. Esta descrição é denominada de *Contrato do Serviço* (W3C, 2001). Os contratos de serviços são publicados pelos próprios *Web Services*, no entanto, para que as aplicações clientes possam obtê-los, os contratos podem ser enviados pelos *Web Services* para um serviço de registro UDDI, provido por algum *Web Service* da rede. A especificação UDDI é um padrão da OASIS para publicar e descobrir serviços em SOA.

O *Web Service* da rede que presta o serviço de registro UDDI publica os contratos recebidos de forma centralizada, permitindo que aplicações clientes encontrem mais facilmente *Web Services* e serviços disponíveis na rede. Para tanto, é suficiente que a aplicação cliente conheça apenas a localização do *Web Service* que presta o serviço de registros UDDI, caso contrário, seria necessário que a aplicação cliente varresse toda a rede a procura de *Web Services* e serviços ativos (OASIS, 2002), ou que fosse identificado manualmente a localização de cada *Web Service* utilizado pela aplicação cliente.

Na Figura 33 apresenta-se uma ilustração da interação entre elementos de um *Web Service*. Uma aplicação cliente encontra um serviço de interesse, através de um serviço de registros UDDI.

Com a informação da localização do *Web Service* que hospeda o serviço desejado, o cliente solicita o contrato de serviço via SOAP, sobre HTTP. Com o contrato WSDL o cliente implementa métodos de interface com o serviço desejado e passa a consumi-lo via SOAP sobre HTTP.

Figura 33: Interação entre elementos de um *Web Service*.



Fonte: Elaborada pelo autor.

3.3.2 Padrões de *Web Services*

Um grande problema enfrentado pelos desenvolvedores de *Web Services* é a interoperabilidade de serviços, embora XML, SOAP e WSDL tenham especificações bem definidas, existem diversas maneiras de utilizar essas tecnologias para implementar *Web Services*.

Com o objetivo de manter a interoperabilidade entre *Web Services*, existem inúmeros esforços de padronização de *Web Services*, mantido por organizações como WS-I (*Web Services Interoperability Organization*), OASIS e W3C.

O JAX-RPC (*Java API for XML - based Remote Procedure Call*) é um esforço da comunidade Java (SUN, 2003) para eliminar o problema de interoperabilidade e fornecer uma API bem definida para implementação de *Web Services*. Este padrão define e utiliza um mecanismo de chamada de procedimento remoto baseado em XML. O *Web Service*, prestador de serviços, define seus serviços usando APIs padronizadas, com interfaces descritas em WSDL. O cliente, o consumidor do serviço, acessa o *Web Service* utilizando APIs também padronizadas.

O JAX-WS estende o padrão JAX-RPC, permitindo, entre outras características, a introdução de funcionalidades assíncronas e mapeamento para tipos de dados complexos, conforme comparação de tipos de dados apresentado na Tabela 14.

O JAX-WS utiliza o JAXB (*Java Architecture for XML Binding*) para realizar o mapeamento de tipos de dados e interfaces de serviços (SUN, 2006), para métodos a partir do Java 5 (JRE - *Java Runtime Environment* 1.5.X), enquanto o JAX-RPC realiza mapeamento até o Java 4 (JRE 1.4.X). Desta forma o JAX-WS permite a utilização de novas características disponíveis a partir do Java 5.

Tabela 14: Alguns tipos de dados mapeados em JAX-RPC e JAX-WS.

Tipo	**JAX-RPC 1.1**	**JAX-WS 2.0**
xsd:anySimpleType	java.lang.String	java.lang.Object
xsd:duration	java.lang.String	javax.xml.datatype.Duration
xsd:dateTime	java.util.Calendar	javax.xml.datatype.XMLGregorianCalendar
xsd:time	java.util.Calendar	javax.xml.datatype.XMLGregorianCalendar
xsd:date	java.util.Calendar	javax.xml.datatype.XMLGregorianCalendar
xsd:gYearMonth	java.lang.String	javax.xml.datatype.XMLGregorianCalendar
xsd:gYear	java.lang.String	javax.xml.datatype.XMLGregorianCalendar
xsd:gMonthDay	java.lang.String	javax.xml.datatype.XMLGregorianCalendar
xsd:gMonth	java.lang.String	javax.xml.datatype.XMLGregorianCalendar
xsd:gDay	java.lang.String	javax.xml.datatype.XMLGregorianCalendar
xsd:anyURI	java.net.URI	java.lang.String
xsd:NMTOKENS	java.lang.String[]	java.util.List<java.lang.String>
xsd:IDREF	java.lang.String	java.lang.Object
xsd:IDREFS	java.lang.String[]	java.util.List<java.lang.Object>
xsd:ENTITY	não suportado	java.lang.String
xsd:ENTITIES	não suportado	java.util.List<java.lang.String>

Fonte: Butek e Gallardo (2006).

Embora o JAX-WS apresente mais recursos para o desenvolvedor, seu emprego é comprometido em muitos casos quando sistemas já existentes não suportam as especificações JAX-WS. JAX-WS também suporta RESTful (TYAGI, 2006), no entanto o padrão da comunidade Java específico para RESTful é o JAX-RS (SUN, 2008).

3.4 PROCEDIMENTOS EXPERIMENTAIS

Embora um dos objetivos deste capítulo seja empregar SOAs sobre plataformas embarcadas de hardware, inicialmente, buscou-se compreender melhor os *Web Services* e serviços RESTful implementando-os, com seus respectivos clientes, em uma plataforma PC convencional, evitando inicialmente as dificuldades impostas pelas restrições de recursos de uma plataforma embarcada.

A seguir, são apresentadas as implementações e tentativas de instalação de aplicações sobre algumas plataformas de hardware.

3.4.1 A plataforma PC

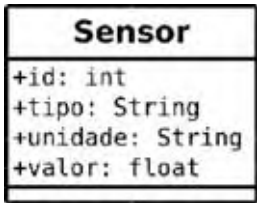
Para o desenvolvimento na plataforma PC, foi adotada a linguagem de programação Java com o IDE NetBeans, o SGBD (Sistema de Gerenciamento de Banco de Dados) PostgreSQL e o servidor de aplicação GlassFish, sobre o JRE, empregando o sistema operacional Linux Ubuntu 10.04.

Mais uma vez, é válido enfatizar que buscou-se nesta implementação apenas compreender melhor os mecanismos de um *Web Service* e serviços RESTful, visto que o conjunto de ferramentas utilizadas: PostgreSQL, GlassFish e JRE, inviabilizam seu emprego em uma plataforma embarcada, dado as limitações de recursos destas plataformas.

3.4.1.1 Implementação de um Web Service

Para o modelo de dados do *Web Service* foi considerada uma classe, denominada *Sensor*, com os atributos *id*, *tipo*, *unidade* e *valor*, conforme os tipos de dados apresentados no diagrama de classe da Figura 34.

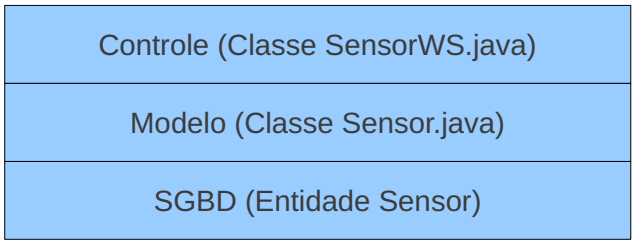
Figura 34: Diagrama de classe Sensor.



Fonte: Elaborada pelo autor.

A partir da classe *Sensor* modelada, utilizou-se o SGBD PostgreSQL para criar a entidade *Sensor*, da qual, a partir do IDE NetBeans e alguns de seus assistentes, gerou-se as camadas de *modelo* e *controle* (serviço) da implementação do *Web Service*, no padrão JAX-WS, conforme identificado na Figura 35.

Figura 35: Camadas do *Web Service* implementado.

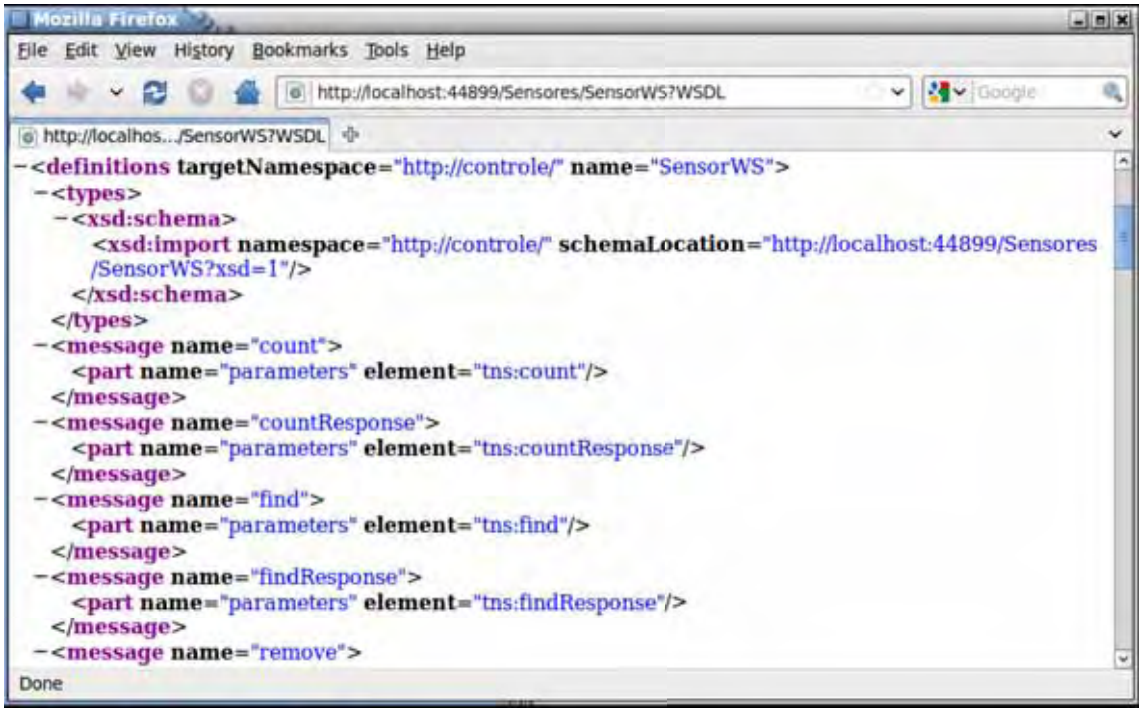


Fonte: Elaborada pelo autor.

Esta implementação do *Web Service* foi implantada no servidor de aplicações GlassFish e inicialmente testada utilizando um navegador para fazer a requisição do arquivo WSDL que descreve o serviço.

Na Figura 36 apresenta-se parte do arquivo WSDL gerado pelo *Web Service*, com dados estruturados em XML. Pode ser observada a descrição de alguns métodos como o *count* e o *find*. O método *count* retorna a quantidade de registros da classe *Sensor* existentes no servidor e o método *find* retorna um registro específico, referente ao atributo *id* passado como parâmetro.

Figura 36: Parte do arquivo WSDL gerado pelo *Web Service*.



Fonte: Elaborada pelo autor.

A partir da URL do arquivo WSDL, utilizou-se um assistente do NetBeans para gerar classes que consumissem o *Web Service* implantado. A partir dessas classes foi implementada uma aplicação cliente Java *desktop* para consumir o serviço.

Com o cliente consumidor do serviço funcionando, foram realizados testes monitorando o tráfego de dados na rede, para posterior análise e compreensão. Mesmo com o cliente já implementado e funcionando, foi possível observar que a primeira requisição realizada pelo cliente foi novamente o arquivo WSDL, conforme apresentado na Figura 37. O arquivo WSDL já havia sido lido do *Web Service* pelo assistente do NetBeans, para gerar as classes da aplicação cliente.

Esta troca do contrato de serviço do *Web Service* em tempo de execução demonstra que este modelo de SOA pode permitir alterações dinâmicas de recursos do serviço, com

atualização dinâmica nos clientes, sem a necessidade de reimplementação da aplicação cliente em tempo de projeto.

Figura 37: Requisição HTTP do cliente do *Web Service*.

```
GET /Sensores/SensorWS?WSDL HTTP/1.1
User-Agent: Java/1.6.0_20
Host: localhost:44899
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Fonte: Elaborada pelo autor.

Neste caso observou-se que o método HTTP utilizado para solicitar o arquivo WSDL foi o *GET* e que o atributo *User-Agent* enviado, demonstra que a comunicação da aplicação cliente foi realizada pelo JRE, em sua versão 1.6.0_20.

Como resposta o *Web Servive* retornou o arquivo WSDL sobre o protocolo HTTP, conforme apresentado na Figura 38. O atributo *Server* informa que o servidor de aplicação que hospeda o *Web Service* é o GlassFish, em sua versão 3.0.1.

Figura 38: Resposta HTTP do *Web Service*.

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.0
Server: GlassFish Server Open Source Edition 3.0.1
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Tue, 21 Sep 2010 14:50:08 GMT

<?xml version='1.0' encoding='UTF-8'?>
<definitions targetNamespace="http://controle/" name="SensorWS">
 … trecho omitido do WSDL ...
</definitions>
```

Fonte: Elaborada pelo autor.

Nas classes da aplicação cliente, geradas a partir do arquivo WSDL do servidor, o método de acesso ao serviço *findAll()* retorna um envelope SOAP contendo os dados registrados na entidade *Sensor* do SGBD.

Na Figura 39 apresenta-se o envelope SOAP enviado pelo cliente, para acessar o método *findAll()* do *Web Service*.

Figura 39: Requisição SOAP, invocando o método *findAll()* .

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header/>
   <S:Body>
      <ns2:findAll xmlns:ns2="http://controle/"/>
   </S:Body>
</S:Envelope>
```

Fonte: Elaborada pelo autor.

Na Figura 40 apresenta-se o envelope SOAP devolvido pelo servidor, contendo os dados de alguns registros da entidade *Sensor* do SGBD.

Figura 40: Resposta SOAP, retornando alguns registros da entidade *Sensor*.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:findAllResponse xmlns:ns2="http://controle/">
         <return>
            <id>1</id>
            <tipo>Termômetro</tipo>
            <unidade>ºC</unidade>
            <valor>27.6500</valor>
         </return>
         <return>
            <id>2</id>
            <tipo>Manômetro</tipo>
            <unidade>psi</unidade>
            <valor>32.1000</valor>
         </return>
         ... trecho omitido ...
      </ns2:findAllResponse>
   </S:Body>
</S:Envelope>
```

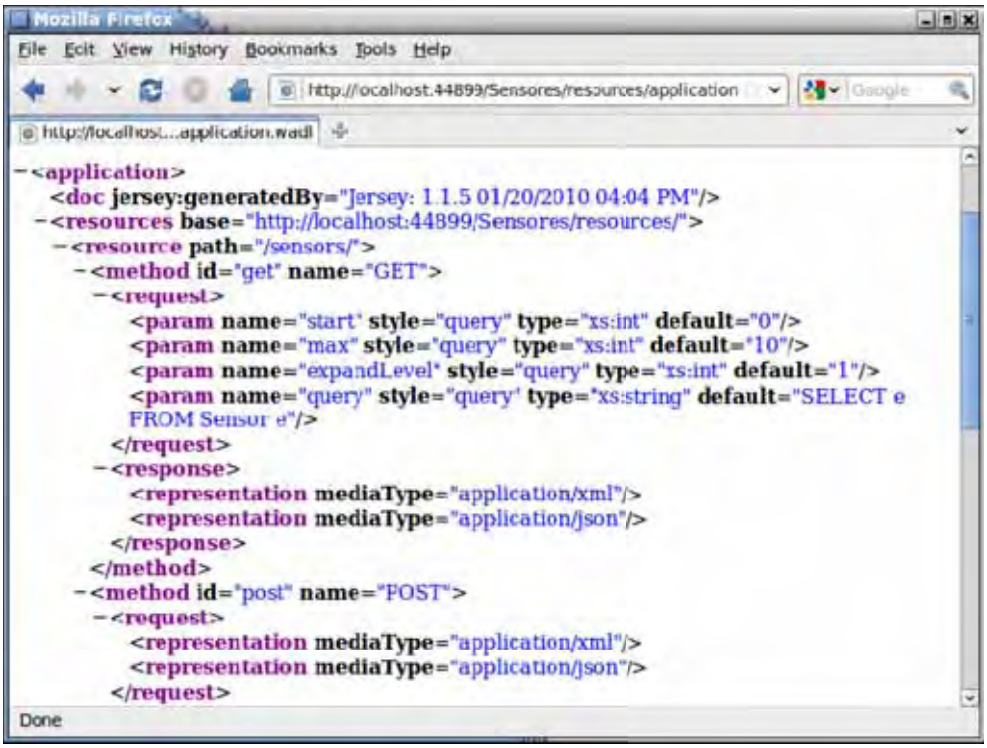
Fonte: Elaborada pelo autor.

No envelope SOAP pode ser analisado a estrutura XML em que os dados de um termômetro foram enviados, com o valor de 27,65 e unidade ºC. No mesmo envelope SOAP pode ser observado também os dados de um manômetro. Por entender que a demonstração já era suficiente os dados dos demais registros foram omitidos, entretanto, seguem a mesma estrutura até a *tag* de fechamento *</ns2:findAllResponse>*.

3.4.1.2 Implementação de um serviço RESTful

Da mesma forma que o *Web Service* desenvolvido, utilizando a mesma entidade *Sensor* do SGBD e outros assistentes do IDE NetBeans, foi implementado um serviço RESTful que adota tanto XML como JSON para transferência de dados. Na Figura 41 pode ser observado parte do arquivo WADL (*Web Application Description Language*) gerado pelo serviço RESTful. Este arquivo descreve em XML os recursos do serviço, semelhante ao arquivo WSDL dos *Web Services*, no entanto, de forma mais simples, refletindo a proposta desta arquitetura.

Figura 41: Parte do arquivo WADL gerado pelo serviço RESTful.




```
-<application>
  <doc jersey:generatedBy="Jersey: 1.1.5 01/20/2010 04:04 PM"/>
 -<resources base="http://localhost:44899/Sensores/resources/">
  -<resource path="/sensors/">
   -<method id="get" name="GET">
    -<request>
      <param name="start" style="query" type="xs:int" default="0"/>
      <param name="max" style="query" type="xs:int" default="10"/>
      <param name="expandLevel" style="query" type="xs:int" default="1"/>
      <param name="query" style="query" type="xs:string" default="SELECT e
      FROM Sensor e"/>
    </request>
    -<response>
      <representation mediaType="application/xml"/>
      <representation mediaType="application/json"/>
    </response>
   </method>
   -<method id="post" name="POST">
    -<request>
      <representation mediaType="application/xml"/>
      <representation mediaType="application/json"/>
    </request>
```

Fonte: Elaborada pelo autor.

Neste caso foi necessário registrar o serviço RESTful na lista de serviços da IDE NetBeans, a partir da URL do arquivo WADL, antes de acessar o assistente que gerou a classe Java para consumo do serviço.

Com o cliente RESTful funcionando, foram realizados os mesmos testes feitos com o *Web Service* e cujos resultados são descritos nos próximos parágrafos.

Na Figura 42 pode ser observada a requisição que o cliente RESTful faz ao serviço. Neste caso, o cliente RESTful não solicita o arquivo de descrição da aplicação antes de entrar

em operação como no *Web Service*, sendo requerido apenas os dados das entidades *sensors*, pelo método *GET* do protocolo HTTP. Salienta-se que *Sensores* foi o nome da pasta onde o serviço foi implantado no servidor GlassFish, compondo assim a URL do serviço RESTful. O atributo *Accept* do protocolo HTTP foi enviado com o parâmetro *application/xml*, identificando o tipo de estruturação de dados requerido, que também pode ser JSON.

Figura 42: Requisição HTTP do cliente RESTful.

```
GET /Sensores/resources/sensors HTTP/1.1
Accept: application/xml
User-Agent: Java/1.6.0_20
Host: localhost:44899
Connection: keep-alive
```

Fonte: Elaborada pelo autor.

Na Figura 43 é apresentada a resposta HTTP do serviço RESTful, em XML, para a aplicação cliente.

Figura 43: Resposta HTTP do serviço RESTful em XML.

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.0
Server: GlassFish Server Open Source Edition 3.0.1
Content-Type: application/xml
Content-Length: 589
Date: Thu, 23 Sep 2010 15:27:52 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sensors uri="http://localhost:44899/Sensores/resources/sensors">
  <sensor uri="http://localhost:44899/Sensores/resources/sensors/1/">
    <id>1</id>
      <tipo>Termômetro</tipo>
      <unidade>ºC</unidade>
      <valor>27.6500</valor>
    </sensor>
    <sensor uri="http://localhost:44899/Sensores/resources/sensors/2/">
     <id>2</id>
      <tipo>Manômetro</tipo>
      <unidade>psi</unidade>
      <valor>32.1000</valor>
    </sensor>
    ... trecho omitido ...
</sensors>
```

Fonte: Elaborada pelo autor.

Conforme pode ser observado na Figura 43, a resposta do serviço RESTful apresenta diretamente os dados dos sensores estruturados em XML, referentes à requisição do método GET com a URL …*/Sensores/resources/sensors*. Isto equivale ao método *findAll()* do *Web Service*, porém sem empregar o protocolo SOAP, apenas XML sobre HTTP.

Na Figura 44 apresenta-se a mesma requisição do método GET com a URL */Sensores/resources/sensors*, porém, para uma implementação em JSON do serviço RESTful. Como pode ser observado, a única diferença é a declaração *application/json* no lugar de *application/xml*, no atributo *Accept* da requisição.

Figura 44: Requisição HTTP/JSON do cliente RESTful.

```
GET /Sensores/resources/sensors HTTP/1.1
Accept: application/json
User-Agent: Java/1.6.0_20
Host: localhost:44899
Connection: keep-alive
```

Fonte: Elaborada pelo autor.

Apresenta-se na Figura 45 a resposta HTTP para a requisição JSON do cliente RESTful.

Figura 45: Resposta HTTP do serviço RESTful em JSON.

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.0
Server: GlassFish Server Open Source Edition 3.0.1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 23 Sep 2010 15:53:07 GMT

{"@uri":"http://localhost:44899/Sensores/resources/sensors",
   "sensor":[
   {"@uri":"http://localhost:44899/Sensores/resources/sensors/1/",
     "id":"1","tipo":"Termômetro","unidade":"ºC","valor":"27.6500"
   },
   {"@uri":"http://localhost:44899/Sensores/resources/sensors/2/",
     "id":"2","tipo":"Manômetro","unidade":"psi","valor":"32.1000"
   },
   … trecho omitido ...
   ]
}
```

Fonte: Elaborada pelo autor.

Como pode ser observado na Figura 45, a quantidade de caracteres utilizado para estruturar os dados em JSON é reduzida, para as mesmas informações dos sensores, se comparado com uma resposta em XML.

Excluindo-se o cabeçalho do protocolo HTTP das respostas retornadas ao cliente RESTful, a resposta em XML apresentou 607 caracteres para três registros da entidade *Sensor*, já a resposta em JSON, para os mesmos registros, contabilizou 477 caracteres, o que corresponde a uma redução de 21,4% na quantidade de dados trafegados na rede.

3.4.2 A plataforma embarcada

A plataforma de hardware empregada nos testes de implementação de SOA foi a NGW100 da Atmel, a mesma utilizada nas pesquisas do capítulo 2 e apresentadada na seção 2.2.4. A plataforma DE2 da Altera, com o *core* NIOS II, também era uma plataforma disponível e possível de ser empregada nesta pesquisa. No entanto, pelos argumentos já apresentados na seção 2.5, utilizou-se a NGW100.

Além da plataforma NGW100, aproveitou-se todo o cenário de desenvolvimento cruzado já instalado e configurado para esta plataforma na pesquisa anterior, conforme descrições apresentadas na seção 2.3.3.

3.4.3 Implementações sobre a NGW100

Inicialmente imaginou-se dois possíveis caminhos para implementar serviços sobre plataformas embarcadas. Adotar algum servidor de aplicações que suportasse serviços ou desenvolver, desde o início, uma aplicação que suportasse requisições HTTP e as interpretassem como serviço.

A utilização de servidores de aplicação, como o caso do servidor GlassFish, esbarra nas limitações de recursos de plataformas embarcadas, como discutido anteriormente. No desenvolvimento, desde o início, de uma aplicação que interprete requisições de serviços em padrões SOAP ou REST pode consumir alto dispêndio de esforços, visto o alto nível de tratamento de dados em alguns padrões, enquanto alguns projetos de softwares livre já implementam vários padrões.

A seguir são descritos alguns dos esforços em implementar serviços embarcados na plataforma NGW100 com o servidor Axix2/C e implementações com *socket*. No entanto, não

obteve-se pleno êxito, como atingido nas implementações sobre a plataforma PC, descritas na seção 3.4.1.

3.4.3.1 Servidor Axis2/C

O Apache Axis é um projeto de software livre, mantido pela Apache Software Foundation[17], que implementa SOAP segundo as especificações da W3C (APACHE, 2005). Esta aplicação em sua configuração mais popular opera como módulo do servidor de aplicações Apache Tomcat, que por sua vez necessita do JRE.

Assim, do mesmo modo como ocorre com o servidor de aplicações GlassFish, seriam muitos recursos para serem embarcados.

Em sua segunda versão o Apache Axis2 foi remodelado a partir do Apache Axis, para ser um servidor de aplicações com suporte a SOAP e REST (APACHE, 2011). O projeto Axis2 em sua versão mais popular, Axis2/Java, necessita do JRE para sua execução, embora também seja disponibilizado na versão Axis2/C, escrito na linguagem de programação C.

Assim, teoricamente, este servidor pode ser compilado para executar diretamente sobre o sistema operacional da plataforma alvo, sem a necessidade do JRE.

O código fonte do Apache Axis2/C em sua versão 1.6.0 foi obtido em *http://ftp.unicamp.br/pub/apache//ws/axis2/c/1_6_0/axis2c-src-1.6.0.tar.gz* e compilado em um primeiro momento para a plataforma PC, com as linhas de comando a seguir:

```
1: sudo tar -vzxf axis2c-src-1.6.0.tar.gz
2: cd  axis2c-src-1.6.0
3: export AXIS2C_HOME=/caminho/pasta_instalacao
4: sudo ./configure --prefix=${AXIS2C_HOME}
5: sudo make
6: sudo make install
7: cd /caminho/pasta_instalacao/bin
8: sudo ./axis2_http_server
```

A primeira linha de comando descomprime os arquivos fontes do servidor Axis2/C criando a pasta *axis2c-src-1.6.0*. A segunda linha acessa a pasta criada com os arquivos-fontes

17 Organização que provê suporte às comunidades de projetos Apache de código aberto, que desenvolvem softwares para o bem público.

do servidor. A terceira e quarta linhas de comando definem variáveis e configurações de ambiente. Na quinta linha de comando são compilados os fontes do servidor e na sexta é instalado o servidor Axis2/C no local indicado pela variável de ambiente *AXIS2C_HOME*. A sétima linha de comando acessa a pasta de instalação do servidor. Na última linha de comando o executável *axis2_http_server*, recém gerado pela compilação e instalado, inicia o servidor Axis2, por padrão na porta TCP 9090, que apresenta no console do sistema a mensagem "*Starded Simple Axis2 HTTP Server ...*".

A partir desse passo conseguiu-se que o servidor Axis2/C entrasse em operação. O próximo passo foi pesquisar como utilizá-lo para implantar serviços, sobre uma plataforma embarcada.

Antes de tentar compilar o servidor Axis2/C para a plataforma embarcada, foi realizada a análise dos recursos consumidos pela versão instalada na plataforma PC, a fim de verificar se a plataforma embarcada teria os recursos disponíveis para esse servidor.

A primeira constatação foi que os arquivos implantados no sistema de arquivos da plataforma PC totalizavam 21,7 MB, quantidade superior a toda a memória *flash* de 16 MB da plataforma embarcada, que implementa o sistema de arquivos. A solução adotada para esta limitação foi implantar os arquivos da instalação do servidor Axis2/C em um cartão externo de memória.

Inicialmente ocorreram problemas nas tentativas de instalar o servidor Axis2/C no cartão de memória. Verificou-se posteriormente que os problemas vinham do formato do sistema de arquivos que estava o cartão de memória, FAT32 (*File Allocation Table*), que não suporta *links* simbólicos do sistema de arquivos da plataforma embarcada. Com a formatação do cartão de memória no formato Ext3 (*Third Extended file system*) o problema foi solucionado.

Também foi analisada a quantidade de memória física e virtual consumida pelo servidor Axis2/C em execução na plataforma PC. Como a análise direta da memória alocada pelo processo *axis2_http_server* poderia não refletir a quantidade real de memória alocada para este servidor, dado que outros processos e *threads*[18] invocados pelo processo inicial poderiam consumir mais recursos, decidiu-se apenas estimar a quantidade de memória alocada, pela diferença da quantidade de memória física e virtual disponíveis antes e depois de iniciar-se tal processo. Pôde ser verificado que não ocorreu alocação de memória virtual e

18 Fluxo de processamento concorrente de um processo.

que a memória física alocada foi em torno de 2500 kB, quantidade razoável para a plataforma embarcada que possui memória RAM de 32 MB e que no momento desta análise, ainda sem o servidor Axis2/C instalado, apresentava cerca de 18 MB livres de memória física.

Para compilar o servidor Axis2/C para a plataforma embarcada alvo foi utilizado o compilador *avr32-linux-gcc* instalado no ambiente de desenvolvimento cruzado descrito anteriormente. Para tanto utilizou-se os mesmos procedimentos descritos para a plataforma PC, exceto na quarta linha de comando, que foi substituída pela linha de comando a seguir:

```
sudo ./configure --prefix=${AXIS2C_HOME} --host=avr32-linux
```

O parâmetro *host* adiciona o prefixo *avr32-linux* na chamada do compilador *gcc*, fazendo com que o compilador chamado pelo *Makefile* seja o *avr32-linux-gcc*.

Os arquivos instalados pelo *make install* na pasta */caminho/pasta_instalacao/* foram copiados para o cartão de memória que foi montado no sistema de arquivos da plataforma embarcada, onde ainda foram executadas as linhas de comando:

```
1: export AXIS2C_HOME=/caminho/pasta_cartao/pasta_instalacao
2: cd /caminho/pasta_cartao/pasta_instalacao/bin
3: ./axis2_http_server
```

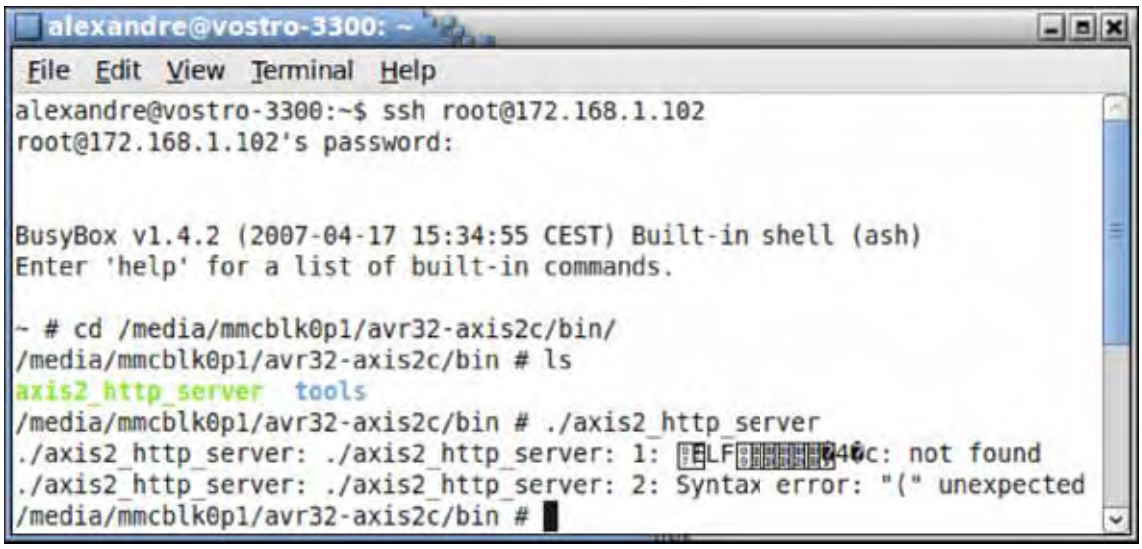
A primeira linha de comando cria a variável de ambiente *AXIS2C_HOME* na plataforma embarcada, apontando para o caminho da instalação do Axis2/C, que agora está dentro da pasta em que o cartão de memória foi montado. Na segunda linha de comando é acessado a pasta *bin* no cartão de memória e na terceira linha é executado o arquivo *axis2_http_server*, para iniciar o servidor.

Neste ponto era de se esperar a mensagem "*Starded Simple Axis2 HTTP Server ...*" do console, no entanto, o retorno foi a indescritível mensagem que pode ser observada na Figura 46. Apenas a indicação de não encontrado não permite esclarecer o que não foi encontrado.

Uma hipótese para o erro apresentado na execução do servidor Axis2/C, sobre a plataforma embarcada, é a de que ele invoque algum recurso em software que não foi instalado no sistema operacional. No entanto, no Guia de Instalação do Apache Axis2/C (APACHE, 2009) é enfática a afirmação de que "Por padrão Axis2/C não depende de

nenhuma biblioteca de software.”. Desta forma, a exceção fica por conta de alguns recursos que utilizam algumas bibliotecas adicionais. Salienta-se que nenhuma modificação foi realizada no pacote padrão dos arquivos fontes do Axis2/C, tão pouco se tentou invocar algum recurso no servidor, apenas tentou-se iniciá-lo.

Figura 46: Mensagem de erro ao tentar iniciar o servidor Axis2/C.



Fonte: Elaborada pelo autor.

3.4.3.2 Socket C

No segundo caminho adotado para se implementar serviços sobre a plataforma embarcada NGW100, foi desenvolver na linguagem C uma aplicação que escutasse requisições HTTP na porta TCP 8000. Para tanto foram utilizadas funções como a *socket()*, *listen()* e *accept()* da biblioteca *sys/socket.h*.

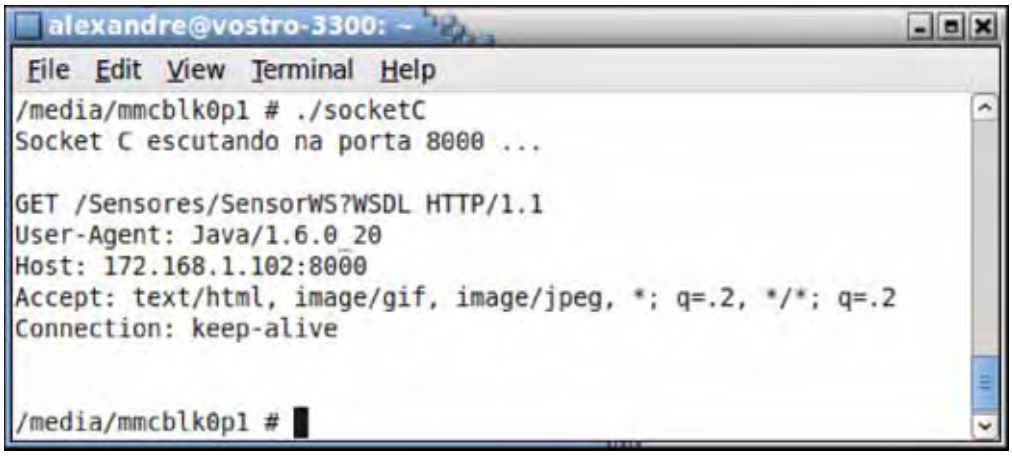
Para compilar o arquivo fonte *socketC.c* da aplicação desenvolvida e gerar o arquivo executável *socketC*, utilizou-se o compilador *avr32-linux-gcc* no ambiente de desenvolvimento cruzado, com a linha de comando a seguir:

```
avr32-linux-gcc socketC.c -o socketC
```

O executável *socketC* foi transferido para o cartão de memória montado na plataforma embarcada, de onde foi executado, conforme apresentado na Figura 47. A aplicação *socketC* foi programada para escutar na porta TCP 8000, até que recebesse uma requisição qualquer

nesta porta, quando simplesmente a repassa para o console do sistema, sem nenhum tipo de tratamento da requisição.

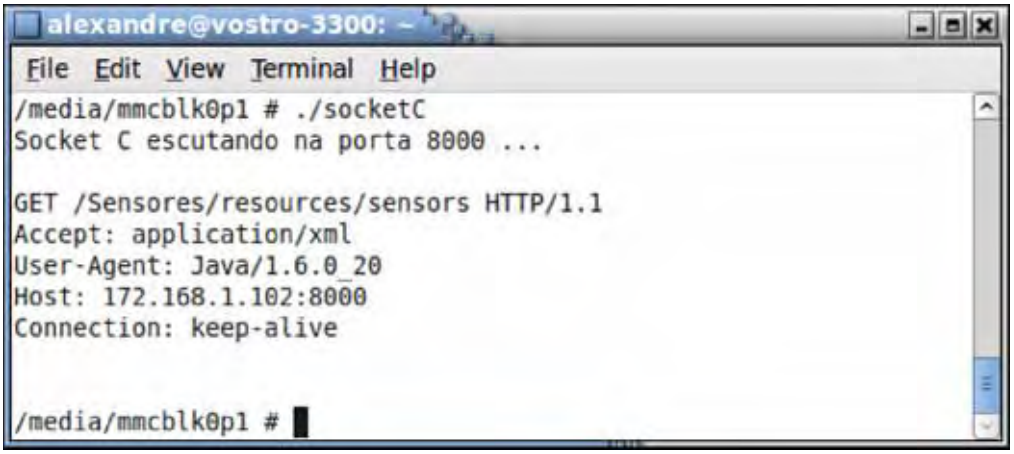
Figura 47: Execução da aplicação s*ocketC* com requisição do cliente do *Web Service*.



Fonte: Elaborada pelo autor.

Na Figura 48 pode ser observada a requisição HTTP realizada para o IP e porta TCP da aplicação *socketC*. A requisição HTTP foi enviada pelo mesmo cliente do *Web Service* desenvolvido anteriormente. O conteúdo enviado também foi o mesmo que para o *Web Service*, no entanto, nenhum retorno à requisição foi devolvido ao cliente, como teria ocorrido se a requisição realmente tivesse sido enviada para o *Web Service*.

Figura 48: Execução da aplicação *socketC* com requisição do cliente RESTful.



Fonte: Elaborada pelo autor.

O mesmo teste de requisição foi realizado com o cliente RESTful, como apresentado na Figura 48. Mais uma vez, a requisição da aplicação cliente foi a mesma recebida pelo

serviço RESTful desenvolvido na plataforma PC.

Pelo método GET do protocolo HTTP foi requerido dados das entidades *sensors*. *Sensores* é o nome da pasta onde o serviço foi implantado na plataforma PC, compondo a URL do serviço RESTful. O atributo *Accept* do protocolo HTTP foi enviado com o parâmetro *application/xml*, identificando o tipo de estruturação de dados requerido.

3.4.3.3 Máquina Virtual Java

Na busca de outras possibilidades para a implementação de serviços, além das duas já relatadas até o momento, buscou-se instalar uma Máquina Virtual Java (JVM – *Java Virtual Machine*) na plataforma embarcada NGW100, para que esta linguagem de programação viesse a fazer parte das ferramentas disponíveis nesta plataforma.

Embora a Oracle forneça gratuitamente seu JRE, ele é distribuído já compilado para plataformas específicas e ainda exige muitos recursos de hardware. Estas características inviabilizam a instalação do JRE da Oracle em muitas plataformas embarcadas, como a NGW100.

Após algumas análises das opções disponíveis, decidiu-se em adotar a GNU Classpath e a JamVM, para compor um JRE para a plataforma NGW100.

A GNU Classpath é um projeto da Free Software Foundation[19] para criar uma biblioteca livre de implementações de classes para máquinas virtuais e compiladores, para a linguagem de programação Java (FSF, 2009).

A JamVM é uma JVM de código aberto, escrita em C e muito pequena se comparada com JVM's comerciais (LOUGHER, 2010). Foi projetada para usar a GNU Classpath e suportar completamente as especificações da linguagem Java.

Embora desejado, não se pode esperar tal feito da JamVM, pois a própria GNU Classpath da qual ela depende, ainda não atingiu o êxito esperado.

Em um primeiro momento foram utilizados os binários já compilados para a arquitetura AVR32, tanto da GNU Classpath 0.98, quanto da JamVM 1.5.0, disponíveis em *http://avr32linux.org/twiki/pub/Main/JamVM/usr_local_classpath.tgz* e *http://avr32linux.org/twiki/pub/Main/JamVM/usr_local_jamvm.tgz***,** respectivamente. Para instalá-los na plataforma embarcada os pacotes baixados foram copiados para o cartão de memória removível e, ainda

19 Organização sem fins lucrativos com missão de promover a liberdade do usuário de computador e defender os direitos de todos os usuários de software livre.

na plataforma PC, descomprimidos com as linhas de comandos a seguir:

```
1: sudo tar -vzxf usr_local_classpath.tgz
2: sudo tar -vzxf usr_local_jamvm.tgz
```

A primeira linha de comando descomprime a biblioteca de classes GNU Classpath, criando a pasta *usr/local/classpath* no cartão de memória. A segunda linha de comando descomprime os arquivos binários da JamVM compilados para a arquitetura AVR32, criando a pasta *usr/local/jamvm* no cartão de memória.

Com o cartão de memória montado na plataforma embarcada, a GNU Classpath e a JamVM necessitavam serem instaladas em */usr/local/classpath* e */usr/local/jamvm*, respectivamente, que são as localizações padrões definidas na compilação. No entanto, para manter os binários no cartão de memória, utilizou-se novamente o recurso de *link* simbólico do sistema de arquivos, com as linhas de comando a seguir:

```
1: ln -s /cartao_mem/usr/local/classpath /usr/local/classpath
2: ln -s /cartao_mem/usr/local/jamvm /usr/local/jamvm
3: export PATH=$PATH:/usr/local/jamvm/bin
```

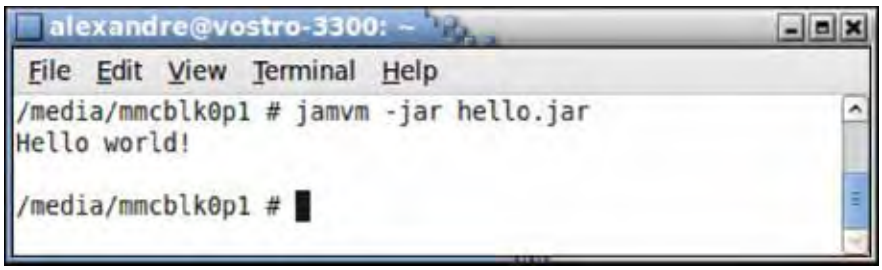
As duas primeiras linhas de comando criam os *links* simbólicos em */usr/local* para as pastas reais no cartão de memória e a última linha adiciona ao *PATH* do sistema a pasta */bin* da instalação da JamVM. Desta forma, tem-se o JRE instalado na plataforma embarcada, respondendo pelo comando *jamvm* em vez do clássico comando *java*. Caso desejado, o comando *java* pode ser opcionalmente adicionado ao sistema, por meio de *link* simbólico ao comando *jamvm*.

Para testar a instalação do JRE foi criado e compilado na plataforma PC um clássico *Hello World*. O arquivo *hello.jar* gerado, foi transferido para a plataforma embarcada e executado com a linha de comando a seguir:

```
jamvm -jar hello.jar
```

Na Figura 49 apresenta-se a execução desta implementação em Java, onde pode ser observado o seu êxito.

Figura 49: Execução da aplicação *hello.jar*.

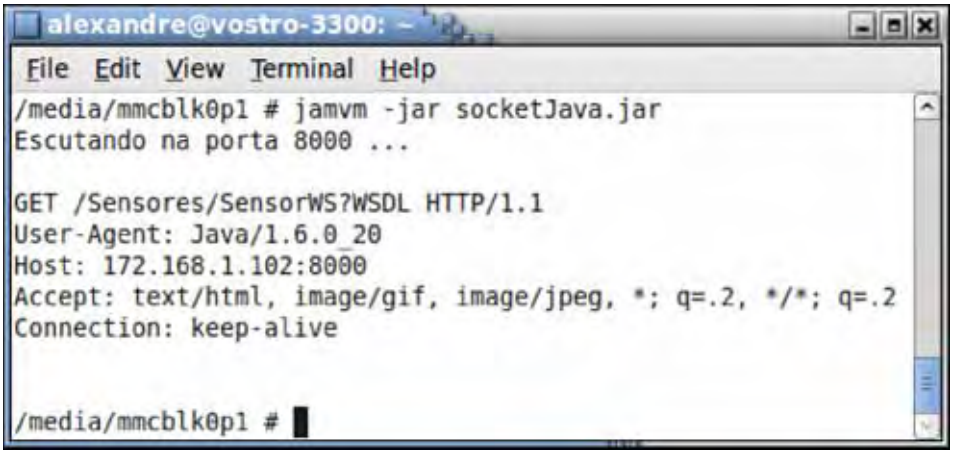


Fonte: Elaborada pelo autor.

3.4.3.4 Socket Java

A mesma aplicação de escuta na porta TCP 8000, também foi desenvolvida em linguagem Java, utilizando a classe *Socket* do pacote *java.net*. Esta implementação foi desenvolvida e compilada em plataforma PC, onde se gerou o pacote *socketJava.jar*, que foi transferido para o cartão de memória e executado na plataforma embarcada, como pode ser observado nas Figuras 50 e 51.

Figura 50: Execução da aplicação s*ocketJava* com requisição do cliente *Web Service.*

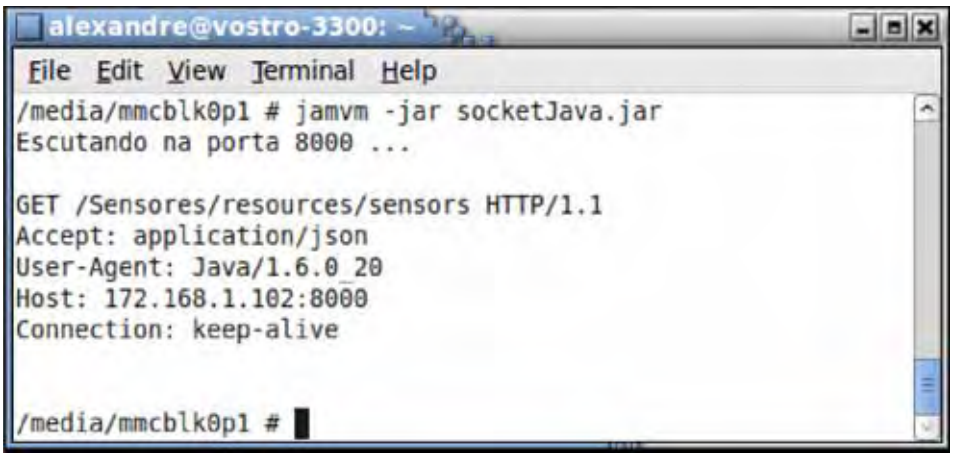


Fonte: Elaborada pelo autor.

Na Figura 50 apresenta-se a requisição HTTP do mesmo cliente do *Web Service* utilizado com a aplicação *socketC* e na Figura 51 apresenta-se a requisição HTTP realizada pelo cliente RESTful.

O funcionamento, como era de se esperar, foi o mesmo da aplicação desenvolvida em C, no entanto, nesta aplicação Java notou-se significativa demora para a aplicação iniciar a escuta, o que não ocorreu com a aplicação em C.

Figura 51: Execução da aplicação s*ocketJava* com requisição do cliente RESTful.



Fonte: Elaborada pelo autor.

Tentou-se também com a aplicação *socketJava*, acondicionar em uma *thread* o processo de escuta do s*ocket*, o que ocasionou erro de alocação de recursos pela máquina virtual, no entanto, como a mesma tentativa não foi realizada em C, surgiu a dúvida se o uso de *threads* é uma limitação de recursos da máquina virtual, ou do próprio sistema operacional embarcado.

3.5 CONCLUSÕES

Embora o erro apresentado na execução do servidor Axis2/C sobre a plataforma embarcada NGW100 impeça sua adoção até o momento, esta opção ainda merece outras investigações.

Uma pesquisa mais aprofundada dos arquivos fontes deste servidor pode não só levar a solução do problema, mas também a melhor compreensão dos mecanismos e recursos deste servidor, permitindo possivelmente que seja criada uma versão mais enxuta do mesmo, específica para o propósito de embarcar em plataformas restritas.

Com a implementação e testes da aplicação *socketC*, pôde-se iniciar o segundo caminho para o desenvolvimento de uma aplicação completa, com suporte a requisições em nível de serviços. Embora o recebimento de requisições HTTP por *socket* TCP tenha sido um grande passo, muito ainda deve ser feito se realmente esta for a escolha, ou a única opção a se seguir.

O JRE instalado na plataforma embarcada expandiu o conjunto de ferramentas disponíveis para desenvolvimento, permitindo a adoção da linguagem Java e muito do legado

disponível para esta tecnologia. Embora não mensurado, com a construção da aplicação *hello*, teve-se a impressão de que a execução da aplicação ficou significativamente mais lenta do que a mesma funcionalidade desenvolvida e compilada em C.

Com o desenvolvimento da aplicação *socketJava* a percepção de lentidão foi acentuada, embora pôde-se identificar que a demora ocorresse principalmente na inicialização da aplicação e não exatamente na execução da mesma. De qualquer forma, essa demora foi apenas uma percepção, que deve ser melhor investigada e aferida, antes de julgar a viabilidade computacional de implementações em Java, para a plataforma embarcada em questão.

3.6 PROPOSTAS DE TRABALHOS FUTUROS

Ao término dos estudos apresentadas neste capítulo, algumas dúvidas e motivações sugerem frentes de pesquisas futuras, como as que seguem:

- Investigação do erro ocorrido na execução embarcada do servidor Axis2/C;
- Pesquisa dos códigos fontes do servidor Axis2/C, para que o mesmo possa ser otimizado para plataformas restritas;
- Desenvolvimento de métodos para tratamento de requisições HTTP e SOAP, a partir de *sockets*;
- Manipulação de dados em XML e JSON;
- Medições e comparações de desempenho de aplicações desenvolvidas em C e em Java, sobre plataforma embarcada;
- Pesquisa de outras soluções já implementadas de *Web Services* em plataformas embarcadas.

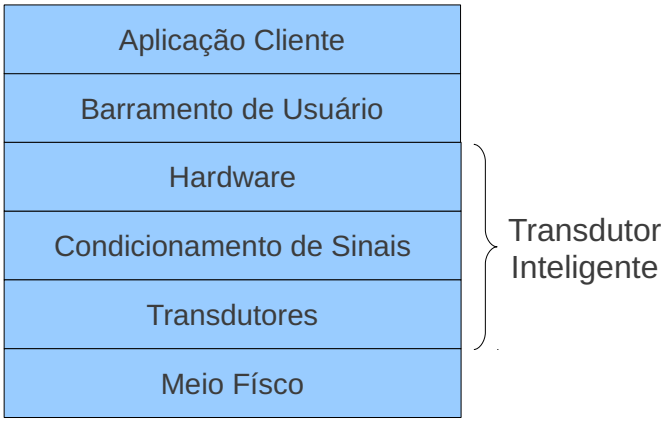
CAPÍTULO 4

TECNOLOGIAS E LINGUAGENS PARA A APLICAÇÃO CLIENTE

4.1 INTRODUÇÃO

A aplicação cliente é a última camada para a publicação de informações provenientes do meio físico na arquitetura proposta. Na Figura 52 ilustra-se essas camadas.

Figura 52: Camadas entre meio físico e aplicação cliente.



Fonte: Elaborada pelo autor.

As informações do meio físico são obtidas ou manipuladas por meio de transdutores que transformam as grandezas físicas em sinais elétricos e vice-versa. Para que os sinais elétricos sejam levados aos níveis de operação da camada de hardware é necessário condicioná-los e digitalizá-los, o que é feito pela camada de condicionamento de sinais.

Na camada de hardware podem ser empregadas plataformas como as estudadas no capítulo 2. Nesta camada é feito o processamento necessário para os sinais provenientes da camada inferior, até serem dispostos no barramento de usuário.

Pretende-se adotar a pilha de protocolos TCP/IP e padrões de SOA para o barramento de usuários, conforme apresentado no capítulo 2 e capítulo 3, respectivamente. Assim, caberá à aplicação cliente acessar os dados no barramento de usuários, como serviços de uma SOA, sobre TCP/IP.

Neste contexto, investiga-se neste capítulo as linguagens de programação, tecnologias e recursos para implementação da aplicação cliente.

Cabe enfatizar que as três camadas entre o meio físico e o barramento de usuários podem ser compreendidas como uma única camada, implementada por um transdutor inteligente, conforme identificado na Figura 52.

4.1.1 Objetivos específicos

O objetivo deste capítulo foi investigar algumas tecnologias, linguagens de programação e recursos de interface gráfica e comunicação distribuída, que pudessem servir para o desenvolvimento de aplicações clientes para arquitetura proposta. Além disso, foi necessário definir uma tecnologia e linguagem de programação à ser adotada no desenvolvimento das aplicações clientes.

4.1.2 Justificativas

Não se pretendeu nesta investigação responder a pergunta sobre qual é a melhor tecnologia ou linguagem de programação. Como é conhecido, tudo depende das necessidades da aplicação e mesmo assim são encontradas algumas tecnologias e linguagens de programação para o mesmo propósito. Embora a pergunta não possa ser respondida, foi necessário definir uma tecnologia e linguagem de programação à ser adotada, dentre as disponíveis.

4.1.3 Trabalhos relacionados

Kato (2010), da Universidade Musashi de Tokyo, utilizou JavaFX para desenvolver um ambiente de programação visual, denominado Splish, para plataformas de hardware da família Arduino, baseadas em microcontroladores de arquitetura AVR da Atmel. Este ambiente permite o desenvolvimento de aplicações para os microcontroladores, a partir de especificações compostas por blocos gráficos do ambiente, em substituição às tradicionais linguagens textuais e procedurais de programação.

Liu Y., Liu X-F. e Mao (2010), da Universidade Shanghai, utilizaram o Microsoft Siverlight para desenvolver um WebGIS (*Web Geographic Information System*) baseado na arquitetura REST. O WebGIS é uma aplicação RIA (*Rich Internet Application*) que se

comunica com outras aplicações no padrão REST. Os autores relatam as significativas vantagens de interação usuário-aplicação com RIA, em substituição às tradicionais páginas HTML que eram adotadas em WebGIS anteriores.

Viegas, Pereira e Girão (2007), da Universidade Técnica de Lisboa, implementaram o padrão IEEE 1451.1 utilizando o .NET Framework da Microsoft. Neste trabalho, além de enfatizarem as vantagens em utilizar um *framework* comercial, eles propuseram melhorias no padrão IEEE 1451.1, para que este padrão suportasse requisições de *Web Services*.

Lammarsch et al. (2008), da Danube University Krems, realizaram um pesquisa comparativa entre diversas tecnologias para visualização interativa em aplicações Web. Dentre as tecnologias comparadas estão: Java Applets, Flash e Silverlight, no lado cliente, e .NET e Java, no lado servidor. Os autores não apontam nenhuma tecnologia como sendo absolutamente melhor, mas relatam características e situações específicas em que uma ou outra tecnologia pode prevalecer dentre as demais.

4.1.4 Organização do capítulo

Neste capítulo é apresentada uma breve introdução sobre o posicionamento da aplicação cliente na arquitetura proposta, os objetivos específicos e justificativas das pesquisas realizadas.

Na seção 4.2 elenca-se uma série de requisitos desejáveis na tecnologia e linguagem de programação a ser adotada para desenvolvimento da aplicação cliente.

Na seção 4.3 são relacionadas algumas das possíveis tecnologias e linguagens de programação à serem adotadas neste trabalho.

Na seção 4.4 são descritos alguns testes e mecanismos de desenvolvimento em JavaFX, para a implementação de uma aplicação cliente.

Na seção 4.5 são relatadas as conclusões deste capítulo.

Na seção 4.6 são elencadas algumas propostas de trabalhos futuros.

4.2 REQUISITOS PARA A TECNOLOGIA E LINGUAGEM A SER ADOTADA

Os recursos dos transdutores inteligentes são publicados no barramento de usuários como serviços. Em tese qualquer tecnologia e linguagem de programação podem ser adotadas, desde que suporte comunicação em TCP ou HTTP.

Adicionalmente, pode-se elencar outros requisitos que embora não sejam imprescindíveis, são desejáveis:

- comunicação distribuída – além dos requisitos de comunicação em TCP ou HTTP, tecnologias que suportam comunicação distribuída oferecem outros recursos de comunicação por troca de mensagens, tanto com recursos de comunicação em outros protocolos, como no oferecimento de *frameworks* para automatizar integrações, além de *parsers* e recursos de fluxo de processamento concorrente;
- *parsers* – permitem extrair informações especificadas em alguma linguagem formal como C, Java, HTML, XML etc. Neste trabalho é necessário o emprego de *parsers* para tratar especificamente os retornos em XML ou JSON dos transdutores inteligentes;
- *threads* – gerenciam fluxos de processamento concorrente na máquina cliente, que são necessários para o tratamento assíncrono de requisições, para que a interface com o usuário não fique bloqueada até que se obtenha o retorno na requisição;
- interface gráfica rica – para que os dados possam ser apresentados de forma mais conveniente e intuitiva, é desejado recursos gráficos ricos para a confecção de gráficos e interface interativa com o usuário;
- interface Web – embora as interfaces gráficas *desktop* em geral ofereçam melhor domínio e usabilidade dos recursos, a publicação desta interface na Web é mais condizente à proposta de universalização e difusão do acesso aos transdutores inteligentes;
- interface Móvel – com a infinidade de dispositivos móveis com acesso à Web, tecnologias que se comprometem com o dinamismo de interfaces existentes, também é conveniente para a proposta de desenvolvimento;
- ferramentas livres – embora a integração como serviço é naturalmente aberta, pelo padronização dos protocolos adotados, o desenvolvimento de aplicações clientes pode ser realizado com tecnologias e ferramentas proprietárias; no entanto é desejável o emprego de ferramentas livres, baseadas em tecnologias livres ou gratuitas;
- comunidade ativa – existe uma infinidade de ferramentas, *frameworks* e tecnologias para desenvolvimento de aplicações em software. Uma boa diretriz à analisar antes de se aventurar por todas as opções encontradas é a pesquisa por comunidades ativas da ferramenta ou tecnologia, pois mesmo que ocorram alterações na ferramenta ou

tecnologia, uma comunidade ativa contribui significativamente na resolução de problemas e dúvidas que possam surgir.

4.3 TECNOLOGIAS PARA DESENVOLVIMENTO DE INTERFACES

Frente aos requisitos desejáveis elencados anteriormente, pensava-se na solução clássica de páginas Web com HTML na estrutura principal, JavaScript[20] para proporcionar interatividade no lado cliente, com técnicas de AJAX[21] (*Asynchronous Javascript and XML*), bibliotecas como a JQuery[22] e alguma linguagem de *script* livre no lado servidor. No entanto, outras tecnologias para RIAs foram analisadas, como segue:

- Adobe AIR (*Adobe Integrated Runtime*) e Flex – Adobe AIR é uma tecnologia RIA com *runtime*[23] gratuito (*freeware*) que permite implantar aplicações *standalone* desenvolvidas em HTML, JavaScript, ActionScript, Flex e Flash em várias plataformas e dispositivos, incluindo Android, BlackBerry OS, os dispositivos iOS, computadores pessoais e televisões (ADOBE, 2011a). O Flex é um *framework* livre da Adobe que permite criar aplicações móveis para iOS, Android, BlackBerry OS, assim como aplicações tradicionais *desktop* e Web, usando o mesmo modelo de programação, ferramentas e códigos-fonte (ADOBE, 2011b), no entanto, a principal ferramenta de desenvolvimento, o IDE Adobe Flash Builder, é proprietária;
- Microsoft Silverlight – Microsoft Silverlight é uma tecnologia RIA com *runtime* gratuito da plataforma .NET Framework, baseado no WPF (*Windows Presentation Foundation*), compatível com múltiplos navegadores, dispositivos e sistemas operacionais. Permite alto nível de interatividade em ambientes onde funcione a Web (MICROSOFT, 2011b). A principal ferramenta de desenvolvimento para esta tecnologia é o IDE Microsoft Visual Studio, que assim como o Adobe Flash Builder, também é proprietária. Embora essa tecnologia se proponha ser multi-plataforma, o .NET Framework não é disponibilizado pela Microsoft para sistemas Linux e Unix, o que impede sua execução nestes sistemas operacionais, exceto se forem utilizadas

20 Popular linguagem de *script* que executa no lado cliente (no navegador).
21 Método de troca assíncrona de dados entre o lado cliente e o lado servidor que permite maior interatividade da interface de usuário e economia de banda após o carregamento da aplicação.
22 Popular biblioteca JavaScript que simplifica a implementação de interfaces e o desenvolvimento de aplicações AJAX.
23 Aplicação que interpreta em tempo de execução os códigos da linguagem, compilados ou não, para as instruções do sistema operacional hospedeiro.

implementações alternativas do *framework*;

- JavaFX – JavaFX é uma tecnologia RIA da plataforma Java, que necessita de um *runtime* adicional ao JRE, ambos disponibilizados gratuitamente (ORACLE, 2011b). Até a versão 1.3 JavaFX suportava a linguagem JavaFX Script que podia interagir com camadas na linguagem Java nativa, no entanto, a linguagem JavaFX Script foi descontinuada na recente versão 2.0, que passou a adotar apenas a linguagem Java. Assim como as tecnologias anteriores JavaFX também se propõe a ser multi-dispositivo, multi-interface e multi-plataforma. Esta última característica já reconhecida e herdada da tecnologia Java, embora até o momento que este texto era escrito, a Oracle só disponibilizava *runtime* JavaFX 2.0 para a plataforma Windows. Na seção 6.2.2, onde é apresentado o *roadmap* da tecnologia JavaFX 2, esta análise é retomada. Para desenvolvimento em JavaFX podem ser utilizados IDEs livres e populares na comunidade Java, como o Eclipse e o Netbeans.

Diante das opções, necessidades e requisitos, não foi trivial decidir sobre que tecnologia adotar para desenvolvimento da aplicação cliente. Dentre as tecnologias analisadas, os produtos da Adobe pareceram ser os mais maduros e populares no desenvolvimento de RIAs, assim como a existência de relevante comunidade de desenvolvedores. No entanto, diante do acesso limitado aos IDEs proprietários, ficou-se na dúvida entre a clássica e madura solução Web com HTML, JavaScript etc., ou a embrionária e promissora tecnologia JavaFX.

Quanto à comunidade, as duas tecnologias (madura e embrionária) são consolidadas, enquanto a primeira data do próprio surgimento da Web, a segunda tem toda a comunidade Java como potenciais colaboradores. Pesou na decisão a possibilidade de investigar e conhecer o novo, visto que a primeira opção já era razoavelmente dominada pelo autor. Além disso, ainda, pareceu relevante a possibilidade de integração nativa com Java, mais especificamente com os recursos do Java EE (*Enterprise Edition*), o que levou a optar-se pela tecnologia JavaFX.

4.4 PROCEDIMENTOS EXPERIMENTAIS

Para o desenvolvimento da aplicação cliente foi adotado o IDE Netbeans 6.9.1, com JavaFX 1.3 e a linguagem JavaFX Script, sobre o sistema operacional Ubuntu. Durante o

desenvolvimento e testes da aplicação cliente foi lançada a versão JavaFX 2.0, que descontinuou a linguagem JavaFX Script, até então presente em JavaFX 1.3. No entanto, até o momento em que este texto era escrito, a Oracle só disponibilizava o *runtime* JavaFX 2.0 para sistemas operacionais Windows. Como boa parte do desenvolvimento e testes pretendidos para esta pesquisa já haviam sidos desenvolvidos, decidiu-se concluí-la em JavaFX Script, mesmo sabendo que os códigos poderiam não ser portáveis para a próxima versão de JavaFX.

4.4.1 A interface

No desenvolvimento em JavaFX iniciou-se pela interface de usuário, onde foram realizados testes de execução tanto na Web como em *desktop*. Ferramentas da JavaFX *Production Suite* permitem a entrada de artes gráficas especificadas em SVG (*Scalable Vector Graphics*) (CLARKE; CONNORS; BRUNO, 2010), que poderiam ser desenvolvidas em ferramentas específicas de arte gráfica. No entanto, com o intuito de apenas de se familiarizar com os elementos de interface, desenvolveu-se uma interface gráfica a partir dos elementos primários disponibilizados pela linguagem JavaFX.

A classe *javafx.stage.Stage* é o principal container para uma exibição em JavaFX, uma instância desta classe produz uma janela nos sistemas *desktop*, um *frame* em páginas Web e telas em dispositivos móveis. Esta classe é a mesma para qualquer dispositivo ou ambiente e pode ser manipulada para cada sistema, conforme as necessidades.

A classe *javafx.scene.Scene* é o segundo elemento de uma interface JavaFX. Em JavaFX todos os elementos gráficos da interface são alocados na classe *Stage* em uma árvore de objetos, sendo *Scene* os filhos de primeiro grau desta árvore que podem receber diversos objetos gráficos em suas folhas.

A linguagem JavaFX Script apresenta classes que implementam diversas primitivas geométricas que podem ser inseridas na árvore de objetos gráficos e manipuladas vetorialmente. Existem também classes de elementos típicos de controle de interfaces, como *buttons* e *inputs*. Há também a possibilidade de inserir imagens matriciais como objetos desta árvores e também a possibilidade de criação de novos objetos, com personalização em nível de *pixels*, tornando infinita as possibilidades de desenvolvimento gráfico da interface.

Na Figura 53 podem ser observados os diversos tipos de objetos gráficos descritos anteriormente. As figuras geométricas foram produzidas vetorialmente com as classes primitivas *Circle*, *Rectangle* e *Line*. Observa-se que o círculo foi personalizado manipulando

alguns *pixels* em seu interior. O logo de JavaFX foi inserido como uma imagem matricial em uma instância da classe *Image*, em um container da classe *ImageView*. O botão de controle é uma instância da classe *Button* e o texto “Texto” uma instância da classe *Text*. Todos estes objetos são folhas de uma instância da classe *Scene*, que por sua vez é filha de uma instância da classe *Stage*, que é raiz desta árvore de objetos gráficos.

Figura 53: Interface gráfica.

Fonte: Elaborada pelo autor.

Nesta árvore de objetos, cada elemento pode ser alterado e reposicionado dinamicamente. Também é possível manipular a camada de apresentação, o que permite a manipulação dos elementos no eixo Z, sendo X e Y o posicionamento horizontal e vertical, respectivamente. Na Figura 53 pode ser observado que o retângulo está em uma camada intermediária entre o círculo e o segmento de reta.

As classes de elementos gráficos já disponíveis pela linguagem apresentam tratamento de eventos de entrada de usuário, como os de mouse e teclado. No entanto, se for necessário a criação de um novo objeto personalizado, deve-se herdar pelo menos a classe base *javafx.scene.Node*, que implementa recursos básicos, como o tratamento de eventos de entrada do usuário, ou qualquer outra classe de elementos gráficos já disponíveis.

Os testes realizados não esgotaram os recursos disponíveis em JavaFX, mas foram suficientes para demonstrar o potencial das interfaces desta tecnologia. A partir destes recursos básicos é possível imaginar uma infinidade de possibilidades, suficientes para o desenvolvimento de interfaces para as aplicações clientes na arquitetura proposta.

Para o trabalho de pesquisa em questão, pretende-se representar graficamente informações de transdutores distribuídos, permitindo a interação com o usuário e integração com outros transdutores e recursos, disponibilizados como serviços.

4.4.2 Comunicação

Para a comunicação e integração de transdutores e recursos, disponibilizados como serviços, é necessário um canal de comunicação. Observa-se que a possibilidade de JavaFX acessar os recursos de Java EE, potencializa significativamente a capacidade de comunicação distribuída. No entanto, para as necessidades desta pesquisa, foi suficiente utilizar a classe nativa *javafx.io.http*. *HttpRequest* de JavaFX, para estabelecer conexões HTTP com os transdutores inteligentes, respondendo em nível de serviços.

Para utilizar a classe *HttpRequest* foi construída a classe *HTTPConnection* estendendo a classe *HttpRequest*. Como pode ser observado no código apresentado na Figura 54, a conexão HTTP é estabelecida quando invocado o método *start()* da instância *httpConnection*. Antes, porém, é necessário atribuir a URL do transdutor inteligente, ou do serviço que deseja-se acessar.

Figura 54: Instância e estabelecimento da conexão HTTP.

```
httpConnection = HttpConnection {};
httpConnection.location = textBoxUrl.text;
httpConnection.start();
```

Fonte: Elaborada pelo autor.

Por padrão, uma conexão disparada pela classe *HttpRequest* adota o método GET do protocolo HTTP, assim, os parâmetros necessários para o transdutor inteligente foram concatenados junto ao endereço contido em *textBoxUrl.text* para composição da requisição GET.

O retorno assíncrono da requisição HTTP foi tratado no método *onInput()*, herdado de *HttpRequest*, que invoca o método *processResult()*, implementado na aplicação principal, para processamento dos dados retornados, conforme pode ser observado no trecho de código apresentado na Figura 55.

Na Figura 56 podem ser observados os dados retornados por um transdutor inteligente RESTful, respondendo em XML à aplicação cliente. Observa-se que este retorno não apresentou o cabeçalho do protocolo HTTP como, por exemplo, no retorno do serviço RESTful apresentado na Figura 43. Isto se deve ao fato de que a conexão estabelecida em HTTP foi tratada pelo objeto da classe *HTTPConnection*, abstraindo do programador detalhes

do protocolo HTTP, o que não aconteceria se a conexão fosse realizada em uma camada de abstração mais baixa, como em TCP.

Figura 55: Retorno assíncrono da conexão HTTP à aplicação principal.

```
public override var onInput = function(input: InputStream) {
   try {
      Main.processResult(input); //processa o retorno
   } finally {                   //assíncrono
      input.close();
   }
}
```

Fonte: Elaborada pelo autor.

Figura 56: Resposta HTTP do serviço RESTful em XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<smartTransducer>
 <id>1093</id>
 <thermometer>
  <id>1</id>
  <temperature>21</temperature>
  <unit>C</unit>
  <calibration>
   <offset>0</offset>
   <slope>1</slope>
  </calibration>
 </thermometer>
 <compressor>
  <id>2</id>
  <power>on</power>
 </compressor>
</smartTransducer>
```

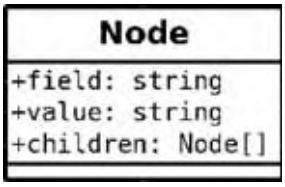
Fonte: Elaborada pelo autor.

4.4.3 *Parser* e a árvore de objetos *Node*

Como apresentado na Figura 56, o retorno do transdutor inteligente RESTful foi em XML. Poderia também ter sido em JSON. Para tratamento destes dados retornados foi utilizada a classe nativa *javafx.data.pull.PullParser* de JavaFX, para implementar um *parser*. A classe *PullParser* possui métodos de tratamento de dados tanto em XML como em JSON.

Para implementar o *parser* na aplicação cliente foi construída uma classe *Parser* estendendo a classe *PullParser*. Os dados extraídos pelo *parser* implementado foram organizados em uma árvore de objetos da classe *Node*, apresentada na Figura 57.

Figura 57: Classe *Node*.

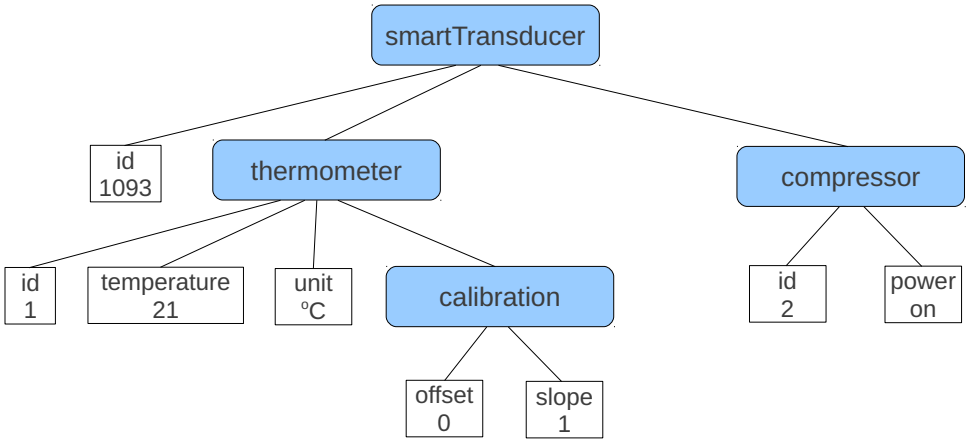


Fonte: Elaborada pelo autor.

Com objetos da classe *Node*, qualquer elemento de uma estrutura XML ou JSON pode ser mapeado em um nó ou folha da árvore de objetos. O atributo *field* recebe o nome do nó ou dado, o atributo *value* seu valor e o atributo *children* é um vetor para os próximos nós da estrutura. As folhas da árvore não possuem atribuições para *children* e os nós intermediários não possuem atribuições para *value*.

Na Figura 58 apresenta-se um diagrama ilustrando a árvore de objetos *Node* formada pelos dados apresentados na Figura 56, retornados por um transdutor inteligente RESTful.

Figura 58: Diagrama de uma árvore de objetos *Node*.

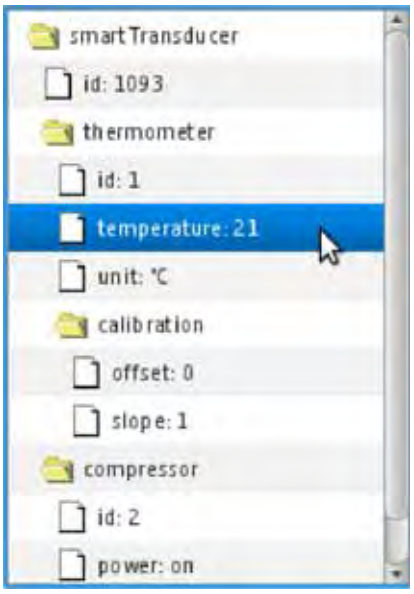


Fonte: Elaborada pelo autor.

Observa-se que os atributos *value* só são alocados nas folhas da árvore e que as folhas possuem o atributo *children* nulos. Com esta estrutura de dado é possível alocar na memória da aplicação cliente informações de qualquer transdutor inteligente, mapeado em XML ou JSON.

Na Figura 59 é apresentada uma representação gráfica produzida com os elementos de interface de JavaFX, para a árvore de objetos *Node* apresentada no diagrama da Figura 58.

Figura 59: Representação gráfica da árvore de objetos *Node*.



Fonte: Elaborada pelo autor.

4.4.4 Controle de fluxos de processamento e *Threads*

Além dos eventos disparados pelo usuário, foi necessário controlar na aplicação cliente fluxos de processamento independentes da ação do usuário para, por exemplo, manter o mecanismo de atualização periódica dos dados provenientes de um dado transdutor inteligente. Para tanto, poderia ter sido utilizado recursos de temporização como a classe *javafx.animation.Timeline* de JavaFX. No entanto, neste caso, optou-se em estender os recursos da linguagem Java, utilizando a classe *java.lang.Thread* de Java.

Para que os métodos em JavaFX Script pudessem acessar a instância em Java da classe *Thread*, foi necessária a implementação de métodos de interface, estendidos da classe *javafx.async.RunnableFuture* de JavaFX.

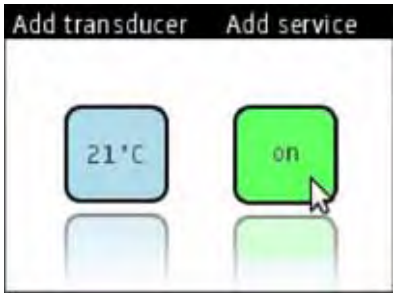
Esta implementação atendeu às necessidades de controle do processamento, independente do fluxo da aplicação principal. No entanto, levou-se para o lado servidor da aplicação cliente/servidor o fluxo de processamento produzido pela *thread*.

Nesta aplicação considerou-se indiferente o processamento no lado cliente ou servidor. No entanto, esta análise não deve ser desprezada para todo tipo de aplicação. Caso seja

necessário ou mais conveniente o processamento no lado cliente, será necessário garantir o controle do fluxo de processamento em JavaFX Script e não em Java.

Na Figura 60 é apresentada a interface de uma aplicação cliente que monitora a temperatura e o estado de um compressor. As informações apresentadas na interface são atualizadas conforme a periodicidade especificada pelo usuário da aplicação.

Figura 60: Interface da aplicação cliente de monitoramento.



Fonte: Elaborada pelo autor.

No período determinado a *thread* dispara uma requisição ao transdutor inteligente por uma conexão HTTP. O retorno da requisição é enviado ao *parser* que constrói uma árvore de objetos *Node* atualizados, referente ao transdutor inteligente que, por fim, é utilizada para atualizar os dados dos objetos da interface que apresentam as informações ao usuário final da aplicação.

4.5 CONCLUSÕES

As pesquisas apresentadas neste capítulo atingiram o objetivo de investigar tecnologias e recursos para o desenvolvimento de aplicações clientes para a arquitetura proposta. Com os testes e implementações realizados foi possível certificar-se das possibilidades de desenvolvimento, que atendem às expectativas para o trabalho.

Optou-se em adotar JavaFX pelos argumentos já descritos anteriormente. No entanto, fica a dúvida sobre JavaFX exatamente em um dos critérios que contribuíram para sua escolha, ou seja, a oferta de ferramentas livres e *runtimes* para várias plataformas. Salienta-se novamente que a Oracle descontinuou a versão JavaFX 1.3, lançando a versão JavaFX 2.0 e disponibilizando, até o momento em que este texto era escrito, apenas de ferramentas e *runtime* para o sistema operacional Microsoft Windows, o que suprime o destaque anteriormente apresentado por JavaFX 1.3. Na continuidade do desenvovimento e utilização

da aplicação cliente, no capítulo 6, esta questão é retomada, com a análise do *roadmap* desta tecnologia, publicado na ocasião pelo desenvolvedor da tecnologia.

4.6 PROPOSTAS DE TRABALHOS FUTUROS

Ao término dos estudos apresentados neste capítulo, algumas dúvidas e motivações sugerem frentes de pesquisas futuras, como as que seguem:

- Adoção e utilização livre de JavaFX 2;
- Implementação de requisições também pelo método POST do protocolo HTTP;
- Construção de árvores de objetos *Node* também de dados em JSON;
- Tratamento de fluxos de processamento da aplicação também no lado cliente;
- Testes de execução em dispositivos móveis;
- Desenvolvimento de recursos de personalização e composição de elementos pelo usuário da aplicação.

CAPÍTULO 5

A ARQUITETURA PROPOSTA

5.1 INTRODUÇÃO

A arquitetura de sistemas busca especificar cada elemento que a compõe, os posicionando e organizando em suas características e funcionalidades. Esta especificação e organização possibilitam melhor compreensão, desenvolvimento, operação e manutenção de sistemas complexos, pois o sistema como um todo pode ser tratado em elementos menores e também menos complexos.

Para Muller (2012) a arquitetura de sistemas é uma forma de criar sistemas eficientes e eficazes, fornecendo visão geral, prezando pela coerência, integridade e equilíbrio entre os elementos do sistema.

5.1.1 Objetivos específicos

Neste capítulo, objetivou-se organizar e documentar características e elementos da arquitetura proposta, já discutidos pontualmente em tópicos de capítulos anteriores. Adicionalmente, a proposta para publicação e universalização do acesso a transdutores inteligentes foi complementada e detalhada neste capítulo.

5.1.2 Justificativas

Como citado nos objetivos anteriores, buscou-se organizar e documentar características e elementos já apresentados, complementando e detalhando a proposta de arquitetura. No entanto, não se tem a pretensão de findar ou esgotar a especificação do modelo arquitetural.

Por se tratar de uma proposta de arquitetura aberta, a qualquer tempo os elementos desta arquitetura podem ser expandidos, alterados ou substituídos, desde que a interoperabilidade entre os mesmos seja mantida.

5.1.3 Trabalhos relacionados

Wobschall (2008) propôs uma arquitetura para integração do padrão IEEE 1451 a padrões em níveis de serviços, como os do OGC SWE e os da OASIS. Um dos elementos principais desta arquitetura é um tradutor que opera como um *gateway* de rede, recebendo TEDS do padrão IEEE 1451 via protocolo HTTP e repassando-as via protocolos de serviços baseados em XML, como o TML (*Transducer Markup Language*) do OGC SWE.

Como apresentado na seção 1.4, é possível empregar o padrão IEEE 1451 como barramento de campo na arquitetura especificada pelo OGC SWE, no entanto, isto não é um requisito. A arquitetura do OGC SWE prevê aplicações em alto nível de abstração sobre padrões abertos para explorar sensores Web e sistemas de sensores (OGC, 2007).

Song e Lee (2009) descrevem uma integração do padrão IEEE 1451 com o OGC SWE, usando STWS. Nesta integração, um *Web Service* atua de um lado como cliente de STWSs e de outro provê serviços nos padrões SOS (*Sensor Observation Service*), SPS (*Sensor Planning Service*), SAS (*Sensor Alert Service*) e WNS (*Web Notification Services*) do OGC SWE.

Sheth, Henson e Sahoo (2008) recorrem à Web semântica[24] e as especificações de alto nível do OGC SWE para descreverem sensores Web semânticos (SSW - *Semantic Sensor Web*), proporcionando maior descrição e significado aos dados dos sensores.

Scherer e Kleinschmidt (2011) propuseram uma arquitetura de *middleware* orientado a serviços para garantir confidencialidade, integridade e disponibilidade dos dados de RSSFs. Nesta arquitetura foram utilizados *Web Services* com a especificação WS-Security da W3C.

Zug, Dietrich e Kaiser (2011) visualizam um futuro com ambientes inteligentes, onde aplicações móveis encontrarão dinamicamente diversas redes de sensores, que oferecerão seus dados de medições. Neste cenário, os autores propõem uma abstração de programação genérica, para sensores tolerantes a falhas e fusão de nós que lidam com a variação de qualidade das medições e da comunicação de dados.

5.1.4 Organização do capítulo

Na seção 5.2 apresenta-se uma síntese da arquitetura proposta para publicação e universalização do acesso a transdutores inteligentes.

24 Extensão da Web atual que pode permitir maior significado aos dados, tanto para máquinas como para pessoas.

Na seção 5.3 é apresentado um exemplo de aplicação sobre o modelo arquitetural, com detalhamento e sugestão de arquitetura de software para a aplicação cliente, modelo de dados e sequência de interação entre os elementos do modelo de dados.

Na seção 5.4 são relatadas as conclusões deste capítulo.

Na seção 5.5 são descritas algumas propostas de trabalhos futuros relacionadas ao tema.

5.2 A ARQUITETURA PROPOSTA

Como se pode concluir pelos trabalhos relacionados na seção 5.1.3, os esforços do padrão IEEE 1451 e do OGC SWE, têm convergido para o mesmo propósito. Enquanto o IEEE 1451 permite a padronização em mais baixo nível de redes de sensores ou transdutores inteligentes, o OGC SWE permite padronizações de aplicações em alto nível de abstração, se complementando mutuamente em uma mesma arquitetura.

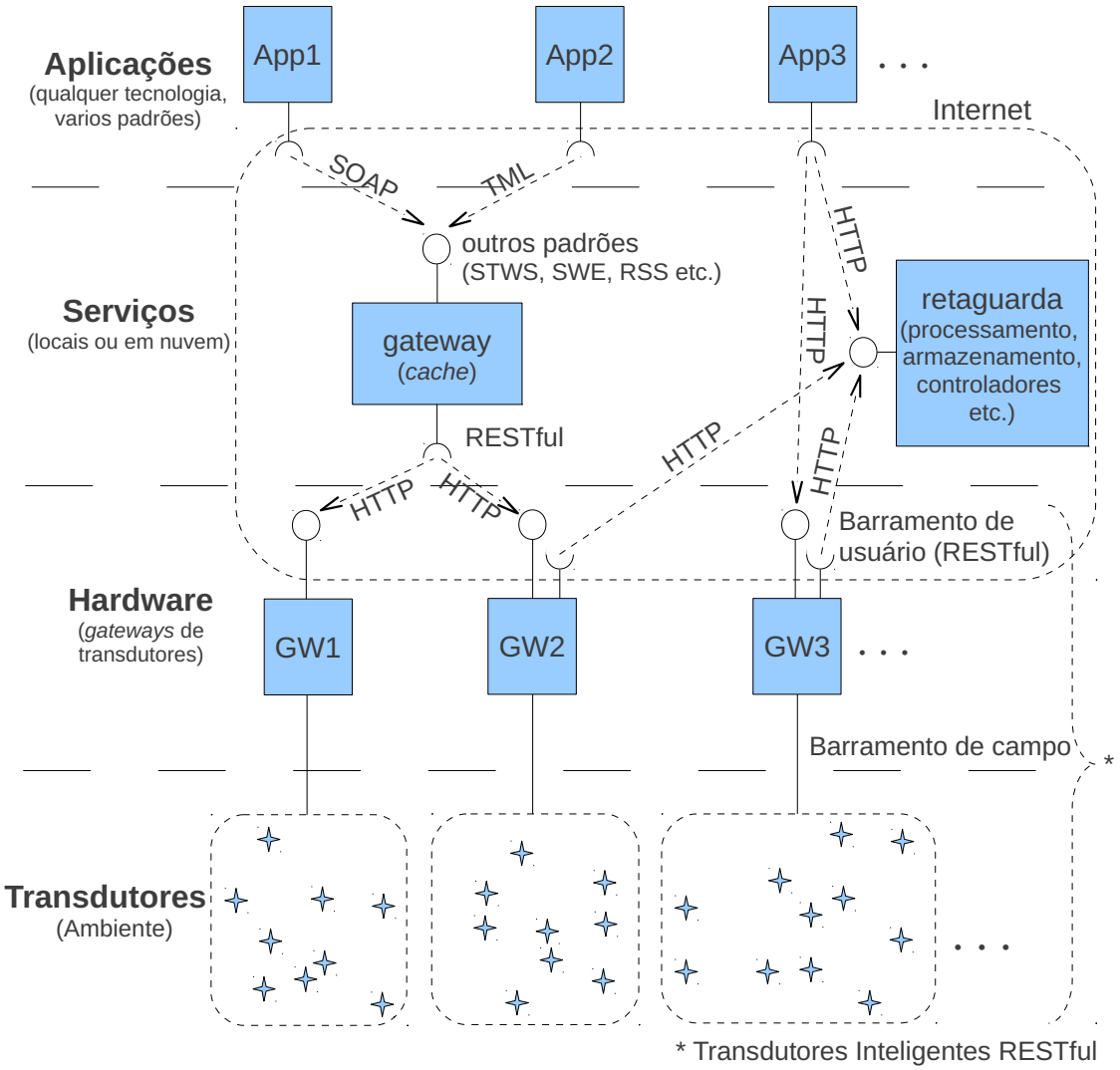
A proposta de arquitetura para publicação e universalização do acesso a transdutores inteligentes permite a integração com os padrões IEEE 1451 e OGC SWE. Embora estes padrões complementem significativamente a arquitetura proposta, eles não são requisitos para atingir o objetivo principal de publicação e universalização do acesso a transdutores inteligentes.

Fundamentalmente, a proposta de arquitetura apresentada difere dos padrões anteriores ao focar a interoperabilidade de elementos em alto nível de abstração, sobre padrões já consagrados, logo, universais. Tal interoperabilidade permite, como já mencionado, a operação com os próprios padrões citados, além de quaisquer outros que queiram se integrar à arquitetura proposta (RIBEIRO, 2012).

Uma síntese da arquitetura proposta é apresentada na Figura 61, em forma de diagrama, organizada em quatro camadas.

Um elemento chave desta arquitetura são os *gateways* de transdutores (hardware) que provêm a comunicação entre o barramento de campo e o barramento de usuário. Esses *gateways* acessam os transdutores do ambiente por meio do barramento de campo e publica as informações disponíveis como serviços RESTful, no barramento de usuários. Os *gateways* de transdutores abstraem detalhes dos transdutores do ambiente e de seu hardware ao usuário final, simplificando e universalizando o acesso aos transdutores inteligentes por adotar padrões universais para esse acesso, como o protocolo HTTP e o formato XML.

Figura 61: Arquitetura proposta.



Fonte: Elaborada pelo autor.

As duas primeiras camadas (transdutores e hardware) compõem os transdutores inteligentes na arquitetura, enquanto a primeira camada emprega o transdutor no ambiente, condiciona seu sinal e acessa o barramento de campo, a camada seguinte eleva o transdutor ao nível de serviço RESTful, no barramento de usuário, no padrão Ethernet e sobre os protocolos IP, TCP e HTTP.

Serviços na rede local, ou em computação em nuvem, podem ser implantados para proverem tradução de padrões, como o de STWS ou SWE, ou fornecer serviços de retaguarda para os transdutores inteligentes ou às aplicações desta arquitetura. Por meio destes serviços as aplicações podem acessar os dados publicados em alto nível de abstração, utilizando-se de diversos padrões, desde que seja implementado o *gateway* tradutor para o padrão desejado.

Embora possível, não é recomendável às aplicações acessarem diretamente os transdutores inteligentes, visto que *caches*[25] no *gateway* desta camada podem preservar os transdutores inteligentes de carga excessiva de requisições, principalmente quando várias aplicações fizerem requisições a um mesmo transdutor inteligente.

Assim como o *cache*, demais técnicas de redes de computadores podem ser aplicadas para melhorar a velocidade e segurança dos dados. Em uma aplicação que necessite operar em tempo real será necessário garantir túneis exclusivos para a aplicação ou implantá-las, juntamente com os serviços, na mesma subrede dos transdutores inteligentes. Por exemplo, *switchs* podem evitar que os transdutores inteligentes recebam pacotes não endereçados a eles e *firewalls* podem filtrar preliminarmente os clientes não autorizados.

A quarta e última camada pode ser composta por uma infinidade de aplicações que podem fazer uso das camadas anteriores da arquitetura. Apenas como exemplo, mas não se limitando a estas áreas, a arquitetura proposta poderia ser explorada por aplicações das seguintes áreas: Agricultura de Precisão, Automação Residencial, Monitoramento Ambiental, Operação de Sistemas de Energia Elétrica, Sensoriamento Remoto, Supervisão e Controle de Processos Industriais.

A maioria das aplicações devem trabalhar por *polling*[26], na camada de aplicações ou serviços. No entanto, é possível que os *gateways* de transdutores, na camada de hardware, possam realizar requisições HTTP à serviços de retaguarda, disparadas por interrupções de hardware, ou mesmo por *polling* no próprio hardware.

Esta estratégia para interrupções permite que sistemas de monitoramento possam disparar alertas a partir de interrupções, entretanto, será necessário que serviços de retaguarda estejam devidamente implantados para tomar as ações necessárias quando forem invocados. Se por um lado tal estratégia reduz a necessidade de processamento e tráfego de rede, por outro leva para o hardware a responsabilidade de tratar interrupções ou *polling*, contrário a ideia de levar às camadas de software o máximo possível de tarefas, reduzindo os requisitos de hardware.

Ainda, quando se realiza *polling* nas camadas de software, pode-se colocar a aplicação em estado de alerta quando algum transdutor inteligente não responder momentaneamente, independente do motivo. Porém, se a estratégia adotada for a de interrupções ou *polling* no

25 Dispositivo intermediário entre cliente e servidor que pode manter dados temporários, agilizando o retorno das respostas e evitando repetidos acessos à fonte dos dados (servidor).

26 Atividade de amostragem de dados onde uma aplicação fica em *looping* verificando (varrendo) o *status* de um ou mais dispositivos externos.

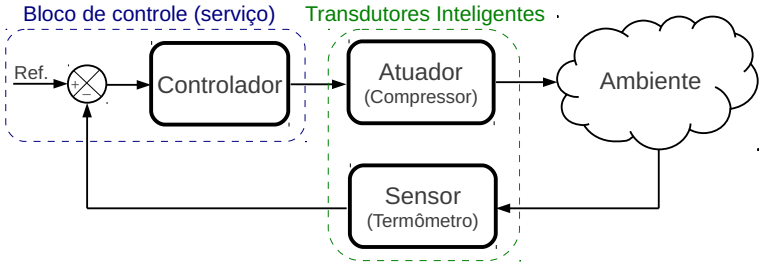
hardware, caso um transdutor inteligente pare de operar e não realize mais requisições ao serviço de retaguarda que trata a requisição, a aplicação poderá nunca perceber a falha do transdutor inteligente.

Neste sentido, sugere-se que os transdutores inteligentes sejam elementos passivos na arquitetura, sempre que possível, realizando a entrada e saída de dados apenas quando acionados pelos serviços ou aplicações. Assim, os transdutores inteligentes atuarão efetivamente como um *gateway*, entre as informações e variáveis de ambiente e o barramento de usuário.

5.3 EXEMPLO DE APLICAÇÃO

Como exemplo de aplicação e exploração da arquitetura proposta será considerado o controle de um ambiente refrigerado em uma planta em malha fechada, conforme apresentado na Figura 62. Os transdutores inteligentes nesta planta em malha fechada podem ser acessados pelo mesmo *gateway* de transdutores ou por *gateways* distintos, dependendo da localização dos transdutores (compressor e termômetro) no ambiente.

Figura 62: Planta em malha fechada de um ambiente refrigerado.

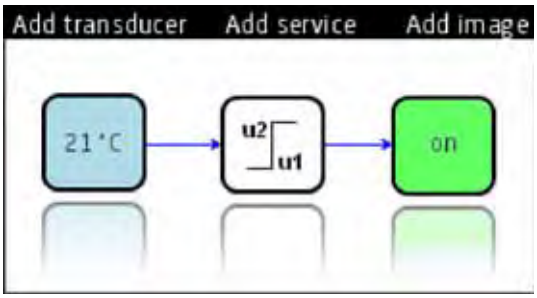


Fonte: Elaborada pelo autor.

O controlador, responsável pela ação de controle e manutenção do valor de referência pode estar implementado como serviço local ou de computação em nuvem, ficando para a aplicação cliente apenas a responsabilidade de interligá-los aos transdutores. Poderia também estar implementado na própria aplicação cliente se fosse mais conveniente, dependendo da ação de controle necessária. Também seria possível disponibilizar na aplicação cliente, ou em computação em nuvem, qualquer outra ação de controle ou serviço de retaguarda necessários para a aplicação.

Na Figura 63 é apresentada uma possível interface para a aplicação considerada de controle de um ambiente refrigerado.

Figura 63: Interface de possível aplicação.



Fonte: Elaborada pelo autor.

O bloco à esquerda indica a temperatura e unidade de medida de um sensor de temperatura e o da direita o estado de um compressor. As informações são requisitadas diretamente a transdutores inteligentes RESTful ou por qualquer outro padrão de serviços, via *gateways* na camada de serviços.

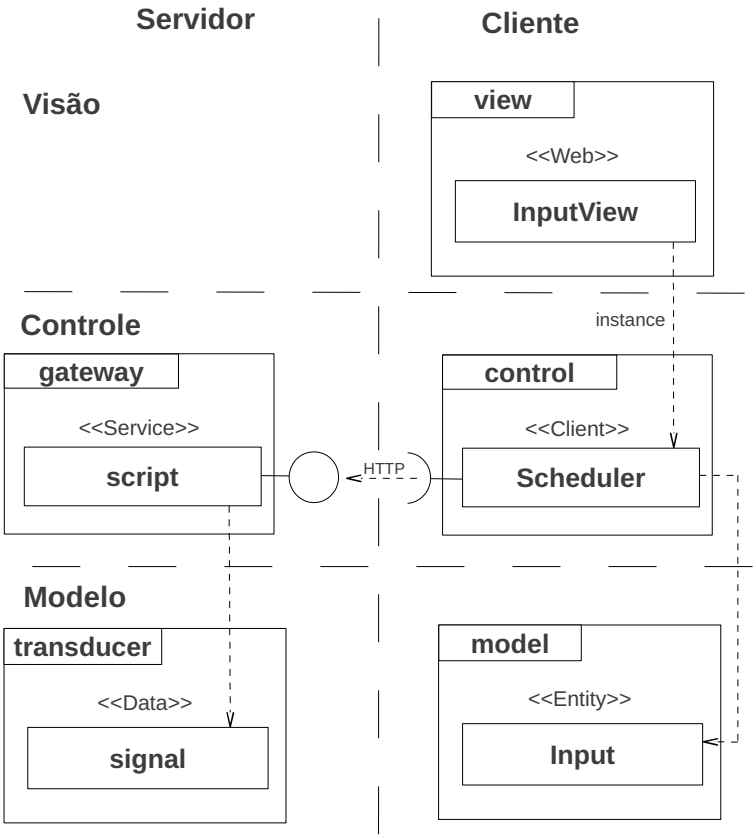
De forma assíncrona, por meio de *threads* independentes, a aplicação estabelece requisições aos transdutores inteligentes ou aos *gateways* e instanciam com os dados retornados árvores de objetos da classe *Node*, interagindo com os objetos gráficos da interface.

Desta forma, os dados do sensor de temperatura podem, por exemplo, ser enviados pela aplicação principal a um serviço que implemente um controlador *on/off*, que por sua vez envia os dados processados para o transdutor inteligente que acessa o compressor, ou para o *gateway* que intermedia o acesso.

Para substituir a ação de controle *on/off* para, por exemplo, um controlador PID (*Proportional Integral Derivative*), seria suficiente que a aplicação cliente encaminhasse os dados do sensor de temperatura para um serviço que implementasse tal controlador. Para tanto, ainda seria necessário que o transdutor inteligente que aciona o compressor também tenha capacidade de controle contínuo do mesmo, para que a saída do controlador PID possa ser interpretada pelo atuador.

Como sugestão, a Figura 64 apresenta uma possível arquitetura de software para aplicações clientes de controle de processos, no modelo MVC (*Model-view-controller*), sobre a proposta de arquitetura para publicação e universalização do acesso a transdutores inteligentes.

Figura 64: Arquitetura de software sugerida para aplicações clientes.



Fonte: Elaborada pelo autor.

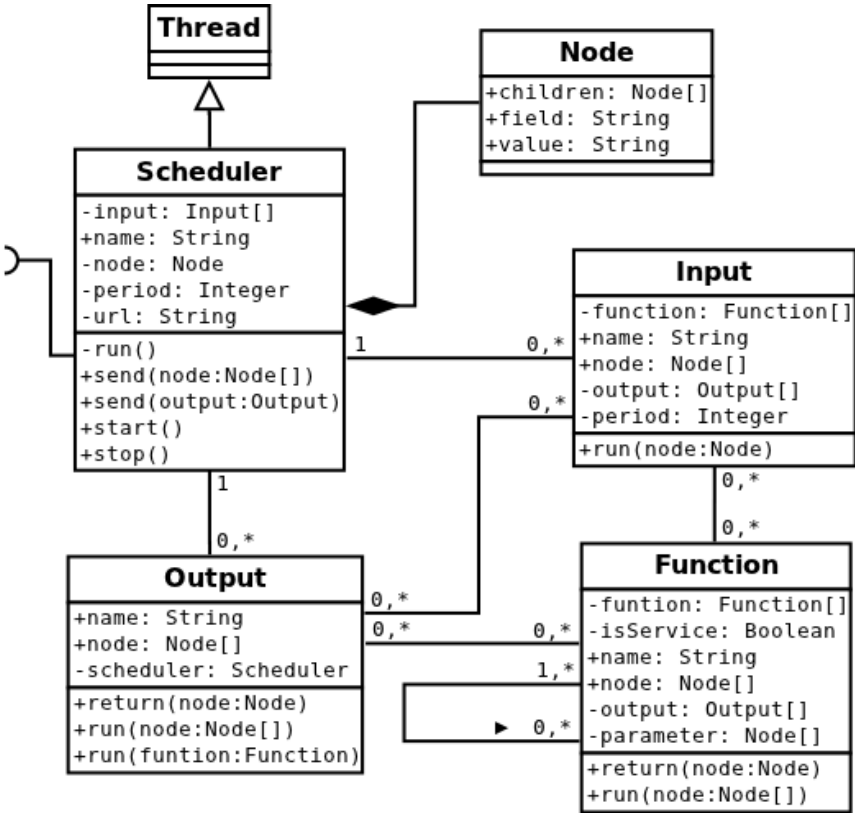
Um objeto de interface cliente Web *inputView* pode criar uma instância de um objeto *scheduler*, que controla a comunicação entre as camadas de visão e modelo e administra a comunicação com o serviço implantado no *gateway* de transdutores, ou em um *gateway* de tradução de padrões da camada de serviços da arquitetura proposta. Por sua vez, o *gateway* de transdutores acessa os dados amostrados do transdutor, provenientes do sinal condicionado do mesmo. Este último, embora adotado no exemplo de arquitetura de software, é um elemento de hardware e pode simplesmente ser abstraído pelo controle proporcionado pelo *gateway* de transdutores, como na arquitetura proposta na Figura 61.

No entanto, nesta arquitetura de software, desejou-se demonstrar a relação entre o modelo de dados do transdutor e a respectiva entidade *Input* que a representa, em alto nível de abstração, na camada de aplicação da arquitetura proposta.

Ainda, entidades *Input* da camada de modelo podem ser persistidas no lado cliente, ou enviada para algum serviço de retaguarda na camada de serviço da arquitetura proposta, para que sejam persistidas em nuvem.

O diagrama de classes apresentado na Figura 65 sugere um modelo de dados para outras entidades da camada de modelo, assim como suas associações entre si e com o *scheduler* da arquitetura de software sugerida.

Figura 65: Modelo de dados sugerido para aplicações clientes.



Fonte: Elaborada pelo autor.

Neste modelo, a classe *Scheduler* realiza a comunicação assíncrona com os serviços no lado servidor. A classe *Scheduler* herda uma classe *Thread* e é composta por uma classe Node, que pode representar estruturas de dados em árvore, como exemplificado na seção 4.4.3.

A classe *Scheduler* alimenta os dados das classes *Input*, que modelam transdutores de entrada de dados, associadas às classes *Function* ou *Output*. Os objetos *scheduler* são responsáveis pelo escalonamento das requisições no período de *refresh* dos objetos *input* associados a eles. O período de *refresh* do *scheduler* será igual ao menor período encontrado nos objetos *input* associados a ele, sendo os demais objetos *input* atualizados em períodos múltiplos do período base. Desta forma, se evita requisições desnecessárias ao *gateways* de transdutores, ou serviços em nuvem, que estejam publicando transdutores inteligentes

associados ao mesmo *scheduler* e respectivo objeto *input*.

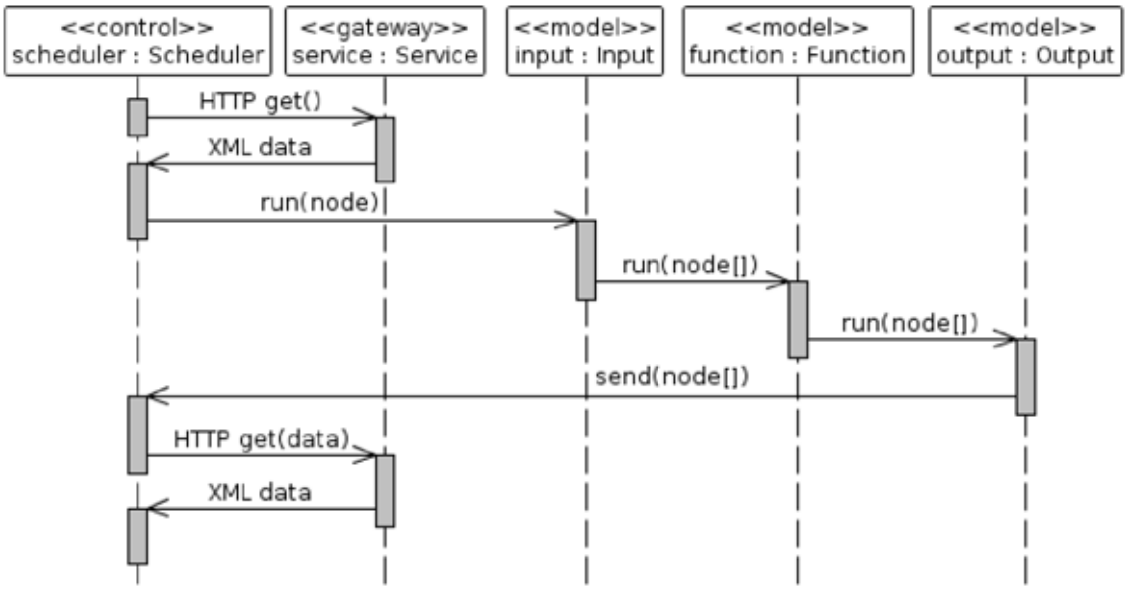
As classes *Function* modelam qualquer função de um sistema de controle associada entre as *classes Input* e *Output*, permitindo a especificação da ação de controle desejada no sistema.

As associações de zero ou muitos (0,*) entre as classes *Function*, *Input* e *Output* indicam qualquer possibilidade de associação em série ou paralelo destas entidades neste modelo de dados.

A classe *Output* por sua vez envia ao *scheduler* os dados recebidos pelas classes *Function* ou *Input*. A classe *Scheduler* trata a submissão dos dados para o serviço correspondente ao transdutor modelado em *Output*.

No diagrama da Figura 66 é exemplificada uma sequência de comunicação entre os elementos da aplicação de controle, do exemplo de planta de controle considerado na Figura 62, sobre a arquitetura proposta e modelos sugeridos.

Figura 66: Diagrama de sequência do exemplo controle de ambiente refrigerado.



Fonte: Elaborada pelo autor.

Para cada disparo de leitura de dados do *scheduler*, o *gateway* recebe uma requisição de dados pelo método GET do protocolo HTTP e retorna dos dados do transdutor inteligente estruturados em XML.

A resposta assíncrona do *gateway* ao *scheduler* inicia a tarefa do *parser* XML em extrair os dados recebidos, instanciando objetos da classe *Node* e compondo a árvore de

objetos *node*. A árvore de objetos *node* é passada a uma instância da classe *Input* que extrai apenas dos dados referente ao transdutor desejado, neste caso o termômetro, e repassa apenas a folha da árvore *node* recebida do *gateway*, referente à temperatura, para o objeto *function*.

O objeto *function* do exemplo modela um controlador *on/off*, que analisa a temperatura recebida e decide seu estado de saída. O sinal de saída do objeto *function* também é modelado em uma folha *node*, que é repassado ao objeto *output*, por meio do método *run()* desta instância de *Output*.

O objeto *output*, por sua vez, só invoca o método *send()* do *scheduler* se o sinal recebido do objeto *function* foi diferente do estado valor anterior, evitando requisições desnecessárias ao *gateway*. Caso o sinal recebido pelo objeto *output* seja diferente, este o repassa ao *scheduler* como folha da árvore *node* referente ao compressor.

O *scheduler* dispara uma requisição de dados pelo método GET do protocolo HTTP com os dados necessários para a mudança de estado do compressor, com acesso RESTful mapeado no *gateway*.

Neste exemplo, embora o *gateway* devolva dados de resposta em XML, estes não são repassados ao objeto *output*, encerrando a sequência de comunicação entre os elementos de software da aplicação cliente, até que o *scheduler* dispare uma nova requisição. Por outro lado, se for necessário ao objeto *output* a confirmação do sinal enviado, pode ser empregando a sobrecarga do método *send()* que recebe como referência o próprio objeto *output* que o invoca, de forma que o *scheduler* possa devolver o valor recebido ao objeto *output*, por meio do método *result()* que recebe como parâmetro a árvore de objetos *node*. Outra possibilidade para tal confirmação é o objeto *output* estar associado a uma nova entidade *Input*, que realiza o monitoramento do estado do compressor.

5.4 CONCLUSÕES

A proposta de arquitetura para publicação e universalização a transdutores inteligentes permite que sejam utilizadas plataformas de hardware restritas em campo e mesmo assim entregar dados às aplicações em alto nível de abstração, por diversos padrões. Isto pode contribuir para a disseminação e popularização do uso de transdutores em rede ao redor do mundo.

O alto nível de abstração dos serviços e a aproximação com as técnicas empregadas na Web podem permitir a publicação e universalização do acesso aos transdutores inteligentes a

diversos profissionais, não apenas aos especialistas em instrumentação, mas a todos os profissionais ou entusiastas em TI.

Ainda, o exemplo de aplicação apresentado, com sugestões de modelos e arquiteturas, permitiu a exemplificação e documentação de detalhes da arquitetura proposta.

5.5 PROPOSTAS DE TRABALHOS FUTUROS

Como já apresentado nas justificativas, por se tratar de uma proposta de arquitetura aberta, a qualquer tempo os elementos desta arquitetura podem ser expandidos, alterados ou substituídos, desde que a interoperabilidade entre os mesmos seja mantida.

Ainda assim, ao término deste capítulo é possível elencar algumas sugestões de trabalhos futuros, associados à arquitetura proposta:

- Levantamento e detalhamento dos principais serviços de retaguarda;
- Especificação de interfaces e modelos de serviços para processamento de dados;
- Exploração da arquitetura proposta por outros exemplos de aplicações;
- Desenvolvimento de modelo de dados para outras áreas de aplicação.

CAPÍTULO 6

IMPLEMENTAÇÕES E RESULTADOS

6.1 INTRODUÇÃO

Neste capítulo são apresentadas algumas implementações e resultados obtidos durante o desenvolvimento de alguns mecanismos e aplicações sobre a arquitetura para publicação e universalização do acesso a transdutores inteligentes.

6.1.1 Objetivos específicos

O objetivo deste capítulo foi a implementação de mecanismos sobre o modelo arquitetural proposto e aplicações que explorassem efetivamente a arquitetura para publicação e universalização do acesso a transdutores inteligentes, de forma a obter resultados que evidenciam e validam as possibilidades de emprego da arquitetura.

6.1.2 Justificativas

Por se tratar de uma arquitetura aberta sobre padrões universais de comunicação, as possibilidades de implementações e aplicações sobre o arquitetura proposta se tornam quase que infinitas. Para se chegar à demonstração de alguma aplicação específica, são apresentadas antes neste capítulo uma série de implementações que evidenciam também outras possibilidades, até que se chegue a demonstração de uma aplicação de controle de temperatura em malha fechada, com um controlador *on/off* e também com um controlador contínuo PID.

Adicionalmente é apresentada uma aplicação de supervisão e aquisição de dados de uma planta didática de controle de processo industriais, assim como a substituição do CLP (Controlador Lógico Programável) desta planta, por um controlador PI (*Proportional Integral*) na aplicação cliente, em software e distribuída, sobre o modelo arquitetural apresentado no capítulo anterior.

6.1.3 Organização do capítulo

Na seção 6.2 é apresentada a implementação de um *gateway* de transdutores, que publicam transdutores inteligentes no barramento de usuários.

Na seção 6.3 é apresentada uma aplicação cliente desenvolvida, com mecanismos de localização e acesso aos transdutores inteligentes publicados pelo *gateway* de transdutores.

Na seção 6.4 são desenvolvidos e demonstrada a utilização de alguns serviços.

Na seção 6.5 é apresentada uma aplicação de controle de temperatura, por ação de controle *on/off* e com controle contínuo, por um controlador PID processado pela aplicação principal.

Na seção 6.6 é apresentada a supervisão e aquisição de dados de uma planta de controle de processos.

Na seção 6.7 é apresentada a substituição do CLP, da planta de controle de processos apresentada na seção 6.6, por um controlador PI na aplicação cliente.

Na seção 6.8 são relatadas as conclusões deste capítulo.

Na seção 6.9 são descritas algumas propostas de trabalhos futuros.

6.2 *GATEWAY* DE TRANSDUTORES

A plataforma adotada na implementação do *gateway* de transdutores foi a DB-DP11115. Como já comentado no capítulo 2, neste momento, decidiu-se empregar plataformas mais restritas em campo, transferindo demandas computacionais para a camada de serviços da arquitetura proposta. Para tanto, seria possível a adoção da plataforma PME-10, se esta ainda estivesse operacional na ocasião em foram desenvolvidos os trabalhos deste capítulo.

Com a indisponibilidade da placa PME-10, se adquiriu a placa DB-DP11115 da Sure Electronics, também de arquitetura microcontrolada e com interface Ethernet. Com a plataforma DB-DP11115, se repetiu os procedimentos realizados com as demais plataformas embarcadas estudadas no capítulo 2 e foi adotada para as implementações deste capítulo, como o *gateway* de transdutores.

Nesta implementação, a proposta foi mapear os transdutores necessários em XML, os publicando, e implementar acessos por métodos do protocolo HTTP, conforme prevê o padrão RESTful e a arquitetura proposta.

Também seria possível a adoção de qualquer uma das outras duas plataformas de hardware adotadas no capítulo 2. Entretanto, como já argumentado anteriormente, a arquitetura proposta prevê a redução de necessidades computacionais nos transdutores inteligentes, transferindo demandas de processamento para aplicações e serviços em software que podem estar em qualquer lugar da rede, em nuvem. Desta forma, a capacidade computacional reduzida desta plataforma, em comparação com as outras duas disponíveis, alinha-se com a proposta de empregar o mínimo possível de recursos computacionais nesta camada da arquitetura.

A placa DB-DP11115 também apresenta um termistor NTC *onboard*, condicionado em malha resistiva que inverte o sentido da curva de resposta do termistor.

O sinal condicionado do termistor *onboard* é amostrado diretamente da malha resistiva pelo conversor AD, presente no microcontrolador PIC24FJ256GB106, por uma das entradas analógicas do dispositivo.

O conversor AD foi configurado para operar com resolução de 10 bits. A linearização da curva de resposta do termistor foi feita por tabela de correção na programação do microcontrolador. A correção da escala para apresentação em graus Celsius também foi realizada pela programação do microcontrolador. Embora a linearização, correção de escala e de unidade tenham sido feitas na programação do microcontrolador, após estas implementações surgiu a ideia de que bastava ter mapeado os valores de saída do conversor AD para a interface Ethernet, em XML, deixando todo o trabalho de tratamento destes dados para as camadas de software da arquitetura proposta, minimizando assim as necessidades de processamento computacional nesta camada da arquitetura. Neste sentido, uma das saídas analógicas mapeadas foi a que está amostrando o sinal do termistor.

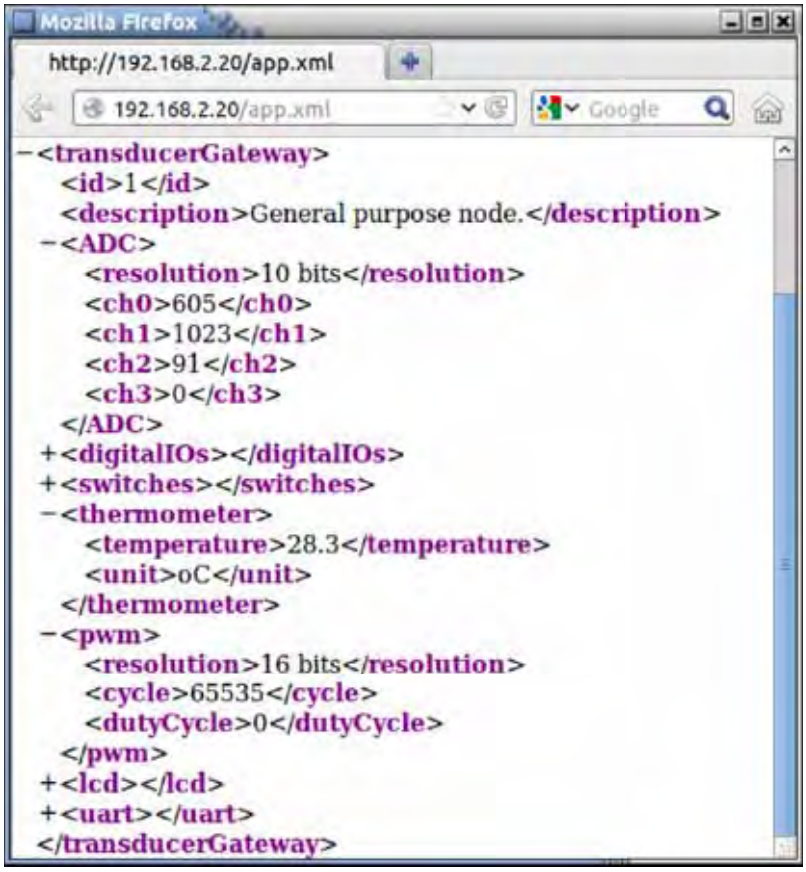
Após o tratamento dos dados discutidos anteriormente, os dados da temperatura em graus Celsius foram mapeados em XML para serem publicados na interface Ethernet, para tanto, foi empregada a implementação da Pilha TCP/IP da Microchip, como também realizado nos experimentos com a placa PME-10.

Realizado o mapeamento em XML do sensor de temperatura, diversos outros recursos da placa DB-DP11115 também foram mapeados e publicados em XML, sobre o protocolo HTTP, como pode ser observado na Figura 67.

A *tag ADC* refere-se ao conversor AD, onde foram mapeados quatro canais analógicos de entrada, com resolução de 10 bits. A *tag ch0* está associada ao mesmo dado da *tag*

temperature, porém sem tratamento. A *tag ch1* está com seu canal AD em nível alto (3,3V), a *tag ch2* está com seu canal AD flutuando (desconectado) e a *tag ch3* está com seu canal AD em nível baixo (0V)

Figura 67: Mapeamento XML de recursos da placa DB-DP11115.



Fonte: Elaborada pelo autor.

A *tag digitalIOs* apresenta-se fechada na Figura 67 para melhor apresentação das demais *tags*, no seu interior são mapeados oito pinos digitais bidirecionais. A *tag switches*, também fechada, contém o estado de três chaves *push-button*. No Apêndice A pode ser verificado o mapeamento completo dos recursos da placa DB-DP11115 em XML, com todas as *tags* abertas.

Na *tag thermometer* foram mapeados os dados da temperatura em graus Celsius na *tag temperature*, proveniente do sinal condicionado do termistor *onboard* e já tratado, como descrito anteriormente.

A *tag pwm* realiza o mapeamento de uma saída PWM com resolução de 16 bits. A *tag cycle* refere-se ao período completo do sinal PWM e a *tag dutyCycle* ao período ativo (alto)

do sinal.

As *tags lcd* e *uart*, também fechadas, realizam o mapeamento, respectivamente, do LCD (*Liquid Crystal Display*) e da UART (*Universal Synchronous Receiver/Transmitter*) presentes na placa DB-DP11115.

A própria publicação dos recursos da placa DB-DP11115 por meio de um navegador Web já é uma demonstração da padronização adotada para o mapeamento dos recursos. O endereçamento dos transdutores inteligentes na rede é realizado pelo protocolo IP, a transferência dos dados é realizada sobre o protocolo de aplicação HTTP, com dados estruturados em XML. A arquitetura proposta compartilha de padrões da Web em seu barramento de usuário, pelo qual as camadas de software têm acesso aos transdutores inteligentes.

Para acessar a escrita aos recursos de saída dos transdutores inteligentes foram adotados os métodos POST e GET do protocolo HTTP. Assim, utilizando o método GET, é possível enviar uma mensagem para o LCD com a seguinte URL:

```
http://ip/app.xml?text=mensagem
```

Sendo *ip* o endereço IP na rede do *gateway* de transdutores e *mensagem* o texto da mensagem. A *tag text* está dentro da estrutura XML da *tag lcd*. O transdutor inteligente reconhece a submissão do método GET com a *tag text* e envia o dado *mensagem* ao LCD da placa.

Outro exemplo de acesso pode ser verificado na URL a seguir:

```
http://ip/app.xml?d7=1
```

Neste caso a *tag d7* refere-se ao oitavo pino mapeado na estrutura XML *digitalIOs*. O transdutor inteligente reconhece a submissão do método GET com a *tag d7* e atribui o valor lógico *1* ao respectivo pino do mapeamento.

O exemplo a seguir acessa o ciclo de trabalho da saída PWM:

```
http://ip/app.xml?cycle=65535&dutyCycle=32767
```

As *tags cycle* e *dutyCycle* estão dentro da estrutura XML da *tag pwm*. O transdutor

inteligente reconhece a submissão do método GET com as *tags cycle* e *dutyCycle* e altera o ciclo de trabalho da saída PWM. Definindo *cycle* como *65535* (2^{16}-1) e *dutyCycle* em *32767* [(2^{16}/2)-1], o transdutor inteligente produzirá modulação PWM com período ativo de 50%.

Assim, o *gateway* de transdutores implementado publica e universaliza o acesso a alguns recursos da placa, mapeando alguns transdutores inteligentes, por um barramento de usuário no padrão Ethernet, sobre os protocolos IP, TCP e HTTP, em XML e no padrão RESTful.

A aplicação desenvolvida para mapeamento em XML, dos recursos da placa DB-DP11115, partiu de um exemplo de aplicação do próprio fabricante de um servidor Web, com páginas gravadas em dispositivo de armazenamento conectado à interface USB. Embora também tenha sido utilizada a interface USB, devido ao reduzido tamanho do único *script* necessário para esta aplicação, imagina-se como trabalho futuro a utilização da memória Flash interna do microcontrolador, liberando a interface USB para outros fins, ou mesmo para redução dos requisitos e custos do hardware do *gateway* de transdutores.

Embora se buscasse implementar o *gateway* de transdutores segundo o padrão RESTful, devido à simplicidade deste padrão se comparado a outros padrões de serviços, as implementações desenvolvidas no *gateway* de transdutores se limitaram a permitir a publicação e acesso dos recursos anteriormente descritos, sem que todos os mecanismos possíveis em um serviço RESTful fossem implementados. Para tanto, na seção 6.4 são apresentados, entre outros serviços, implementações de *gateways* de tradução de padrões, que exploram a capacidade computacional em nuvem, para implementar mecanismos neste e em quais outros padrões forem necessários.

6.3 A APLICAÇÃO CLIENTE

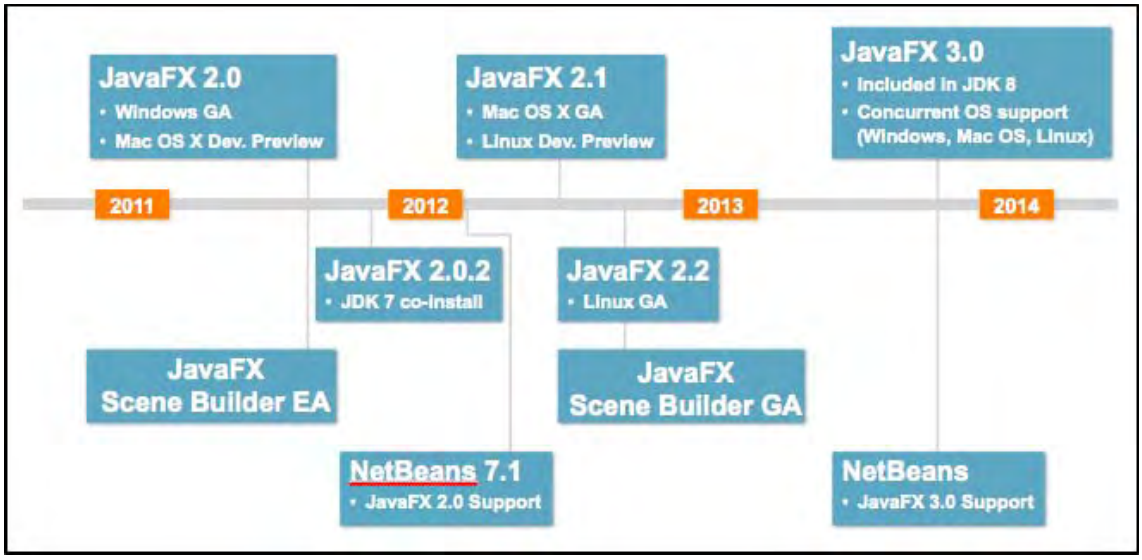
Neste momento do desenvolvimento de aplicações, continuavam as dúvidas levantadas no capítulo 4, sobre as pretensões da Oracle quanto à tecnologia JavaFX 2. Embora tenham sido disponibilizadas ferramentas para o desenvolvimento em JavaFX 2 também em outras plataformas, além da Windows, e *runtime* também para o sistema operacional Mac, passado quase um ano do lançamento de JavaFX 2.0 ainda não havia *runtime* disponível para a plataforma Linux.

Neste cenário, decidiu-se manter a aplicação e mecanismos desenvolvidos em JavaFX 1.3, nesta versão da tecnologia. Mais uma vez, não havia confiança de que a migração para

JavaFX 2 seria a melhor opção, retomando-se a discussão do capítulo 4 de outras possibilidades de tecnologias para a aplicação cliente.

Com o desenvolvimento da aplicação cliente em curso a Oracle (2012) publicou o *roadmap* da tecnologia JavaFX 2, apresentado na Figura 68, onde se compromete a disponibilizar o *runtime* para a plataforma Linux (Linux GA[27]) em meados do segundo semestre de 2012, o que se confirmou até o fechamento deste trabalho.

Figura 68: *Roadmap* de JavaFX 2.



Fonte: (ORACLE, 2012).

Neste *roadmap também* é assumido o compromisso de que a partir do JavaFX 3, o suporte às plataformas será concorrente, diferente do que tem ocorrido até o momento, e alinhado a antiga proposta de tecnologia multiplataforma consolidada na tecnologia Java.

Diante do exposto, o desenvolvimento da aplicação cliente, mecanismos e serviços, necessários para a implementação dos estudos de caso apresentados neste capítulo, foram finalizados em JavaFX 1.3. Ficando como sugestão de desenvolvimento futuro a migração para JavaFX 2 ou 3, caso o compromisso do desenvolvedor, em retomar a característica multiplataforma desta tecnologia, se concretizar.

6.3.1 Acesso aos transdutores inteligentes

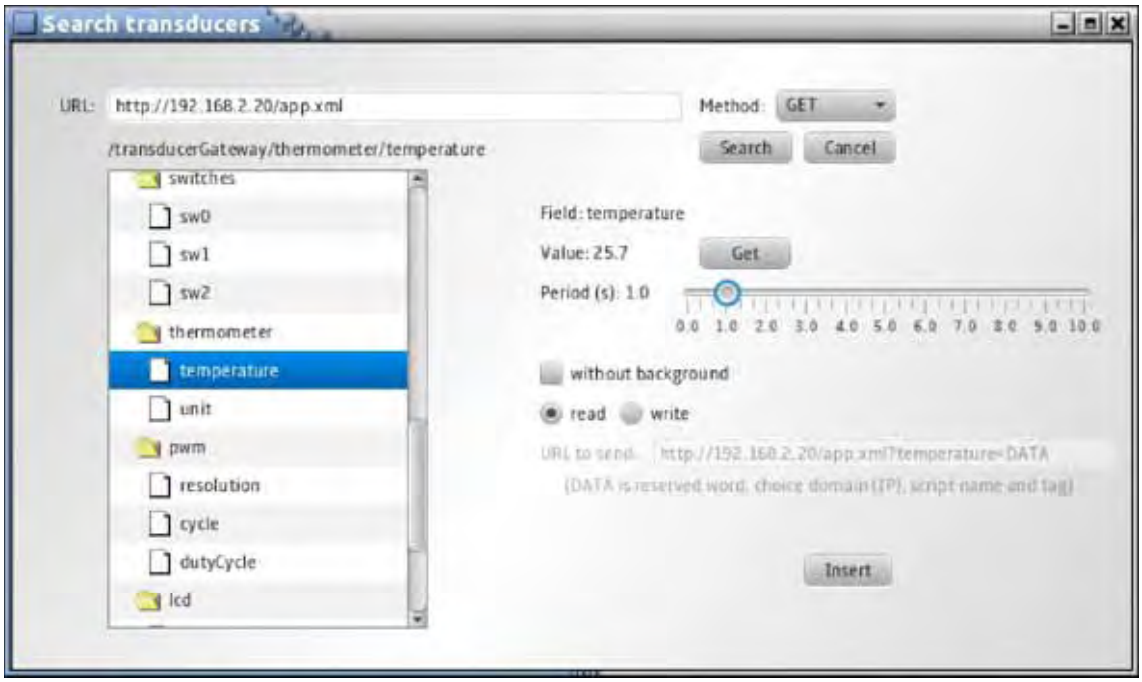
Utilizando-se de diversos mecanismos desenvolvidos em JavaFX, já apresentados no

27 O termo GA (*General Availability*) refere-se a disponibilidade completa dos recursos da tecnologia.

capítulo 4, como: conexão HTTP, *thread* de controle de fluxo de processamento, *parser* XML, classe *Node*, elementos gráficos etc., foram desenvolvidas algumas funcionalidades na aplicação cliente para a busca de transdutores inteligentes na rede e estabelecimento de comunicação entre os transdutores inteligentes, ou serviços, e a aplicação cliente.

Na Figura 69 é apresentada a interface de busca de transdutores inteligentes.

Figura 69: Interface de busca de transdutores inteligentes (entrada de dados).



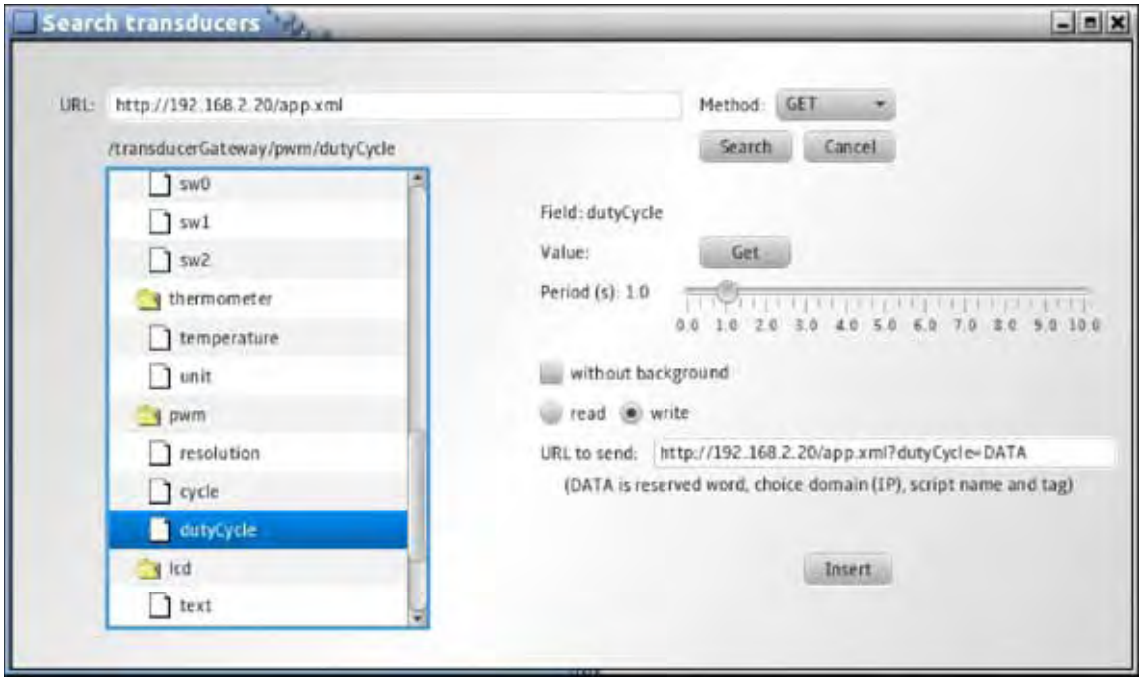
Fonte: Elaborada pelo autor.

Um *gateway* de transdutores responde em HTTP no endereço IP 192.168.2.20 a uma requisição realizada por uma *thread* da aplicação cliente. Este *gateway* de transdutores retorna o mapeamento de seus recursos em uma estrutura de dados XML. O *parser* XML extrai os dados da estrutura XML e instancia uma arvore de objetos da classe *Node*. A árvore de objetos da classe *Node* é representada por elementos gráficos da interface, compondo uma estrutura de pastas e arquivos. Por esta representação gráfica o usuário seleciona o transdutor inteligente desejado *(thermometer)*, o período de *refresh* (*1,0*), o sentido da comunicação (*read*) e insere um objeto de entrada de dados na aplicação cliente.

Na Figura 70 também é apresentada a interface de busca de transdutores inteligentes, que também utiliza os recursos descritos anteriormente. No entanto, o usuário seleciona o transdutor de saída de dados (*pwm*), o sentido da comunicação (*write*) e insere um objeto de

saída de dados na aplicação cliente. Neste caso, o período de *refresh* é ignorado, pois a escrita de dados é disparada assincronamente pela *thread* que trata a comunicação do objeto de entrada de dados.

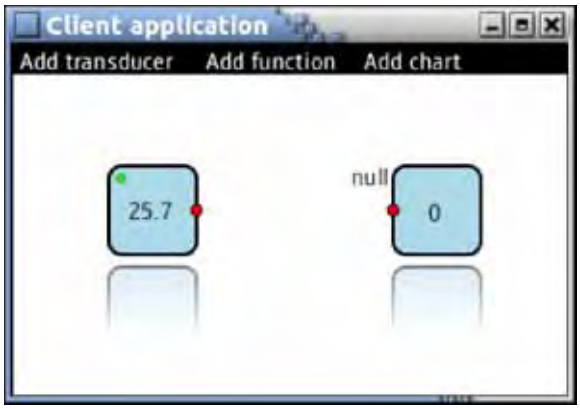
Figura 70: Interface de busca de transdutores inteligentes (saída de dados).



Fonte: Elaborada pelo autor.

Na Figura 71 são apresentadas, da esquerda para direita, respectivamente, as representações gráficas dos objetos de entrada de dados e saída de dados inseridos na aplicação cliente, pelas interfaces anteriores de busca de transdutores inteligentes.

Figura 71: Interface da aplicação cliente com objetos de entrada e saída de dados.



Fonte: Elaborada pelo autor.

O bloco do lado esquerdo representa graficamente o objeto de entrada de dados. O ponto verde, no canto esquerdo superior, indica que a *thread*, que trata a comunicação com o *gateway* de transdutores, realizou uma requisição e está aguardando a resposta. O ponto vermelho, na lateral direita do bloco, indica o ponto de ligação gráfica do bloco de entrada de dados, com outras representações gráficas na aplicação cliente. O dado *25,7* indica valor atual da *tag temperature*, do transdutor inteligente *thermometer*.

O bloco do lado direito, representa graficamente o objeto de saída de dados. O ponto vermelho, na lateral esquerda do bloco, indica o ponto de ligação gráfica do bloco de saída de dados, com outras representações gráficas na aplicação cliente. O dado *null* indica que nenhum dado está sendo enviado ao objeto de saída de dados e *0* indica o valor atual da *tag dutyClycle*, do transdutor inteligente *pwm*.

O usuário pode interagir com a interface gráfica para configurar e interligar os blocos, compondo a aplicação cliente.

6.3.2 Blocos de função

Dando continuidade no desenvolvimento de recursos na aplicação cliente, foram implementadas três funções locais típicas de sistemas de controle de processos: *on*/*off*, ganho e PID.

Sistemas de controle de processos são arranjos de elementos que agem em conjunto para manter uma dada variável do processo (variável controlada) estabilizada em um dado valor (*setpoint*) ou dentro de uma faixa de valores.

Uma das ações de controle mais simples e popular de sistemas de controle domésticos e industriais é a ação de duas posições (*on/off*). Nestes sistemas o elemento atuante (atuador) tem apenas duas posições fixas, como ligado ou desligado, aberto ou fechado, etc. (OGATA, 2003).

Na Figura 72 é apresentada a interface de parametrização e inserção da função de controle *on/off* na aplicação cliente.

Escolhida a opção O*n/Off* é habilitada a escolha dos valores de mínimo e máximo do sinal de entrada, para disparo das ações *on* e *off*, respectivamente. Os sinais de saída *on* e *off* podem ser substituídos por *alias*, ajustando o sinal de saída da função ao tipo de sinal necessário, para a função seguinte ou objeto de saída de dados. Ainda, dependendo da necessidade da aplicação, o estado inicial da função de controle *on/off* pode ser: *null*, *on*, ou

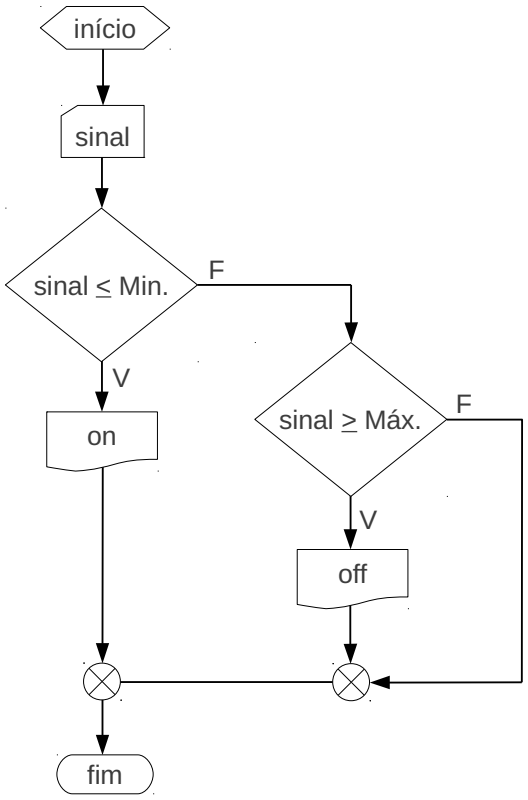
off. Este estado inicial da função de controle *on/off* deve ser definido no instante de sua inserção na aplicação cliente.

Figura 72: Interface de parametrização e inserção da função de controle *on/off*.

Fonte: Elaborada pelo autor.

O fluxograma mostrado na Figura 73 apresenta o fluxo da tomada de decisão da função de controle *on/off*.

Figura 73: Fluxo de tomada de decisão da função de controle *on*/*off*.



Fonte: Elaborada pelo autor.

O fluxo de processamento só é disparado pelo objeto associado à função de controle *on/off* (entrada de dados ou outra função), se o dado do objeto associado sofrer alguma alteração.

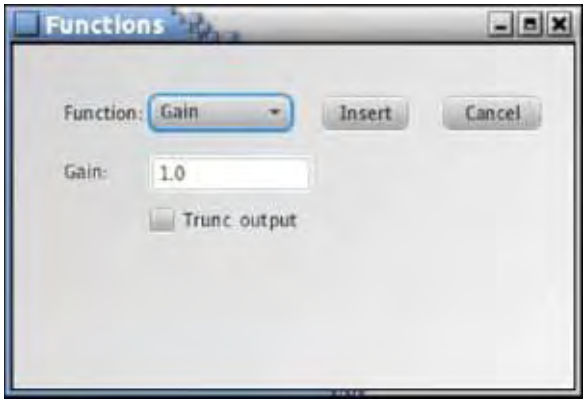
Quando disparado pelo objeto associado à função de controle *on/off*, o sinal de entrada da função é lido. A primeira tomada de decisão verifica se o sinal é menor ou igual ao valor mínimo de referência. Caso verdadeiro, o dado *on* é enviado à saída da função e o fluxo de processamento se finaliza. Caso falso, o fluxo de processamento vai para a segunda tomada de decisão. A segunda tomada de decisão verifica se o sinal de entrada lido é maior ou igual ao valor máximo de referência. Caso verdadeiro, o dado *off* é enviado à saída da função e o fluxo de processamento se finaliza. Caso também seja falso, o fluxo também se finaliza, porém, sem que o dado da saída da função tenha sido alterado.

Assim que o sinal enviado à função de controle *on*/*off* sofrer alteração, o objeto associado à função irá disparar novamente o fluxo de tomada de decisão.

Outra função típica de sistemas de controle de processos é o ganho, esta pode ser compreendida essencialmente como um amplificador de sinal. Pode também ser compreendida e utilizada como uma ação de controle proporcional (OGATA, 2003).

Na Figura 74 é apresentada a interface de parametrização e inserção da função de ganho na aplicação cliente.

Figura 74: Interface de parametrização e inserção da função de ganho.



Fonte: Elaborada pelo autor.

Escolhida a opção *Gain* é habilitada a escolha do valor do ganho e a opção de truncamento do valor de saída da função. A opção de truncamento do valor de saída é útil quando a função associada seguinte, ou objeto de saída de dados, não trabalha com dados em

ponto flutuantes, o que é muito comum em plataformas microcontroladas que não possuem FPU (*Float Point Unit*).

O valor do sinal de saída (*s*) da função de ganho é dado pelo produto do fator de ganho (*A*) e do valor do sinal de entrada (*e*), que pode ser expressão pela equação (1) a seguir:

$$s_i = A.e_i \quad (1)$$

Sendo *i* o índice do dado amostrado.

A função PID combina as ações de controle proporcional, integral e derivativo. Controladores PID permitem o controle contínuo de variáveis, diferente dos controladores *on/off*, e podem ser aplicados na maioria dos sistemas de controle. Em particular, controladores PID podem ser empregados quando não se conhece o modelo matemático da planta, utilizando-se de métodos empíricos para a sintonia do sistema de controle (OGATA, 2003; GUERRA, 2009).

Na Figura 75 é apresentada a interface de parametrização e inserção da função de controle PID na aplicação cliente.

Figura 75: Interface de parametrização e inserção da função de controle PID.

Fonte: Elaborada pelo autor.

Escolhida a opção *PID* é habilitada a escolha dos parâmetros da função de controle PID: *setpoint* (*SP*), constante proporcional (*Kp*), constante integral (*Ki*), constante derivativa (*Kd*) e o período.

SP é o valor desejado para a variável controlada, *Kp*, *Ki* e *Kd* são os valores de sintonia do controlador para as ações proporcional, integral e derivativa, respectivamente.

O processamento da função de controle PID é calculado pela equação (2) do PID

Digital apresentada por Guerra (2009), para *setpoint* invariante no tempo:

$$s_i = s_{i-1} + Kp.(erro_i - erro_{i-1}) + Ki.T.erro_i + \frac{Kd}{T}.(erro_i - 2.erro_{i-1} + erro_{i-2}) \quad (2)$$

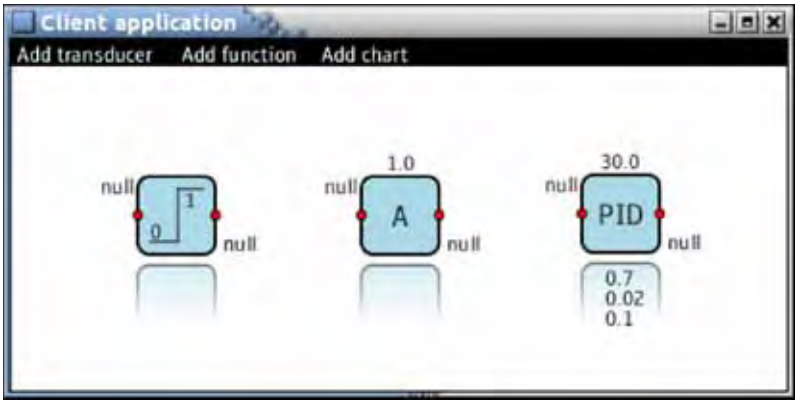
Sendo:

- $erro_i = SP - e_i$;
- $s \rightarrow$ sinal de saída;
- $e \rightarrow$ sinal de entrada;
- $i \rightarrow$ índice do dado amostrado;
- $T \rightarrow$ período de amostragem.

Guerra (2009) também sugere em seu trabalho um modelo matemático para o PID digital, com *setpoint* variante no tempo.

Na Figura 76 são apresentadas, da esquerda para direita, respectivamente, as representações gráficas das funções de controle *on/off*, ganho e controle PID. As representações gráficas dos blocos de funções são inseridas pelo usuário, na aplicação cliente, pelas interfaces gráficas de parametrização e inserção de funções apresentadas anteriormente.

Figura 76: Interface da aplicação cliente com representações gráficas de funções.



Fonte: Elaborada pelo autor.

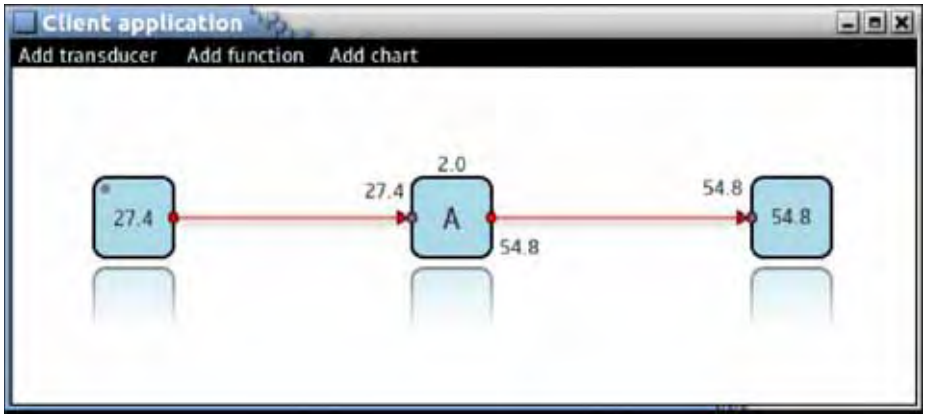
Os pontos vermelhos do lado esquerdo dos blocos das funções representam graficamente o ponto de entrada de sinal e o ponto vermelho do lado direito o ponto de saída de sinal. Os dados *null* do lado esquerdo dos blocos indicam que os blocos de funções não têm nenhum sinal associado e os dados *null* do lado direito dos blocos indicam que nenhum valor foi assumido para as saídas.

O bloco do lado esquerdo é a representação gráfica da função de controle *on/off*, onde também é indicado o valor de mínimo (0) e máximo (1) da função. Ao centro encontra-se a representação gráfica da função de ganho, onde pode ser observada também a indicação do fator de ganho (1,0). O bloco do lado direito é a representação gráfica da função de controle PID, onde é indicado o *setpoint* (30,0) acima do bloco e as constantes proporcional (0,7), integral (0,02) e derivativa (0,1) abaixo do bloco de função.

6.3.3 Resultados

Com o objetivo de demonstrar o funcionamento da aplicação cliente sobre a arquitetura proposta, no entanto, sem realizar efetivamente uma aplicação de propósito específico. A Figura 77 apresenta a interface de uma aplicação cliente com função de ganho, que explora dois transdutores inteligentes publicados pelo *gateway* de transdutores.

Figura 77: Interface de aplicação cliente com função de ganho.



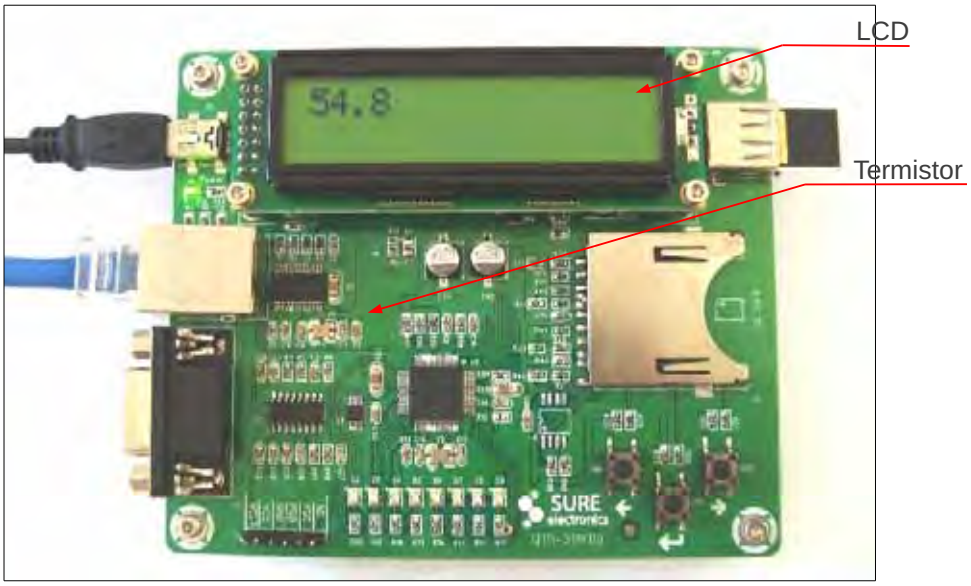
Fonte: Elaborada pelo autor.

O bloco do lado esquerdo realiza a entrada de dados da *tag temperature*, do transdutor inteligente *thermometer,* enviando os dados ao bloco central. O bloco de função ao centro aplica um fator de ganho (2,0) e envia o resultado ao bloco do lado direito. O bloco direito realiza a saída dos dados recebidos do bloco anterior, enviando para os dados ao transdutor inteligente *lcd*, especificamente pelo acesso à *tag text*.

A associação entre os blocos, representada pelas setas vermelhas, é realizada de forma interativa com o usuário, que com o cursor do mouse pode efetuar as ligações dos blocos, definindo o fluxo de dados da aplicação.

Assim, a temperatura ambiente amostrada do termistor da placa microcontrolada, que embarca o *gateway* de transdutores, é levada à aplicação principal em graus Celsius pelo objeto de entrada de dados. O valor da temperatura é multiplicado por dois pela função de ganho. Por fim, o valor da temperatura multiplicado por dois é enviado para o LCD da placa microcontrolada pelo objeto de saída de dados, conforme pode ser observado na Figura 78.

Figura 78: Termistor e LCD da placa microcontrolada.



Fonte: Elaborada pelo autor.

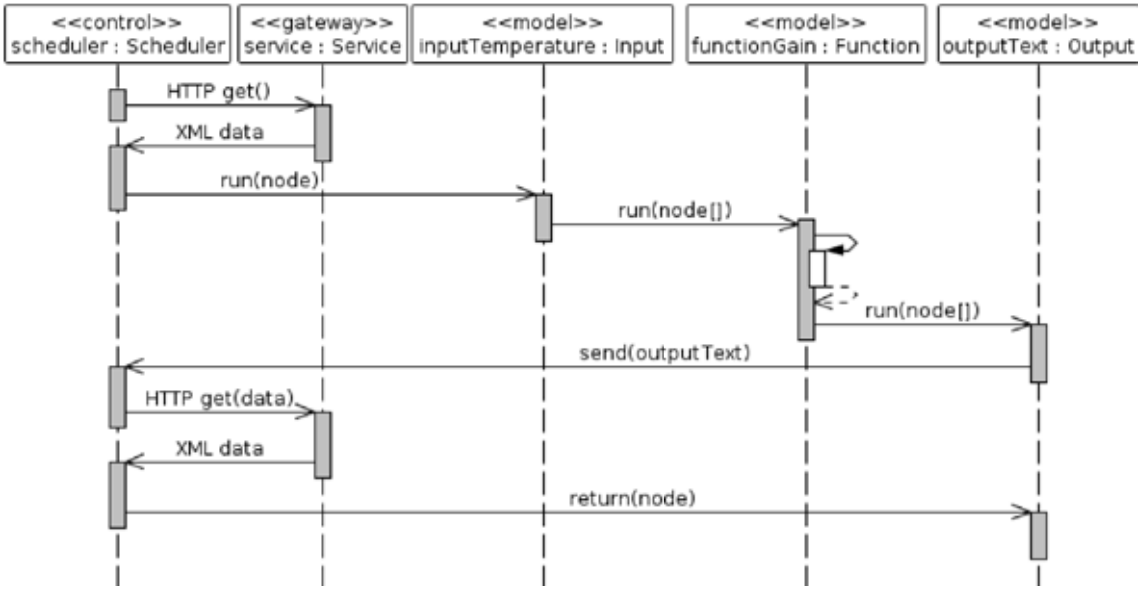
Levando em conta apenas que o dado proveniente do termistor foi apresentado no LCD, ao dobro do valor correspondente da temperatura em graus Celsius. Esta aplicação parece não ser muito relevante, tão pouco útil. Seria mais simples programar o próprio microcontrolador da placa para levar o dado proveniente do termistor ao LCD, na escala desejada.

No entanto, tal implementação percorre as camadas da arquitetura para publicação e universalização do acesso a transdutores inteligentes, nos padrões já descritos anteriormente. Observa-se que esta placa DB-DP11115 embarca o *gateway* de transdutores, oferecendo mecanismos de comunicação com os transdutores envolvidos. Nesta aplicação, a placa DB-DP11115 também embarca os transdutores envolvidos, no entanto, quaisquer outros transdutores externos poderiam ser acessados.

Qualquer necessidade de processamento, como a multiplicação pelo fator de ganho, é realizada externamente, pela aplicação cliente, como neste caso, ou em serviço em nuvem.

O diagrama de sequência apresentado na Figura 79 detalha o fluxo de dados desta aplicação desenvolvida. Neste diagrama é considerado o modelo de dados sugerido no capítulo 5, para a arquitetura de software da aplicação cliente. No entanto, a aplicação efetivamente desenvolvida não reflete exatamente este modelo.

Figura 79: Diagrama de sequência da aplicação com termistor e LCD.



Fonte: Elaborada pelo autor.

O modelo de dados sugerido, para a arquitetura de software da aplicação cliente, foi refinado durante o desenvolvimento da aplicação cliente. No entanto, a implementação da aplicação cliente não acompanhou o modelo, pois foi concebida bem antes da sugestão apresentada e refinada. Neste sentido, o modelo de dados sugerido documenta a experiência do desenvolvimento da aplicação cliente, sendo referência para futuro desenvolvimento de nova versão da aplicação cliente, em JavaFX 2, 3, ou em outra tecnologia que vier a ser adotada.

Para cada período de *refresh* o objeto *scheduler* dispara o fluxo de dados com uma requisição ao *gateway* de transdutores. Assim que o *scheduler* recebe a resposta assíncrona do *gateway*, o *scheduler* invoca o método *run()* do objeto *inputTemperature*, passando toda a árvore de objetos da classe *Node*.

O objeto *inputTemperature* recupera a folha da árvore *node* associação à *tag temperature*, do termistor, e a repassa ao objeto *functionGain*, por meio de seu método *run()*.

O objeto *functionGain* multiplica o dado recebido pelo fator de ganho e repassa o

resultado ao objeto *outputText*, que está associado à *tag text* do LCD. Exceto pelo processamento dos mecanismos de comunicação e extração de dados, este é o único momento que a aplicação realiza uma manipulação de dados, na multiplicação do dado de entrada pelo fator de ganho.

Neste caso, o objeto *outputText* envia sua referência ao *scheduler* que, além de enviar o dado contido em *outpuText* para o LCD, ainda devolve o retorno em uma árvore de objetos da classe *Node*, pela referência do objeto *outputText* recebido, invocando o método *return()* deste objeto. Este retorno ao objeto *Output* permite a confirmação de sucesso do dado enviado, o que não ocorreu no exemplo de sequência de comunicação apresentada no capítulo 5, ilustrada no diagrama de sequência da Figura 66.

6.4 SERVIÇOS

Nesta seção são apresentadas algumas implementações que evidenciam as infinitas possibilidades do emprego de serviços, para o processamento em nuvem dos dados de transdutores inteligentes.

6.4.1 *Gateway* RESTful

A fim de implementar demais mecanismos, como a navegação em profundidade na estrutura de dados XML e descrição automática do serviço (WADL), foi implementado um serviço em nuvem para operar como um *gateway* de tradução de padrões. Neste caso, os dois lados do *gateway* se comunicam no padrão RESTful, no entanto, o lado da aplicação cliente, via este *gateway* de tradução, oferece mecanismos do padrão que não foram implementados no *gateway* de transdutores.

Para implementar este *gateway* RESTful foi utilizado o IDE Netbeans, assim como alguns assistentes do IDE e a tecnologia Java EE.

Na Figura 80 é apresentado parte do arquivo WADL dos serviços RESTful.

Na parte do arquivo WADL apresentado, a maioria das *tags* estão fechadas para melhor visualização das *tags* do módulo PWM. O Apêndice B apresenta o arquivo WADL completo deste *gateway* RESTful.

O arquivo WADL é bastante sucinto, mas permite o mapeamento automático de parte dos recursos disponíveis, por aplicações clientes consumidoras. No entanto, a descrição se

limita a apresentar o caminho (*pwm*) do recurso, os identificadores (“*postXml*” e “*getXml*”) e nomes (“*POST*” e “*GET*”) dos métodos de acesso e a forma em que o dado é representado (“*application/xml*”).

Figura 80: Arquivo WADL do *gateway* RESTful.

Fonte: Elaborada pelo autor.

Esta descrição não é suficiente para se conhecer todos os recursos dos transdutores inteligentes disponíveis. O acesso aos seus mapeamentos XML individuais, como o do PWM apresentado a seguir, proporciona as mesmas informações já disponíveis antes, sem este *gateway* RESTful. No entanto, é possível afirmar que esta é uma implementação completa de um serviço RESTful, o que não era possível afirmar da implementação deste serviço no *gateway* de transdutores.

Na Figura 81 pode ser observada uma requisição HTTP da aplicação cliente, pelo método GET, ao módulo PWM. Na Figura 82 é apresentada a resposta do serviço à requisição realizada.

Na requisição a *tag pwm* completa a URL e o retorno da estrutura de dados XML devolvida refere-se apenas ao módulo PWM.

Figura 81: Requisição GET ao serviço RESTful do módulo PWM.

```
GET /gatewayRESTful/webresources/pwm HTTP/1.1
Accept: application/xml
User-Agent: Java/1.7.0_02
Host: localhost:8080
Connection: keep-alive
```

Fonte: Elaborada pelo autor.

Este *gateway* RESTful permite que os transdutores inteligentes sejam acessados por URLs independentes, reduzindo a quantidade de dados trafegados e simplificando o acesso pela aplicação cliente.

Figura 82: Resposta do serviço RESTful com dados do módulo PWM.

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.0 JSP/2.2
Server: GlassFish Server Open Source Edition 3.1.1
Content-Type: application/xml
Content-Length: 91
Date: Tue, 25 Sep 2012 14:34:22 GMT

<pwm>
  <resolution>16 bits</resolution>
  <cycle>65535</cycle>
  <dutyCycle>0</dutyCycle>
</pwm>
```

Fonte: Elaborada pelo autor.

Embora não tenha sido implementado, é possível que seja desenvolvido neste *gateway*, mecanismos de *cache* e controle de demanda, preservado o *gateway* de transdutores de requisições excessivas.

Outra possibilidade de extensão das funcionalidades deste *gateway* RESTful é o desenvolvimento de métodos de comunicação em JSON, além dos métodos em XML já implementados.

Na Figura 83 é mostrado a requisição de uma aplicação cliente, em HTTP pelo método POST, para alterar o *dutyCycle* do módulo PWM. Na Figura 84 é apresentada a resposta do serviço RESTful à requisição anterior.

O dado enviado do *dutyCycle* é enviado em XML. Também é possível enviar a estrutura XML completa do módulo PWM com o valor desejado para o *dutyCycle*. No

entanto, demais recursos mapeados para escrita no módulo, como o *cycle*, também serão reescritos.

Figura 83: Requisição POST ao serviço RESTful para alteração do *dutyCycle*.

```
POST /gatewayRESTful/webresources/pwm HTTP/1.1
Content-Type: application/xml
User-Agent: Java/1.7.0_02
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 28

<dutyCycle>40000</dutyCycle>
```

Fonte: Elaborada pelo autor.

Figura 84: Resposta do serviço RESTful à requisição de alteração do *dutyCycle*.

```
HTTP/1.1 204 No Content
X-Powered-By: Servlet/3.0 JSP/2.2
Server: GlassFish Server Open Source Edition 3.1.1
Content-Type: application/xml
Date: Tue, 25 Sep 2012 14:57:15 GMT
```

Fonte: Elaborada pelo autor.

Como o método de alteração do *dutyCycle* não retorna nenhum dado de confirmação, a resposta HTTP retorna o código 204 (*No Content*). No entanto, esta é apenas uma característica do serviço e não deve ser tratada como erro de comunicação.

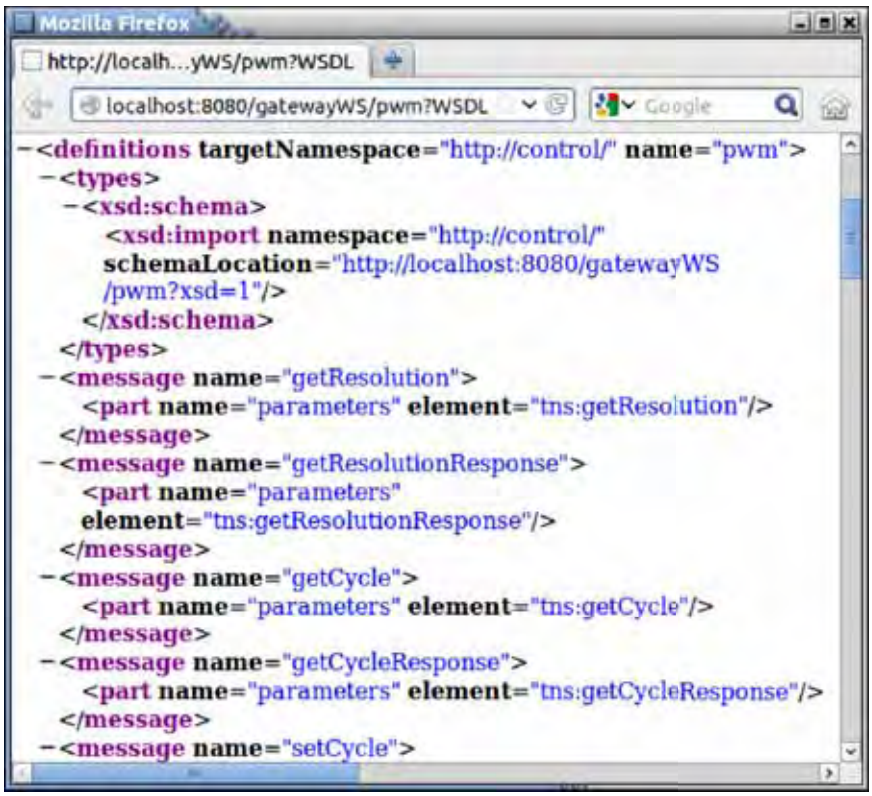
Assim, com este serviço, os transdutores inteligentes mapeados em XML no *gateway* de transdutores, são publicados como serviços RESTful completos na camada de serviços da arquitetura proposta.

6.4.2 *Gateway Web Service*

Outra proposta de tradução de padrões para a arquitetura para publicação e universalização do acesso a transdutores inteligentes é a adoção do padrão de *Web Services*. Para tanto, também foi utilizado o IDE Netbeans, alguns de seus assistentes e a tecnologia Java EE.

Na Figura 85 é apresentado parte do arquivo WSDL do *Web Service pwm*.

Figura 85: Arquivo WSDL do *Web Service pwm*.



Fonte: Elaborada pelo autor.

O Apêndice C apresenta o arquivo WSDL completo deste *Web Service pwm*.

Na implementação do *gateway Web Service*, para cada transdutor inteligente mapeado em XML, no *gateway* de transdutores, foi implementado um *Web Service* independente. Desta forma, para cada transdutor inteligente elevado ao padrão de *Web Service*, há um arquivo WSDL independente.

Na descrição do arquivo WSDL é possível identificar individualmente os métodos de leitura e escrita dos recursos de cada transdutor inteligente, o que não era possível no padrão RESTful.

Na Figura 86 é apresentada uma requisição HTTP da aplicação cliente, pelo método GET, enviada ao *Web Service pwm*. Na Figura 87 é mostrada a resposta do *Web Service* à requisição anterior.

Como pode ser observado na requisição, é solicitado o arquivo WSDL do *Web Service* pela aplicação cliente, no instante em que o objeto que trata a comunicação com o *Web Service* é instanciado. Como pode ser comprovado pela resposta, após o cabeçalho do protocolo HTTP são enviados os dados do arquivo WSDL.

Figura 86: Requisição HTTP ao *Web Service pwm*.

```
GET /gatewayWS/pwm?wsdl HTTP/1.1
User-Agent: Java/1.7.0_02
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Fonte: Elaborada pelo autor.

Figura 87: Resposta HTTP do *Web Service pwm*.

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.0
Server: GlassFish Server Open Source Edition 3.1.1
Content-Type: text/xml; charset=utf-8
Content-Length: 4314
Date: Tue, 25 Sep 2012 16:05:47 GMT

<?xml version='1.0' encoding='UTF-8'?>
<definitions xmlns[...] targetNamespace="http://control/" name="pwm">
 … trecho omitido do WSDL ...
</definitions>
```

Fonte: Elaborada pelo autor.

Na Figura 88 pode ser observada o envelope SOAP enviado pela aplicação cliente consumidora do *Web Service*, para o método *getDutyCycle()* do módulo PWM. Na Figura 89 é apresentada o envelope SOAP retornado pelo *Web Service* à requisição realizada anteriormente pela aplicação cliente.

Figura 88: Requisição SOAP ao método *getDutyCycle()* do *Web Service pwm*.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header/>
   <S:Body>
      <ns2:getDutyCycle xmlns:ns2="http://control/"/>
   </S:Body>
</S:Envelope>
```

Fonte: Elaborada pelo autor.

A resposta também é um envelope SOAP, o dado solicitado do *dutyCycle* (0) pode ser observado na *tag return*.

Figura 89: Resposta SOAP do método *getDutyCycle()* do *Web Service pwm*.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getDutyCycleResponse xmlns:ns2="http://control/">
         <return>0</return>
      </ns2:getDutyCycleResponse>
   </S:Body>
</S:Envelope>
```

Fonte: Elaborada pelo autor.

O processo de escrita no *Web Service* é realizado de forma semelhante à consulta exemplificada anteriormente. Na Figura 90 pode ser observado o envelope SOAP enviado ao *Web Service*, para o método *setDutyCycle()* do módulo PWM. Na Figura 91 é apresentada o envelope SOAP retornado pelo *Web Service* à requisição realizada.

Figura 90: Requisição SOAP ao método *setDutyCycle()* do *Web Service pwm*.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header/>
   <S:Body>
      <ns2:setDutyCycle xmlns:ns2="http://control/">
         <dutyCycle>40000</dutyCycle>
      </ns2:setDutyCycle>
   </S:Body>
</S:Envelope>
```

Fonte: Elaborada pelo autor.

Figura 91: Resposta SOAP do método *setDutyCycle()* do *Web Service pwm*.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:setDutyCycleResponse xmlns:ns2="http://control/">
         <return>true</return>
      </ns2:setDutyCycleResponse>
   </S:Body>
</S:Envelope>
```

Fonte: Elaborada pelo autor.

A requisição agora é enviada também com a *tag dutyCycle*, com o dado 40000 no seu interior, como também exemplificado no serviço RESTful. No entanto, neste caso, é retornado

o envelope SOAP com a confirmação de execução do método *setDutyCycle()*, pelo o dado *true* que pode ser observado na *tag return*.

Sem dúvidas este padrão de *Web Services* oferece mecanismos e recursos superiores ao padrão RESTful, como já discutido no capítulo 3. Na arquitetura proposta é um excelente padrão para se trabalhar como serviço de tradução em nuvem.

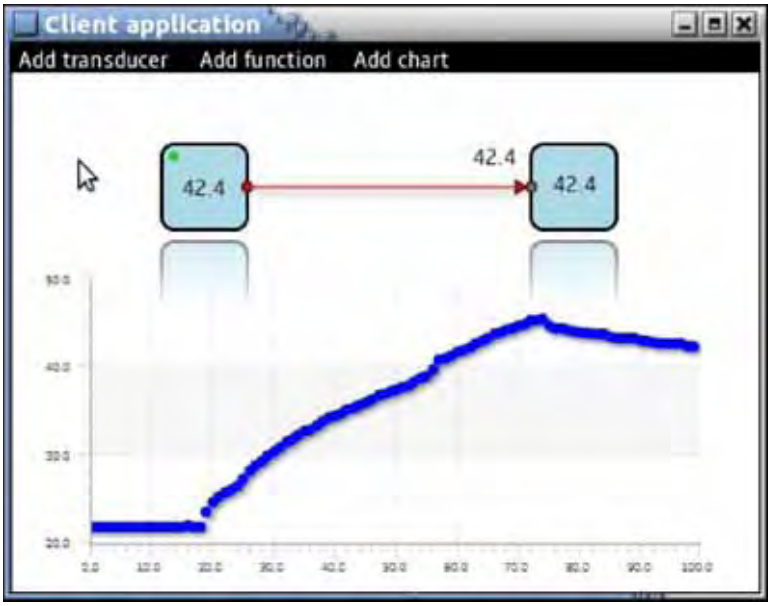
Assim, com este serviço, os transdutores inteligentes mapeados em XML no *gateway* de transdutores, são publicados como *Web Services* na camada de serviços da arquitetura para publicação e universalização do acesso a transdutores inteligentes.

6.4.3 Registro de dados

Para exemplificar uma possibilidade de serviço de retaguarda foi implementado um serviço de registro de dados RESTful, desenvolvido a partir de um *socket* em Java. Este serviço escuta requisições HTTP e registra parâmetros com o nome *log* que são passados por este protocolo. Como resposta, o serviço devolve o dado recebido estruturado em XML em uma *tag log*, no conteúdo de retorno do protocolo HTTP. Se necessário, o cliente do serviço pode confirmar que o dado foi registrado com êxito, pela cópia devolvida como resposta.

Na Figura 92 é apresentada a interface da aplicação cliente que interliga o transdutor inteligente *temperature* com o serviço de registro de dados.

Figura 92: Interface de aplicação cliente com serviço de registro de dados.



Fonte: Elaborada pelo autor.

O bloco do lado esquerdo realiza a entrada de dados, realizando requisições ao transdutor inteligente *temperature*. O bloco do lado direito recebe dados de temperatura em graus Celsius e os submete para o serviço de registro de dados, externo a aplicação.

Adicionalmente, a aplicação apresenta de forma gráfica o histórico das últimas cem amostras de dados. Estes dados apresentados de forma gráfica estão na aplicação cliente, diferente dos enviados para o serviço externo de registro de dados, que pode estar distribuído em qualquer outro local da rede.

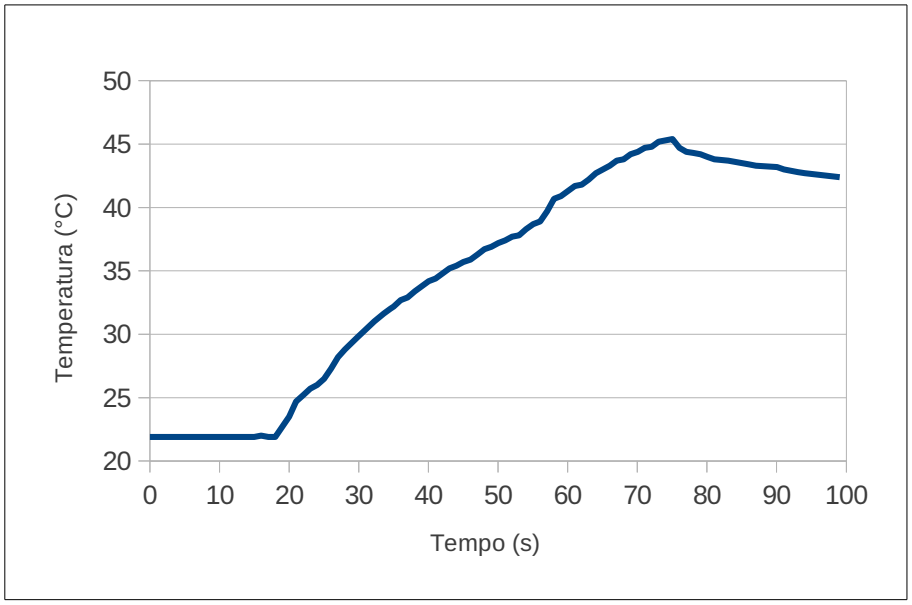
Na Figura 93 são apresentadas algumas linhas de dados registradas pelo serviço. Na Figura 94 é apresentado um gráfico com os dados registrados pelo serviço, no mesmo período de amostragem apresentado pelo gráfico da aplicação cliente.

Figura 93: Linhas de registros de temperatura.

```
43.0; 28 Sep 2012 10:57:57; 153746
42.8; 28 Sep 2012 10:57:59; 155747
42.7; 28 Sep 2012 10:58:00; 156748
42.4; 28 Sep 2012 10:58:05; 161750
```

Fonte: Elaborada pelo autor.

Figura 94: Gráfico com os dados de temperatura registrados.



Fonte: Elaborada pelo autor.

Os dados registrados estão separados em colunas pelo caractere “;”. Na primeira coluna podem ser observados alguns dados de temperatura em queda. Na segunda coluna o

serviço registra a data e hora do registro. Por fim, a terceira coluna registra o período em milisegundos a partir no primeiro dado enviado ao serviço, permitido melhor referência temporal do evento registrado.

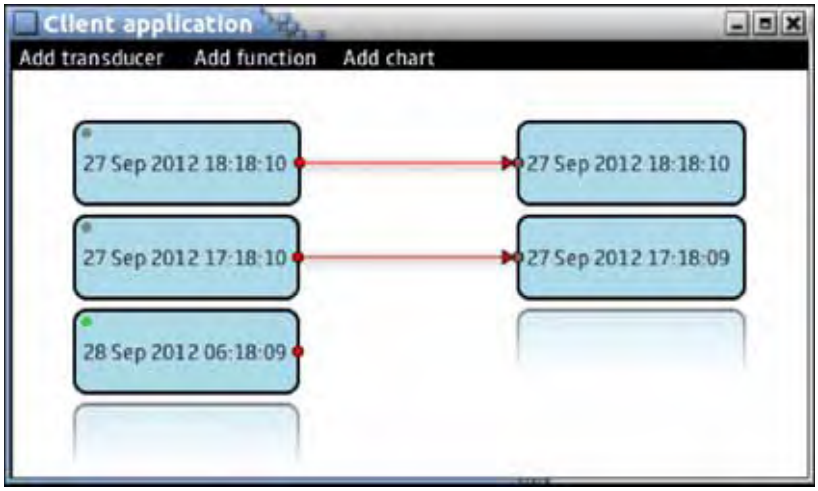
Após estabilidade da temperatura, por volta de 22 °C, o sensor de temperatura foi submetido a aquecimento por convecção forçada, até que ultrapassasse os 45 °C, quando foi devolvido à convecção natural com o ambiente.

Este foi apenas um exemplo de serviço de retaguarda que pode ser oferecido à arquitetura proposta, diversos outros serviços de armazenamento ou processamento também podem ser oferecidos em nuvem.

6.4.4 Serviços de terceiros

Para exemplificar mais uma possibilidade de integração e composição de aplicações, na Figura 95 é apresentada a interface da aplicação cliente com uma composição entre: transdutor inteligente, serviço de registro de dados e um serviço externo de terceiro.

Figura 95: Composição de aplicação com serviço de terceiro.



Fonte: Elaborada pelo autor.

O serviço de terceiro utilizado foi o *timezone* do EarthTools[28]. Este serviço RESTful recebe como parâmetros coordenadas geográficas (latitude e longitude) e devolve algumas informações da localidade passada como parâmetro. A informação escolhida para o exemplo

28 EarthTools é um *site* que fornece ferramentas em interface Web e serviços com informações de localidades geográficas como: coordenadas, horários do nascer e pôr do sol, elevação do nível do mar, horário, fuso horário etc.

de composição de aplicação foi a data e horário da localidade passada como parâmetro.

Os blocos do lado esquerdo estão enviando como parâmetro para o serviço *timezone*, pela URL do serviço, as coordenadas geográficas das cidades de Ilha Solteira, Campo Grande e Tóquio, respectivamente de cima para baixo. Observa-se que o ponto verde no canto esquerdo superior do terceiro bloco indica que a *thread* da requisição ainda não obteve resposta do serviço, o que justifica o horário apresentado neste bloco estar com um segundo de atraso.

Do lado direito da interface cliente, os blocos gráficos apresentados estão associados ao serviço de registro de dados e ao transdutor inteligente *lcd*, respectivamente de cima para baixo. Semelhande ao ocorrido na entrada de dados, o atraso no segundo bloco de saída de dados é justificado pelo atraso no recebimento da resposta do transdutor inteligente *lcd*, embora, neste caso, não exista nenhum indicador gráfico com esta informação.

Na Figura 96 são apresentadas algumas linhas de dados registrados no serviço de registro de dados e na Figura 97 é apresentada a imagem do LCD com o dado recebido.

Figura 96: Linhas de registros de dados do serviço *timezone*.

```
27 Sep 2012 18:18:07; 27 Sep 2012 18:18:13; 195550
27 Sep 2012 18:18:08; 27 Sep 2012 18:18:14; 196553
27 Sep 2012 18:18:09; 27 Sep 2012 18:18:15; 197551
27 Sep 2012 18:18:10; 27 Sep 2012 18:18:16; 198553
```

Fonte: Elaborada pelo autor.

Figura 97: Dados do *timezone* apresentados no LCD da placa microcontrolada.

Fonte: Elaborada pelo autor.

Assim como o registro de temperatura, os dados estão separados pelo caractere “;”. A primeira coluna apresenta dos dados do serviço *timezone* redirecionados pela aplicação principal ao serviço de registro de dados. A segunda coluna apresenta a data e horário local,

que, como pode ser observado, o horário local estava seis segundos adiantado do horário recuperado do serviço externo. Por fim, a terceira coluna apresenta o registro em milisegundos a partir do primeiro dado recebido pelo serviço de registros. Como também pode ser conferido, há pequenas variações de até três milisegundos nos instantes dos registros, algo perfeitamente aceitável tratando-se de mecanismos assíncronos sobre um barramento de usuários não determinístico.

Embora o dado enviado pela aplicação cliente, proveniente do serviço *timezone*, tenha excedido a quantidade de caracteres da primeira linha do LCD, pode-se observar que o dado do serviço externo chegou até o transdutor inteligente.

Mais uma vez, o apresentado reforça as evidências das diversas possibilidades para a comunicação entre quaisquer serviços disponíveis e transdutores inteligentes, em padrões universais como o RESTful, em uma arquitetura aberta como a proposta para publicação e universalização do acesso a transdutores inteligentes.

6.5 CONTROLE DE TEMPERATURA

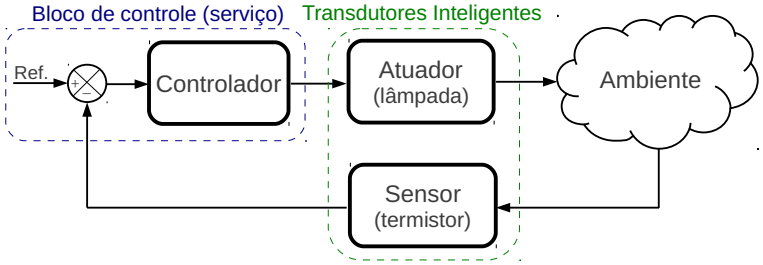
Como estudo de caso de uma aplicação efetiva foi escolhido o desenvolvimento de uma planta de controle de temperatura, como a de uma estufa ou forno controlado, de forma a explorar a arquitetura proposta.

A seguir são apresentados os transdutores utilizados na aplicação e seus mapeamentos no *gateway* de transdutores.

6.5.1 Transdutores inteligentes

A Figura 98 apresenta a planta em malha fechada de controle de temperatura do estudo de caso desenvolvido.

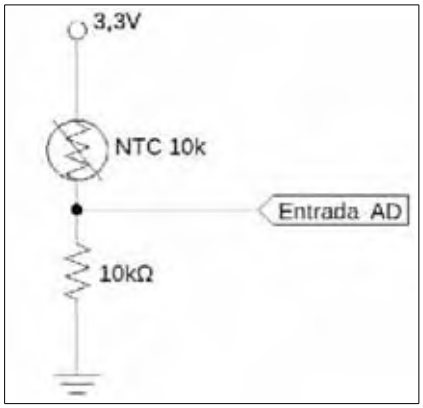
Figura 98: Planta em malha fechada de controle de temperatura.



Fonte: Elaborada pelo autor.

O sensor adotado para a leitura da temperatura foi um termistor NTC 10k, semelhante ao termistor *onboard* presente na plataforma microcontrolada adotada na seção anterior. Para tanto, este sensor foi condicionado em uma malha resistiva de divisão de tensão, de forma a explorar todo o *range* de 0 a 3,3 V da entrada do conversor AD da plataforma microcontrolada e inverter o sinal NTC do termistor. Este circuito de condicionamento de sinal é apresentado na Figura 99.

Figura 99: Circuito de condicionamento de sinal do termistor.



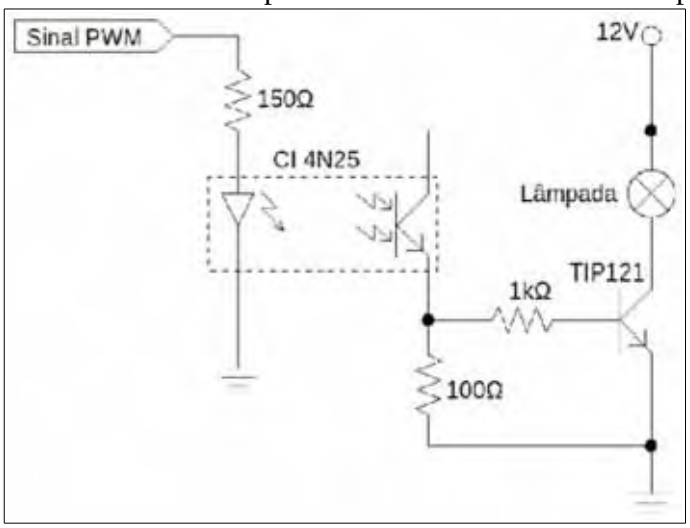
Fonte: Elaborada pelo autor.

A alimentação de 3,3V é a mesma do microcontrolador da placa, que recebe o sinal em uma de suas entradas AD. Os dados deste termistor externo foram mapeados na *tag thermometer,* pela reprogramação do canal AD desta *tag* no microcontrolador. Outra possibilidade é acessar os dados do termistor diretamente pelo mapeamento dos canais AD na *tag ADC*, sem o tratamento de dados realizados para a *tag temperature*.

Como elemento de atuação na planta de controle de temperatura foi utilizada a saída PWM mapeada na placa DB-DP11115 para acionar uma lâmpada DC de 12V e 21W, para aquecer o ambiente em que o termistor estava inserido. Para tanto, a saída PWM do microcontrolador foi acoplada à lâmpada por meio do circuito de potência apresentado na Figura 100.

O sinal PWM gerado pelo microcontrolador é ligado ao LED do acoplador óptico 4N25. A referência do LED é a mesma da placa microcontrolada do *gateway* de transdutores. Já a referência do lado direito deste circuito de potência é a da fonte de tensão que supre os 12V para o coletor do foto transistor e para a lâmpada. O emissor do foto transistor tem a função de chavear o TIP121 pelo controle do sinal na base deste TJB (Transistor de Junção Bipolar).

Figura 100: Circuito de potência de acionamento da lâmpada.



Fonte: Elaborada pelo autor.

Desta forma, a potência da lâmpada pode ser acionada de forma contínua pela saída PWM, com resolução de 16 bits. Diferente do tratamento de dados feito no condicionamento do sinal do termistor, neste caso a saída PWM ficou mapeada apenas em função do controlador PWM do microcontrolador e não do transdutor (lâmpada) associado a ele. Essa abordagem diferenciada fez com que a necessidade de correção de escala fosse transferida para as camadas superiores da arquitetura proposta, simplificando a implementação desta camada.

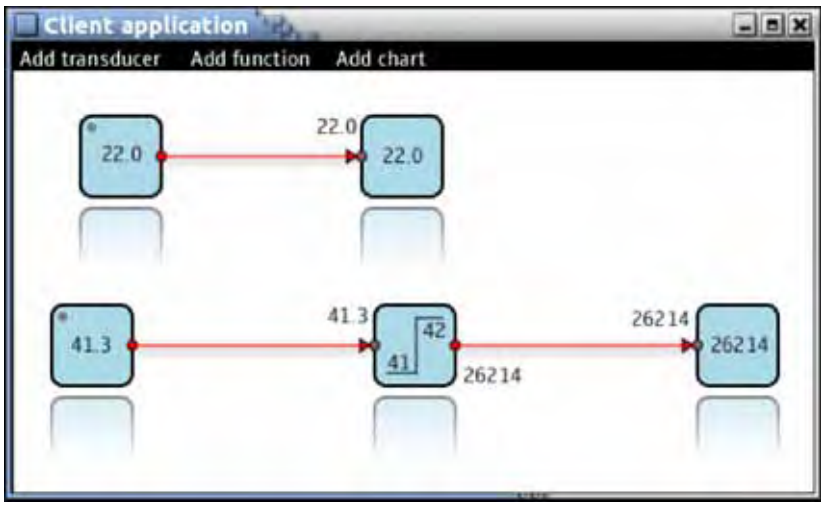
Neste ponto, concluiu-se a implementação de elementos das duas primeiras camadas da arquitetura proposta, restando agora o desenvolvimento de elementos em software para a efetiva implementação da planta de controle de temperatura. O condicionamento e acoplamento dos transdutores referem-se à primeira camada, embora neste caso não se tenha um barramento de campo.

6.5.2 Resultados

A composição da aplicação de controle *on/off* de temperatura pode ser observada pela interface da aplicação cliente com controle *on/off* de temperatura, apresentada na Figura 101.

Nesta aplicação foram utilizados dois *gateways* de transdutores, no primeiro o transdutor inteligente *thermometer* refere-se ao termistor externo, no segundo *gateway* o transdutor inteligente *thermometer* refere-se a um termistor *onboard*, que foi utilizado para monitorar a temperatura ambiente.

Figura 101: Interface da aplicação cliente com controle *on/off* de temperatura.



Fonte: Elaborada pelo autor.

Nos blocos superiores é realizada a interligação do transdutor inteligente *thermometer,* do segundo *gateway* de transdutores, enviando o dado da *tag temperature* para o serviço de registro de dados.

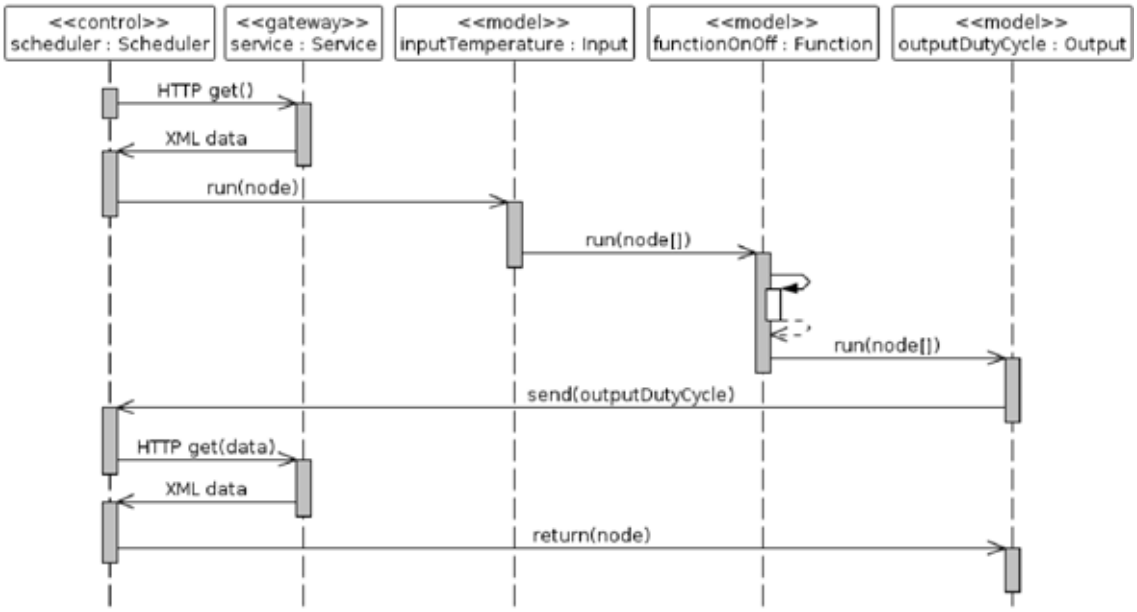
Nos blocos inferiores, a temperatura do bloco de entrada de dados é lida a partir do transdutor inteligente *thermometer*, do primeiro *gateway* de transdutores. O dado da *tag temperature* é enviado para um bloco de função *on/off* que, após sua ação, entre 41 °C e 42 °C, sua saída é transmitida para um bloco de saída de dados. O bloco de saída de dados é associado à lâmpada por meio da saída PWM, do mesmo gateway de transdutores que monitora a temperatura ambiente.

A saída do bloco de função *on/off* poderia ter sido associado a uma saída de dados digital, como os mapeamentos *digitalIOs* dos *gateways* de transdutores. Desta forma o circuito de potência poderia ligar e desligar a lâmpada conforme sinal enviado. No entanto, devido à possibilidade de controle de potência da lâmpada preferiu-se a saída PWM. Após alguns testes definiu-se o *dutyCycle* em 26214 (40% ativo), para a saída *off* do controlador, e 0 (0% ativo), para a saída *on*.

Nestes blocos inferiores, os dados da temperatura controlada e da saída do controlador *on/off* foram registrados pela própria aplicação cliente. Pensou-se na possibilidade de também enviá-los para o serviço de registro de dados. No entanto, verificou-se que a forma em que os blocos da aplicação estavam implementados, não permitia derivação do sinal para outras saídas de dados. Desta forma, tais dados foram registrados pela aplicação principal, ficando o desenvolvimento de divisões de sinal como sugestão de desenvolvimento futuro.

O diagrama de sequência apresentado na Figura 102 detalha o fluxo de dados desta aplicação desenvolvida, de controle *on/off* de temperatura.

Figura 102: Diagrama de sequência da aplicação com controle *on/off*.



Fonte: Elaborada pelo autor.

Para cada período de *refresh* o objeto *scheduler* dispara o fluxo de dados com uma requisição ao *gateway* de transdutores. Assim que o *scheduler* recebe a resposta assíncrona do *gateway*, o *scheduler* invoca o método *run()* do objeto *inputTemperature*, passando toda a árvore de objetos da classe *Node*.

O objeto *inputTemperature* recupera a folha da árvore *node* associação à *tag temperature*, do termistor, e a repassa ao objeto *functionOnOff*, por meio de seu método *run()*.

O objeto *functionOnOff* decide seu estado de saída em função do dado recebido e repassa o resultado ao objeto *outputDutyClycle*, que está associado à *tag dutyCycle* do módulo PWM. Exceto pelo processamento dos mecanismos de comunicação e extração de dados, este também é o único momento que a aplicação realiza uma manipulação de dados, na tomada de decisão da função *on*/off.

Neste caso, o objeto *outputDutyCycle* envia sua referência ao *scheduler* que, além de enviar o dado contido em *outputDutyCycle* para o módulo PWM, ainda devolve o retorno em uma árvore de objetos da classe *Node*, pela referência do objeto *outputDutyCycle* recebido, invocando o método *return()* deste objeto. Este retorno ao objeto *outputDutyCycle* permite a confirmação de sucesso do dado enviado.

Na Figura 103 é apresentado um gráfico confeccionado com os dados registrados pelo serviço de registro de dados e pela aplicação cliente composta para o controle *on/off* de temperatura.

O experimento foi realizado em ambiente com temperatura controlada. No gráfico pode ser observado que a ação de controle sobre a temperatura ambiente também foi do tipo *on/*off, com períodos de atuação bem superiores aos da aplicação desenvolvida, mantendo a temperatura ambiente por volta de 22 °C.

Antes da aplicação de controle entrar em operação a temperatura do termistor externo estava por volta de 24 °C. A saída PWM foi acionada com 40% de ciclo ativo. Em pouco mais de 2 minutos a temperatura controlada entrou na faixa de tolerância (41,5°C±1,2%) e assim ela foi mantida por todo o período seguinte amostrado.

No gráfico apresentado, pode ser observado claramente os instantes de atuação do controlador on/off, com 0% e 40% da saída PWM, com consequente queda e ascensão da variável controlada, respectivamente.

A composição da aplicação de controle contínuo de temperatura pode ser observada na interface da aplicação cliente com controle PID de temperatura, apresentada na Figura 104.

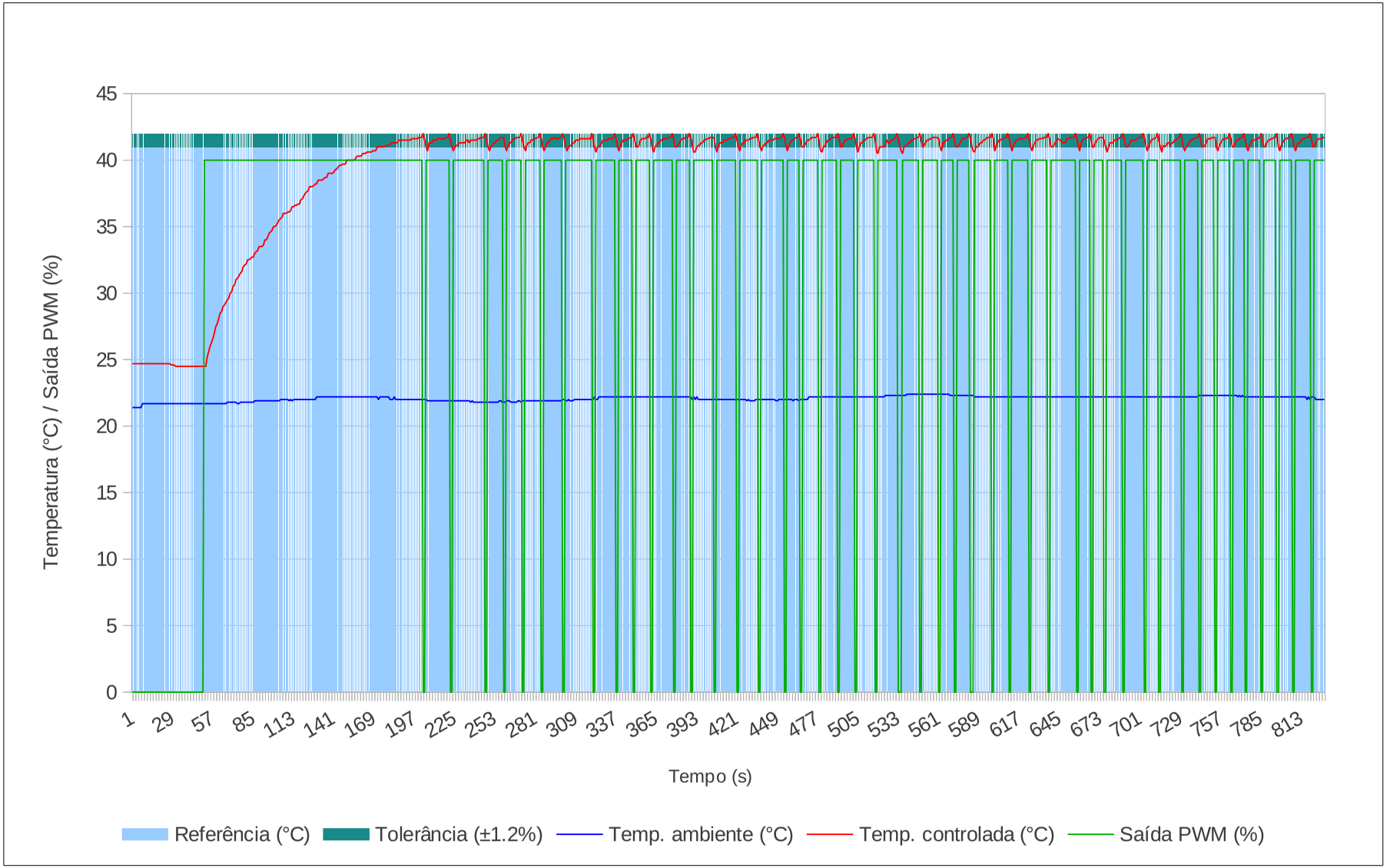
Nesta aplicação também foram utilizados dois *gateways* de transdutores, no primeiro o transdutor inteligente *thermometer* refere-se ao termistor externo, no segundo *gateway* o transdutor inteligente *thermometer* refere-se a um termistor *onboard,* utilizado para monitorar a temperatura ambiente.

Assim como na aplicação *on/off,* nos blocos superiores é realizada a interligação do transdutor inteligente *thermometer*, do segundo *gateway* de transdutores, enviando o dado da *tag temperature* para o serviço de registro de dados.

Nos blocos inferiores, a temperatura do bloco de entrada de dados é lida a partir do transdutor inteligente *thermometer*, do primeiro *gateway* de transdutores. O dado da *tag temperature* é enviado para um bloco de função PID com *setpoint* definido em 42 °C que, após sua ação, é transmitido para um bloco de ganho. Após a aplicação do ganho o dado é enviado para o bloco de saída de dados, que está associado a uma saída PWM, no segundo *gateway* de transdutores.

O controlador PID foi sintonizado empiricamente. Para tanto, foram realizados alguns testes, alterando os parâmetros *Kp*, *Ki* e *Kd*, até que se obtivessem tempos de resposta e sobre sinal aceitáveis para a aplicação. O bloco de ganho foi utilizado para ajustar a escala de 0 a 100 da saída do controlador PID para 0 a 65.535 que é o *range* de trabalho do módulo PWM.

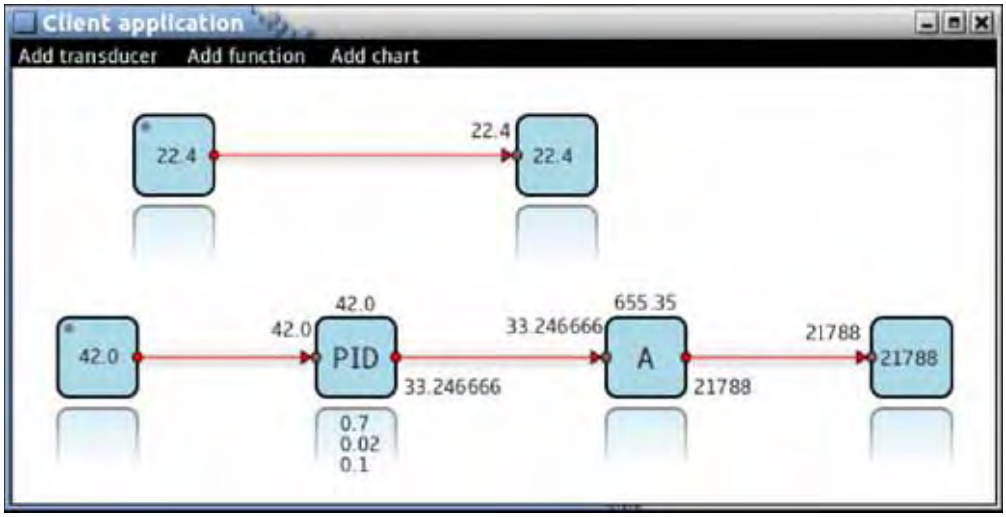
Figura 103: Gráfico com dados do controle *on/off* de temperatura.



Fonte: Elaborada pelo autor.

Para tanto, o sinal recebido pela função de ganho foi multiplicado por 655,35. O bloco de ganho também teve a função de truncar o dado de saída antes que o mesmo fosse enviado ao bloco de saída de dados, pois o dado da propriedade *dutyCycle* do mapeamento *pwm* deve ser um valor inteiro.

Figura 104: Interface da aplicação cliente com controle PID de temperatura.



Fonte: Elaborada pelo autor.

Nestes blocos inferiores, os dados da temperatura controlada e da saída do controlador PID também foram registrados pela própria aplicação cliente.

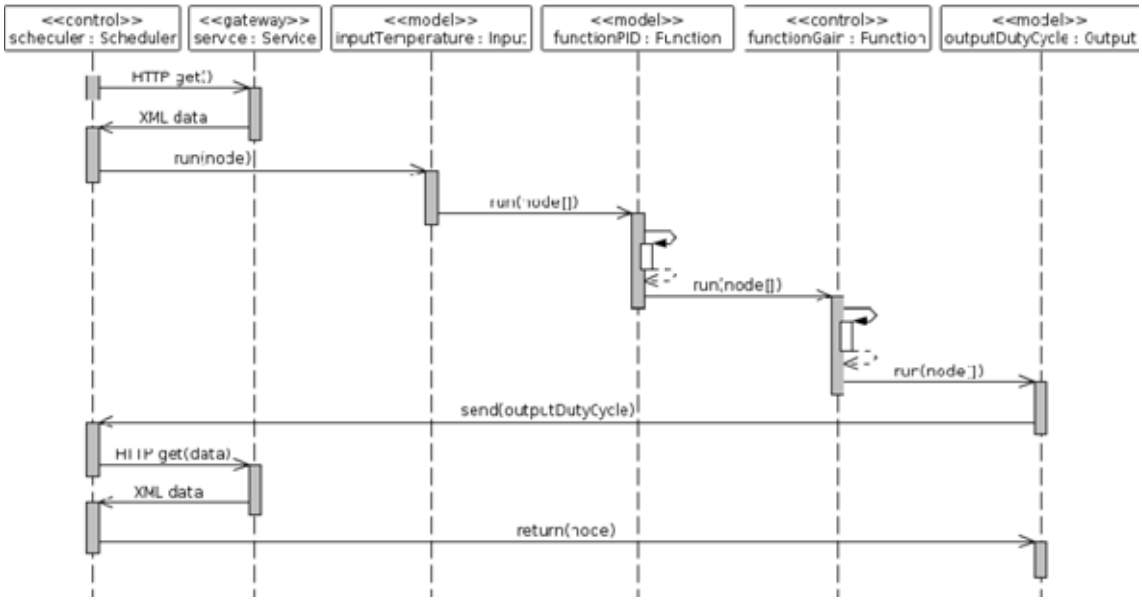
O diagrama de sequência apresentado na Figura 105 detalha o fluxo de dados desta aplicação desenvolvida de controle PID de temperatura.

Para cada período de *refresh* o objeto *scheduler* dispara o fluxo de dados com uma requisição ao *gateway* de transdutores. Assim que o *scheduler* recebe a resposta assíncrona do *gateway*, o *scheduler* invoca o método *run()* do objeto *inputTemperature*, passando toda a árvore de objetos da classe *Node*.

O objeto *inputTemperature* recupera a folha da árvore *node* associação à *tag temperature*, do termistor, e a repassa ao objeto *functionPID*, por meio de seu método *run()*.

O objeto *functionPID* recalcula sua saída em função do dado recebido e repassa o resultado ao objeto output*Gain* que multiplica e trunca o dado recebido por 655,35, repassando sua saída ao objeto *outputDutyCycle*, que está associado à *tag dutyCycle* do módulo PWM. Neste caso, ocorre processamento de dados tanto pelo objeto *functionPID* quanto pelo objeto *functionGain*, exceto também pelo processamento dos mecanismos de comunicação e extração de dados.

Figura 105: Diagrama de sequência da aplicação com controle PID de temperatura.



Fonte: Elaborada pelo autor.

Neste caso, o objeto *outputDutyCycle* envia sua referência ao *scheduler* que, além de enviar o dado contido em *outpuDutyCycle* para o módulo PWM, ainda devolve o retorno em uma árvore de objetos da classe *Node*, pela referência do objeto *outputDutyCycle* recebido, invocando o método *return()* deste objeto. Este retorno ao objeto *outputDutyCycle* permite a confirmação de sucesso do dado enviado.

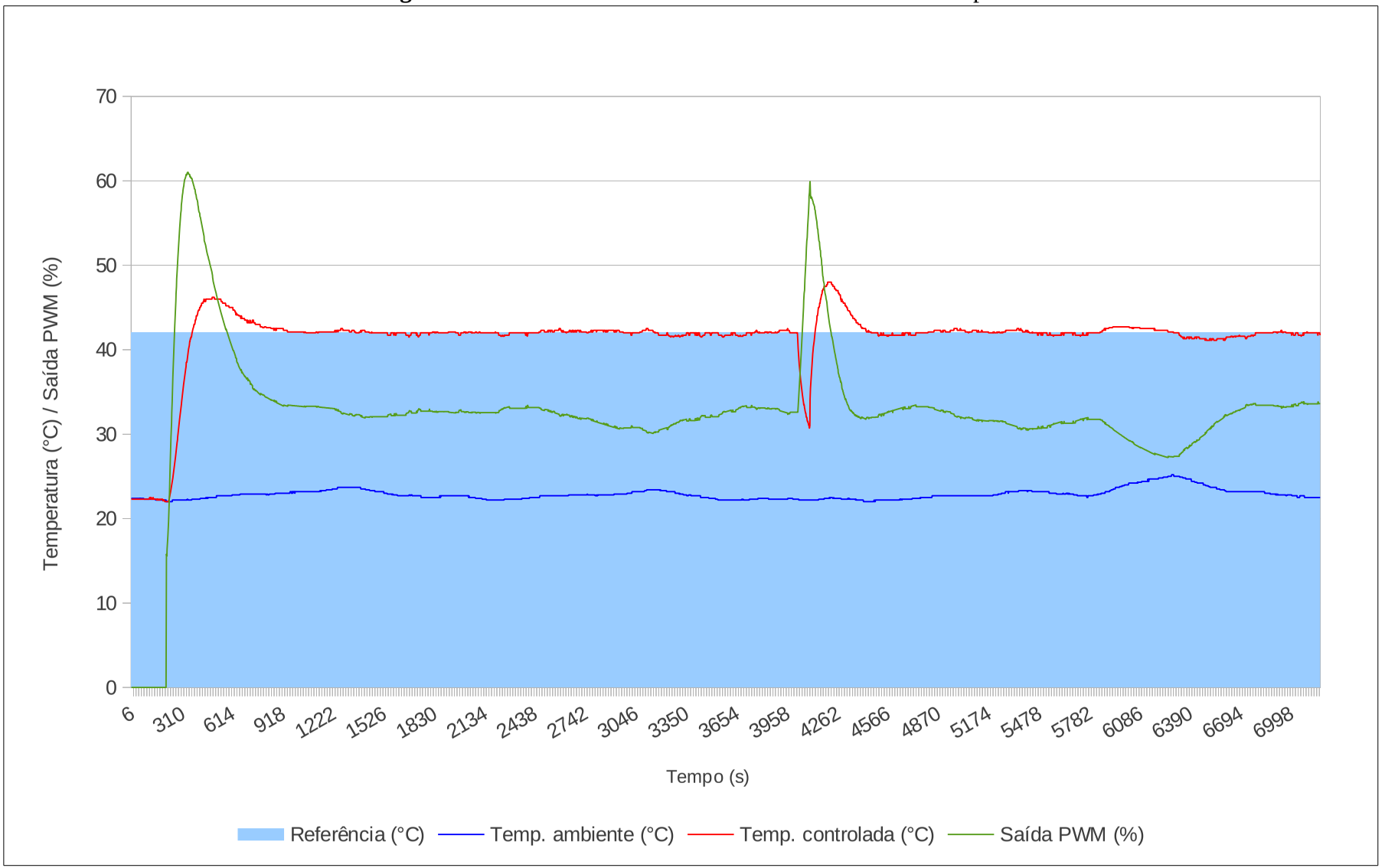
Na Figura 106 é apresentado um gráfico confeccionado com os dados registrados pelo serviço de registro de dados e pela aplicação cliente composta para o controle contínuo de temperatura, com controlador PID.

Este experimento também foi realizado em ambiente com temperatura controlada, por volta de 22 °C. No entanto, neste caso foram provocadas perturbações na temperatura ambiente para verificar a resposta do controlador PID durante as perturbações.

Antes da aplicação de controle entrar em operação a temperatura do termistor externo estava por volta de 22 °C, a mesma que a ambiente. No gráfico apresentado, pode ser observado o instante de acionamento do sistema, com o aumento contínuo da saída PWM de 0% até pouco mais de 60%, quando atinge o pico do sobre sinal e inverte sua inclinação.

A ação de controle também provoca sobre sinal da variável controlada, chegando a 46 °C até que inverta a inclinação da curva. Na sequência a temperatura controlada decai até entrar em regime permanente. Nota-se pela curva da saída PWM que a atuação do controlador PID é constante a fim de manter estabilizada a variável controlada.

Figura 106: Gráfico com dados do controle contínuo de temperatura.



Fonte: Elaborada pelo autor.

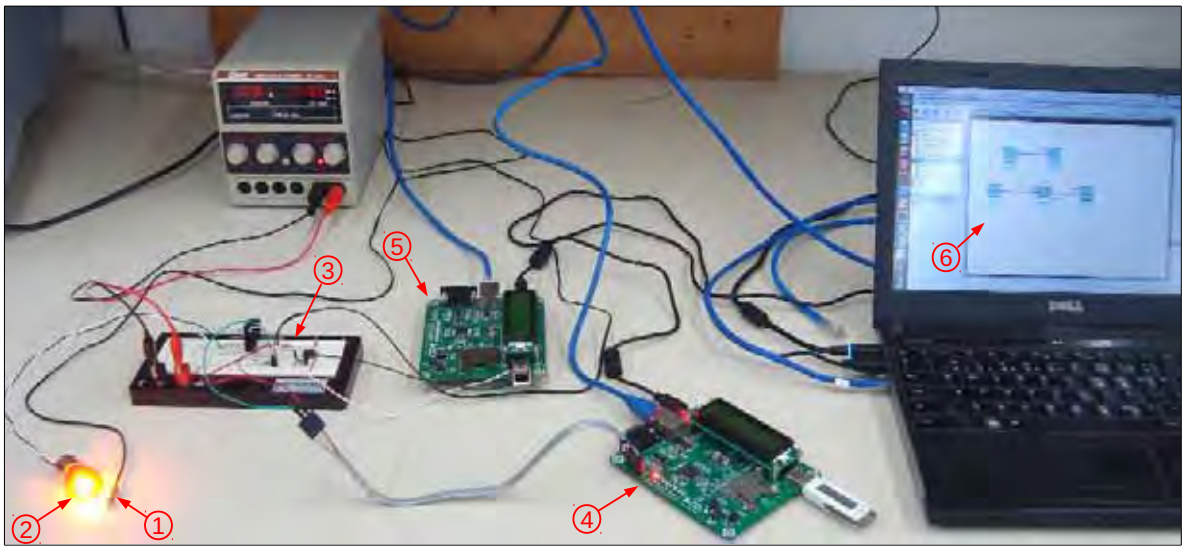
Por volta de 4.000 segundos de operação da aplicação de controle contínuo de temperatura, foi criada uma perturbação no sistema, afastando-se o atuador (lâmpada) do sensor (termistor) de temperatura.

Com a perturbação provocada por vota dos 4.000 segundos, a temperatura do termistor começou a cair rapidamente, enquanto o controlador tentava aumentar a saída PWM para compensar a queda de temperatura identificada. Após alguns segundos, a lâmpada foi devolvida à posição original de forma a permitir novamente a transferência de calor ao termistor, o que permitiu novamente que o sistema pudesse efetivamente voltar a controlar a temperatura, até que novamente a estabilizasse.

Por volta dos 6.000 segundos foi provocada outra perturbação no ambiente. Desta vez o sistema de refrigeração do ambiente foi desligado, fazendo com que a temperatura ambiente subisse. Pode-se observar nas curvas da temperatura do termistor e da saída PWM que a temperatura controlada começa a subir, o que é imediatamente compensado pelo controlador PID na saída PWM a fim de corrigir a variação. Na sequência o sistema de refrigeração do ambiente é novamente ligado o que gera outra perturbação, no sentido contrário. Mais uma vez se percebe a correção automática do sistema até levar novamente a variável controlada ao regime permanente.

Na Figura 107 são apresentados e identificados os módulos utilizados nestas aplicações de controle de temperatura.

Figura 107: Módulos da aplicação de controle de temperatura.



Fonte: Elaborada pelo autor.

Os elementos identificados são:

1 – termistor externo;

2 – lâmpada;

3 – *protoboard*, com o circuito de condicionamento de sinal do termistor externo e circuito de potência de acionamento da lâmpada;

4 – primeiro *gateway* de transdutores, interligado ao circuito de condicionamento de sinal do termistor externo;

5 – segundo *gateway* de transdutores, interligado ao circuito de potência de acionamento da lâmpada. Também é utilizado o termistor *onboard* deste *gateway* para fazer a aquisição da temperatura ambiente;

6 – aplicação cliente em execução.

O sinal do termistor externo é condicionado pelo circuito de condicionamento de sinal implementado no *protoboard* e ligado a uma das entradas AD do primeiro *gateway* de transdutores.

A saída PWM do segundo *gateway* de transdutores é ligada ao circuito de potência implementado no mesmo *protoboard* que está o circuito de condicionamento do sinal do termistor. O circuito de potência por sua vez está ligado à lâmpada, realizando seu acionamento.

A aplicação cliente realiza a composição da aplicação, com controle *on/off* ou PID, realizando a leitura dos dados de temperatura e enviando dados para o acionamento da lâmpada pelo módulo PWM, conforme composições descritas anteriormente.

6.6 SUPERVISÃO E AQUISIÇÃO DE DADOS DE UMA PLANTA DE CONTROLE DE PROCESSOS

Como segundo estudo de caso de uma aplicação efetiva, sobre a arquitetura proposta, foi realizada a supervisão e aquisição de dados de uma planta didática de controle de processos.

6.6.1 Planta didática de controle de processos

Na Figura 108 é apresentada a planta didática de controle de processos da Prosys Engenharia.

Figura 108: Planta didática de controle de processos.

Fonte: Elaborada pelo autor.

Este equipamento possui um reservatório inferior de 100 L e um reservatório superior de 75 L. Os reservatórios são interligados entre si e com outras tubulações que passam por uma bomba centrífuga, acoplada a um motor elétrico trifásico de ½ CV. Nas tubulações existem algumas válvulas manuais, uma válvula proporcional acoplada a um atuador elétrico de 24 V e um aquecedor de passagem de 5 kW (PROSYS, 2011).

Além dos atuadores descritos, este equipamento apresenta dois transmissores de temperatura, um transmissor de vazão e um transmissor de pressão, além de indicadores

digitais e analógicos de temperatura, pressão, vazão e nível. O transmissor de pressão pode ser utilizado para indicar ou transmitir, indiretamente, o nível do reservatório superior.

Os transmissores operam no padrão 4 a 20 mA e estão ligados ao painel elétrico do equipamento. No painel elétrico do equipamento estão, entre outros dispositivos, um inversor de frequência, para acionamento do motor elétrico trifásico acoplado à bomba centrífuga, um controlador de potência, para acionamento do aquecedor de passagem, e um CLP.

O CLP proporciona o controle lógico da planta, executando o software fornecido pelo fabricante do equipamento. A aplicação do CLP é acionada e parametrizada por uma aplicação supervisória, também fornecida pelo fabricante do equipamento. A aplicação supervisória deve ser executada em uma máquina com o Elipse E3 Viewer, que se comunica com o CLP, por sua interface RS485, utilizando o protocolo Modbus RTU (*Remote Terminal Unit*) (PROSYS, 2011).

Neste estudo de caso foi escolhida a aplicação de controle de nível do reservatório superior do equipamento, que utiliza indiretamente o sinal de pressão do transmissor de pressão para atuar na bomba centrífuga.

Para tanto, pela interface da aplicação supervisória, o CLP foi parametrizado e ligado em modo automático. Neste modo o CLP recalcula a cada um segundo a saída do controlador PID, em função da variável controlada e dos parâmetros configurados.

Para configurar o CLP foi utilizado uma das sugestões de parâmetros do fabricante do equipamento, com constante proporcional igual a seis (6), integral igual a oito (8) e derivativa igual a zero (0), para um *setpoint* de 0,35 mca.

A válvula proporcional foi mantida totalmente aberta e o aquecedor de passagem foi desligado.

Embora a aplicação supervisória do equipamento permita parametrizar, acionar e monitorar as variáveis do processo, ela não oferece função de aquisição e registro dos dados do processo controlado.

Para tanto, foi utilizado o *gateway* de transdutores e a aplicação cliente desenvolvida sobre a arquitetura proposta, para realizar a aquisição dos dados gerados pela aplicação de controle de nível.

6.6.2 Transdutores inteligentes

Para realizar a amostragem dos sinais do transmissor de pressão e do transmissor de

vazão foram utilizadas duas entradas AD de um *gateway* de transdutores. Embora os transmissores operassem em 4 a 20 mA, verificou-se nas entradas do CLP níveis de tensões proporcionais ao *range* dos transmissores, realizando-se apenas adequações das tensões aplicadas ao ADC por malhas resistivas de queda de tensão.

A correção de escala e ajuste de *offset* dos sinais amostrados foram deixados para serem tratados na aplicação cliente.

A aquisição dos dados de operação da bomba centrífuga foi realizada amostrando o sinal de acionamento enviado ao inversor de frequência. O inversor de frequência estava configurado para operar no modo 4 a 20 mA. No entanto, também se verificou na saída do CLP níveis de tensões proporcionais à curva de operação da bomba centrífuga. Neste caso, os níveis de tensão verificados já eram adequados para a entrada do ADC do *gateway* de transdutores, que foi ligado diretamente ao sinal de acionamento do inversor de frequência.

Desta forma, o sinal de acionamento da bomba centrífuga foi publicado por um segundo *gateway* de transdutores, sendo que os sinais dos transmissores de pressão e vazão já haviam sido publicados pelo primeiro *gateway*.

6.6.3 Resultados

Na Figura 109 é apresentado o painel elétrico da planta de controle de processos com os *gateways* de transdutores ligados aos sinais dos transmissores de pressão e vazão e ao sinal de acionamento do inversor de frequência.

Os elementos identificados são:

1 – *gateways* de transdutores;

2 – *protoboard*, com os circuitos de condicionamento de sinais dos transmissores e circuito de potência para a saída PWM;

3 – inversor de frequência;

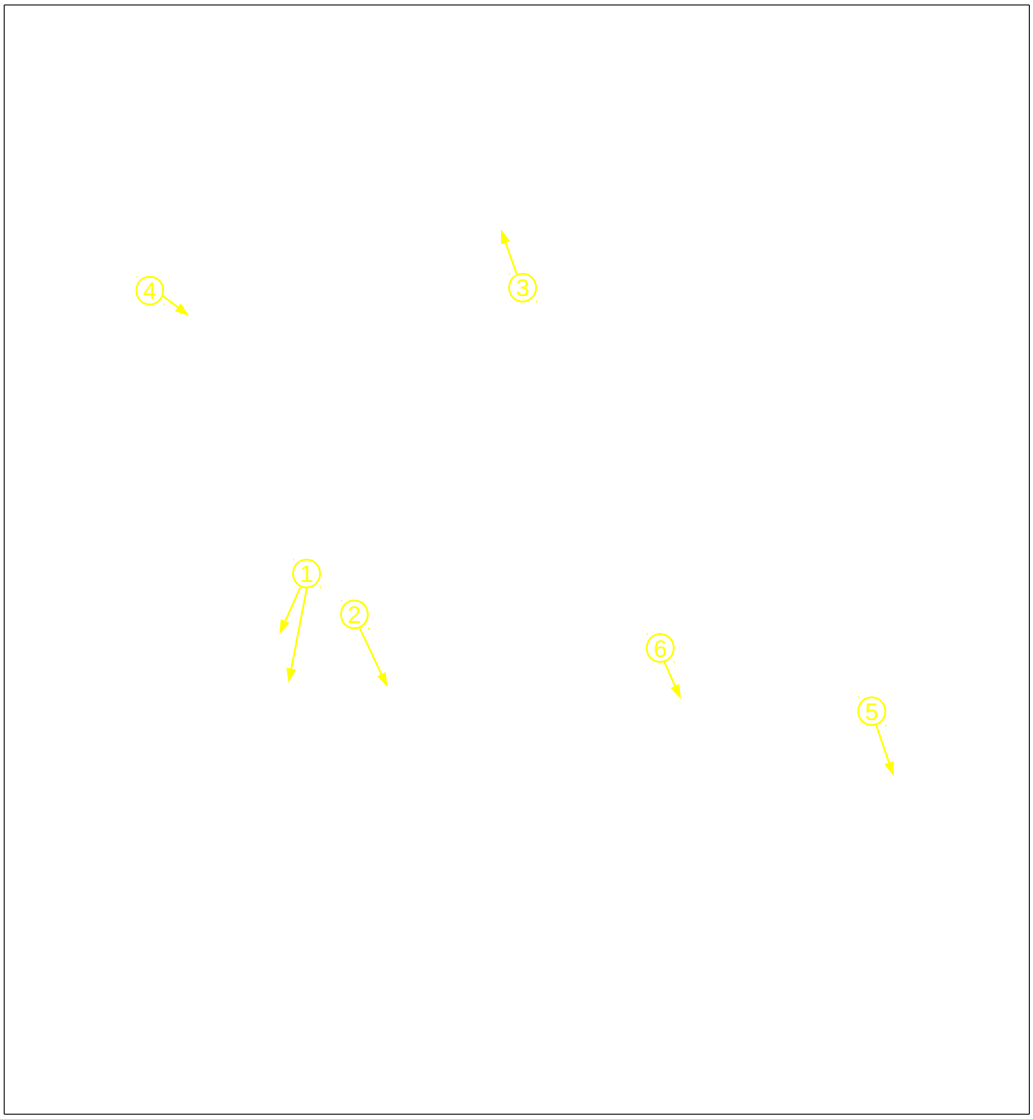
4 – CLP;

5 – aplicação supervisória no Elipse E3 Viewer;

6 – aplicação cliente composta sobre a arquitetura proposta.

A aplicação cliente rodando em uma das máquinas foi composta para fazer o registro local dos dados dos transmissores de pressão e vazão e do sinal de acionamento da bomba centrífuga.

Figura 109: Painel elétrico da planta de controle de processos.



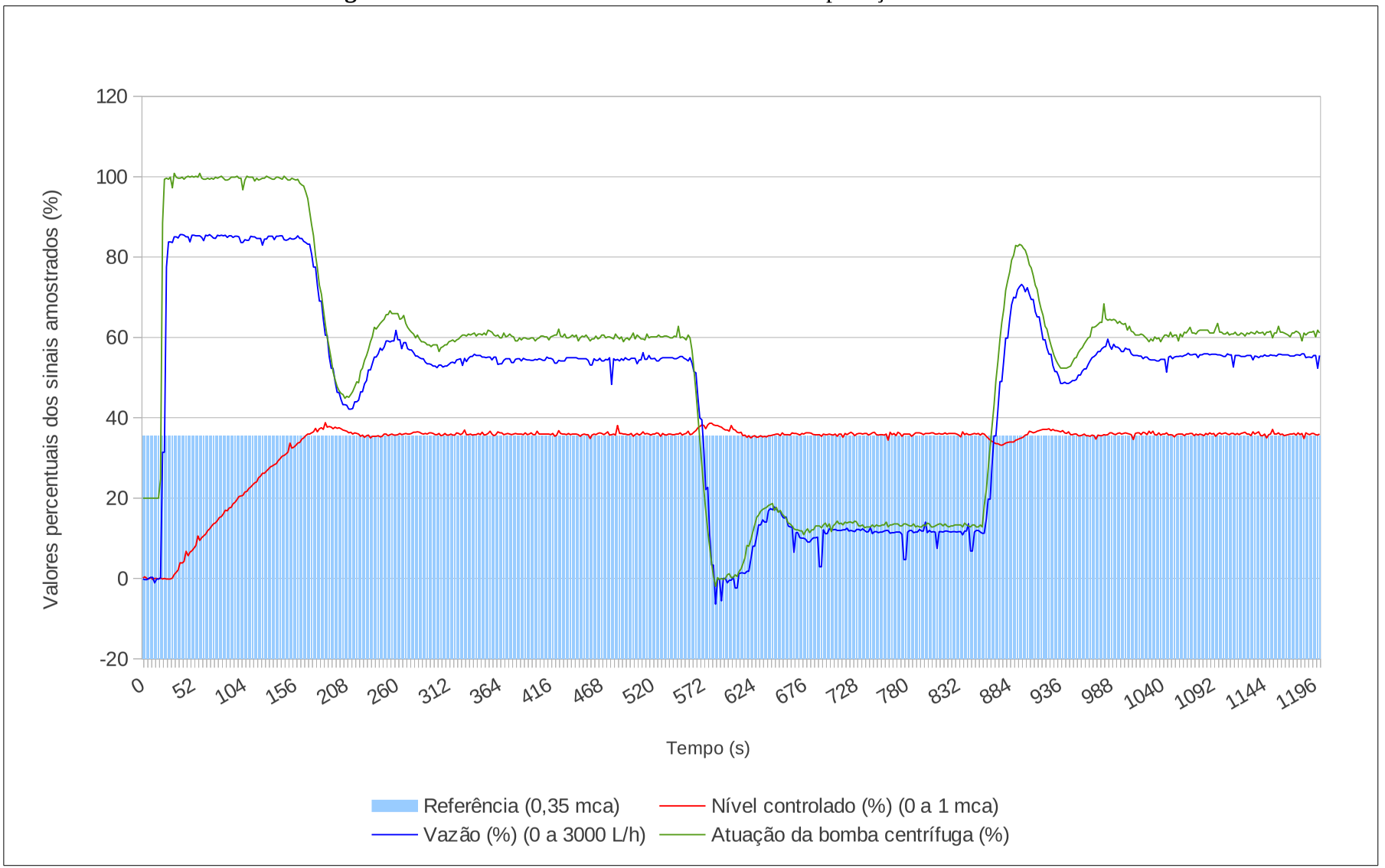
Fonte: Elaborada pelo autor.

Pela aplicação supervisória o equipamento foi ligado em modo manual de operação. A válvula proporcional foi aberta totalmente, a bomba centrífuga foi ligada com atuação mínima (20%) e os parâmetros do controlador PID foram configurados em 6, 8 e 0, para as ações, proporcional, integral e derivativa, respectivamente. O *setpoint* foi definido em 0,35 mca, quando o equipamento foi passado para o modo automático.

Na Figura 110 é apresentado um gráfico confeccionado com os dados registrados pela aplicação cliente, a partir dos sinais amostrados do processo de controle de nível.

Ao colocar o controlador em modo automático, o mesmo eleva o sinal de acionamento da bomba centrífuga ao valor máximo, durante todo o tempo de subida.

Figura 110: Gráfico com os sinais amostrados da aplicação de controle de nível.



Fonte: Elaborada pelo autor.

Neste momento, a bomba centrífuga bombeava a água presente no reservatório inferior para o reservatório superior. A água do reservatório superior voltava para o reservatório inferior por gravidade, por uma tubulação com uma válvula manual deixada totalmente aberta. Como a vazão proporcionada pela bomba centrífuga era maior que a vazão gravitacional, o reservatório superior foi se enchendo.

A vazão de água proporcionada pela bomba centrífuga acompanhou quase que diretamente a ação da bomba, exceto em oscilações mais bruscas da bomba centrífuga, onde se percebe pequeno atraso na respectiva alteração da vazão. A vazão gravitacional não era instrumentada nesta planta de controle de processos.

Com o início do *overshoot* o controlador reduziu rapidamente o sinal de acionamento da bomba centrífuga, na tentativa de corrigir o nível.

Aparentemente, a variável controlada se estabiliza a partir dos 250 segundos, no entanto, observando o sinal de acionamento da bomba e a vazão, percebe-se que a estabilidade só é obtida a partir dos 380 segundos.

Por volta dos 580 segundos a válvula manual que permitia a vazão gravitacional foi fechada parcialmente, provocando a perturbação apresentada no gráfico neste instante.

O sistema volta a se estabilizar por volta dos 750 segundos. Por volta dos 880 segundos a válvula manual anterior volta a ser aberta totalmente, provocando nova perturbação ao sistema.

Mais uma vez, como pode ser observado no gráfico da Figura 110, após algum período transitório a variável controlada volta a se estabilizar, assim como o sinal de acionamento da bomba centrífuga e a vazão por ela provocada.

6.7 SUBSTITUIÇÃO DO CONTROLADOR DE UMA PLANTA DE CONTROLE DE PROCESSOS

Neste terceiro estudo de caso de uma aplicação efetiva, sobre a arquitetura proposta, foi realizada a substituição do controlador (CLP) da planta de controle de processos utilizada no estudo de caso anterior. Neste estudo de caso foi adotada a mesma aplicação de controle de nível da seção anterior.

O CLP da planta de controle de processos foi substituído por um bloco PID em software, em uma aplicação cliente composta sobre a arquitetura para publicação e universalização do acesso a transdutores inteligentes.

6.7.1 Transdutores inteligentes

Os sinais dos transmissores de pressão e vazão já estavam mapeados na *tag ADC* do primeiro *gateway* de transdutores, conforme desenvolvimento descrito na seção anterior.

No entanto, foi necessário atuar na bomba centrífuga e não mais apenas fazer a aquisição do sinal de acionamento, anteriormente gerado pelo CLP.

Para acionar a bomba centrífuga foi utilizada a saída PWM do segundo *gateway* de transdutores e o circuito de potência já apresentado na Figura 100.

Foi necessário alterar o modo de operação do inversor de frequência para acionamento por tensão. Neste modo o inversor é acionado em um *range* de 2 a 10 V.

O valor máximo do *range* do inversor, que representa 100% de potência entregue ao motor elétrico trifásico, foi ajustado pela tensão de alimentação do circuito de potência, em corrente contínua, acoplado à saída PWM do *gateway* de transdutores.

O valor mínimo de 2 V para acionar a bomba centrífuga representa 20% da potência entregue ao motor elétrico trifásico. Esta é uma característica de operação de motores elétricos já implementada pelo inversor de frequência. No entanto, o circuito de potência utilizado pela saída PWM não implementa esta característica, podendo provocar o desligamento do inversor por detecção de falha, caso não garanta o mínimo de 2 V.

A garantia de entrega mínima de 2 V ao inversor de frequência, pelo circuito de potência, também foi deixada para ser tratada pela aplicação cliente.

6.7.2 Sintonização do controlador PID

Na tentativa de controlar a aplicação de controle de nível, com a arquitetura proposta, foi composta a aplicação cliente apresentada na Figura 111.

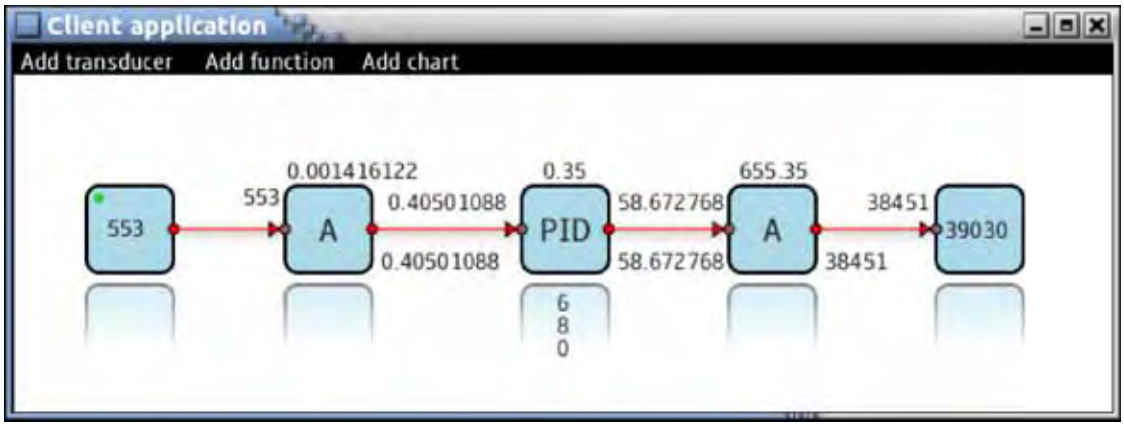
O bloco à esquerda está mapeado em um canal da *tag ADC* do *gateway* de transdutores. Este canal está ligado ao transmissor de pressão.

Logo após o bloco de entrada do transmissor de pressão, um bloco de ganho faz a correção de escala e *offset* dos dados do conversor ADC para o *range* 0 a 1 mca, mesmo *range* adotado pela aplicação supervisória Elipse E3 e pelo indicador digital de pressão presente na planta de controle de processos.

Na sequência, o dado corrigido da pressão é enviado a um bloco de controle PID. Em um primeiro momento tentou-se utilizar os mesmos parâmetros configurados no controlador

PID do CLP. No entanto, percebeu-se que estes parâmetros não sintonizavam adequadamente o PID da aplicação cliente, como será apresentado a seguir.

Figura 111: Interface gráfica da aplicação cliente para controle de nível.



Fonte: Elaborada pelo autor.

O bloco seguinte corrige o *range* de saída do bloco PID (0 a 100) para o *range* de atuação da saída PWM (0 a 65.535), mapeada pelo bloco de saída desta aplicação cliente.

Na Figura 112 é apresentado um gráfico com as variáveis do processo para as constantes proporcional, integral e derivativa em 6, 8 e 0, respectivamente, e *setpoint* de 0,35 mca. Estes foram os mesmos valores utilizados para configurar o CLP, via a aplicação supervisória Elipse E3.

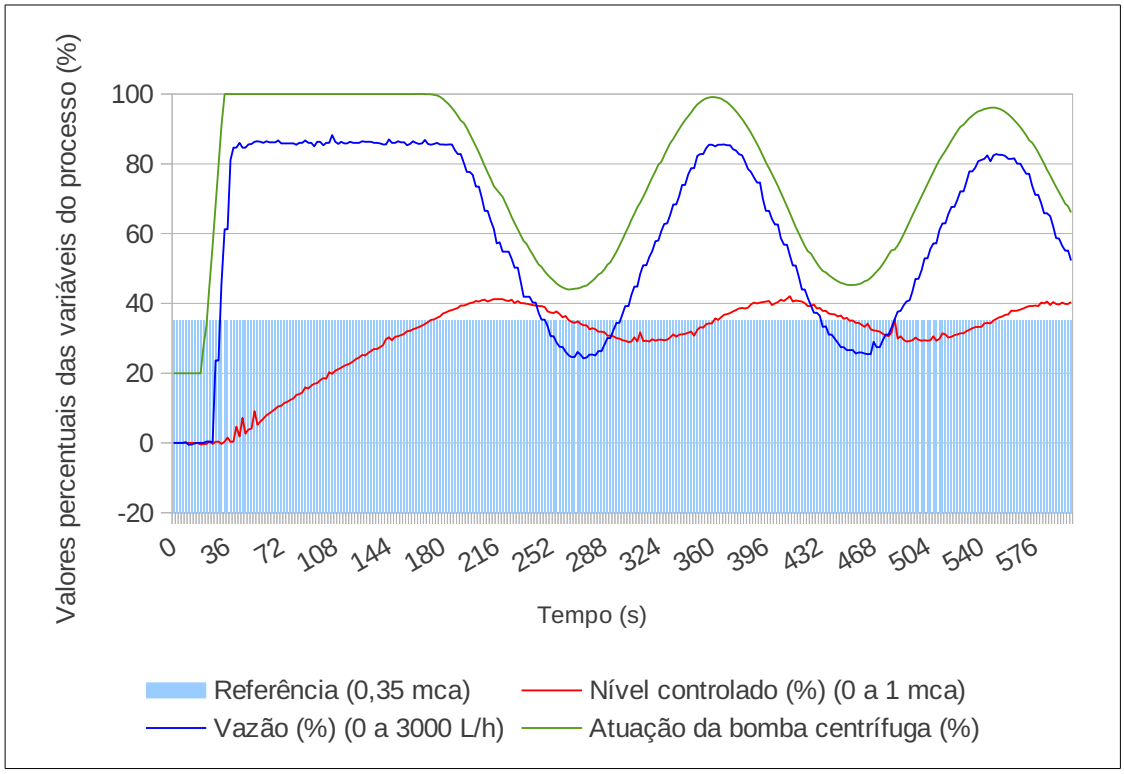
Como pode ser observado no gráfico da Figura 112, o tempo de subida foi praticamente o mesmo que o controlado pelo PID do CLP. No entanto, o amortecimento das variáveis do processo foi muito lento, acarretando em um tempo de acomodação muito elevado.

Com a diferença das respostas dos dois sistemas para os mesmos parâmetros, surgiram algumas dúvidas.

Uma delas foi quanto a correta correção de escala das variáveis do processo. Como não se analisou a aplicação do CLP, não se sabe ao certo se os valores indicados no sistema supervisório Elipse E3 condizem com o sinal de entrada e referência aplicados ao PID do CLP.

Outra dúvida ocorrida foi se os parâmetros integral e derivativo do PID do CLP referiam-se às constantes integral (*Ki*) e derivativa (*Kd*), ou aos períodos integral (*Ti*) e derivativo (*Td*).

Figura 112: Gráfico com variáveis do processo para os parâmetros 6, 8 e 0.



Fonte: Elaborada pelo autor.

Mais uma vez, como a aplicação do CLP não foi analisada, não foi possível saber ao certo o modelo matemático de seu controlador PID. A aplicação supervisória Elipse E3 indicava a unidade “seg.” apenas nos campos dos parâmetros Integral e Derivativo, conforme pode ser visualizado na tela da aplicação apresentada na Figura 113.

Considerando a hipótese dos parâmetros do controlador PID do CLP serem *Ti* e *Td*, de um modelo de controlador PID como o da expressão (3) a seguir:

$$G(s) = Kp \cdot \left(1 + \frac{1}{Ti.s} + Td.s\right) \quad (3)$$

Sabendo que o modelo de controlador PID, implementado na aplicação cliente com os parâmetros *Ki* e *Kd*, pode ser expresso pelo modelo (4) a seguir:

$$G(s) = Kp + \frac{Ki}{s} + Kd.s \quad (4)$$

É simples deduzir que *Ki* e *Kd* podem ser expressos em função de *Ti* e *Td*, respectivamente, como apresentado nas equações (5) e (6) a seguir:

$$Ki = \frac{Kp}{Ti} \quad (5)$$

$$Kd = Kp.Td \quad (6)$$

Assim, considerando os valores 6 e 8 como sendo *Kp* e *Ti*, respectivamente, o valor de *Ki* foi calculado pela equação (5), obtendo-se *Ki* igual a *0,75*.

Figura 113: Interface de configuração do PID da aplicação supervisória Elipse E3.

Fonte: Elaborada pelo autor.

Com este novo parâmetro para *Ki*, a aplicação cliente foi recomposta e a aplicação de controle de nível pela aplicação cliente foi novamente executada.

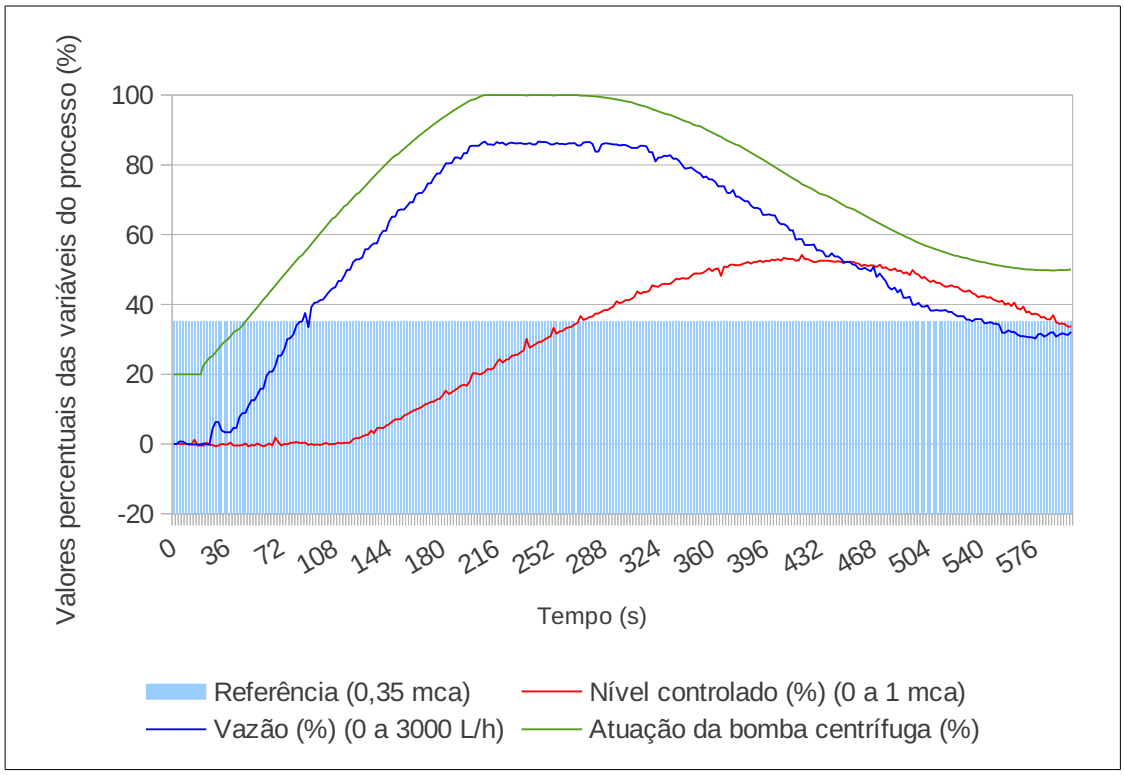
Na Figura 114 é apresentado um gráfico com as variáveis do processo, agora com a constante integral igual a 0,75. Como pode ser observado, o tempo de subida para esta sintonização do controlador PID foi quase o dobro do tempo de subida obtido com o PID do CLP. O *overshoot* também foi muito mais elevado com período de amortecimento muito elevado, acarretando também um tempo de acomodação excessivo.

Esgotada as tentativas de aplicar os mesmos parâmetros empregados no controlador PID do CLP, partiu-se para a tentativa de sintonizar o PID da aplicação cliente por métodos empíricos.

Para tanto se tentou aplicar as regras de Ziegler-Nichols para sintonia de controladores PID (OGATA, 2003).

Na tentativa de obter-se uma curva de resposta da planta com o aspecto de um S, aplicou-se um degrau na planta em malha aberta por meio da atuação da bomba centrífuga.

Figura 114: Gráfico com variáveis do processo para os parâmetros 6, 0,75 e 0.



Fonte: Elaborada pelo autor.

Na Figura 115 é apresentado o gráfico obtido da resposta da planta em malha aberta, para um degrau aplicado à bomba centrífuga.

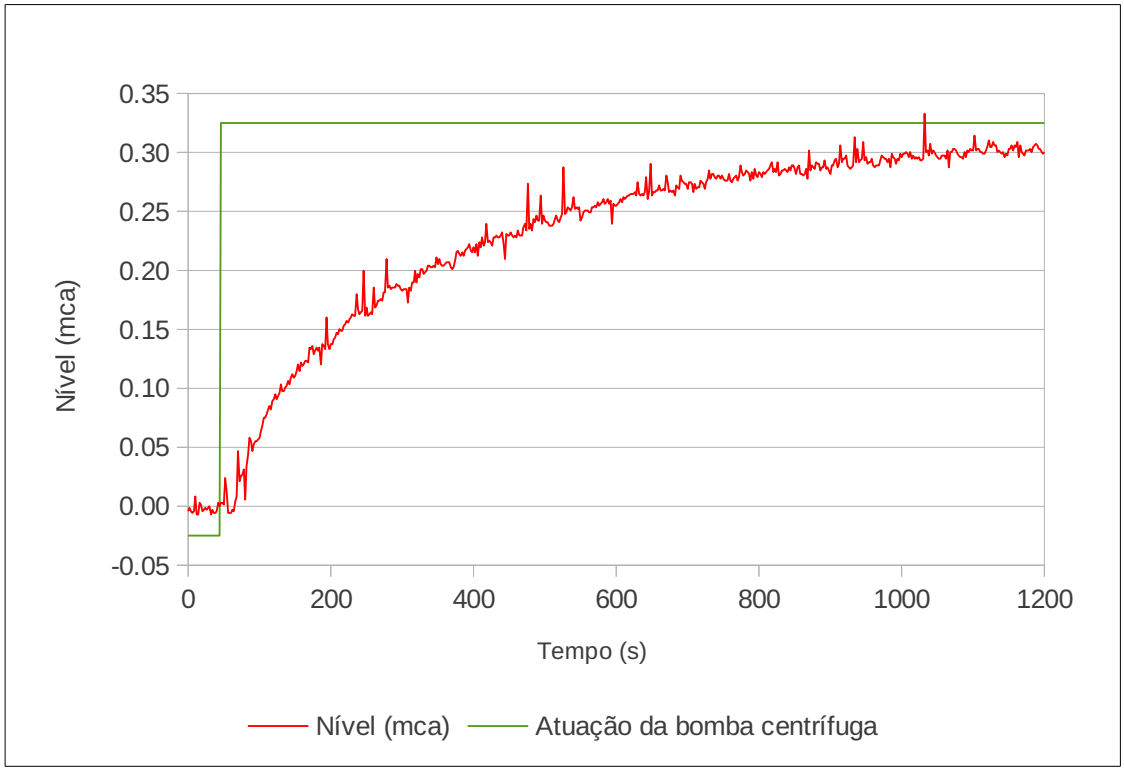
O degrau aplicado à bomba centrífuga foi de 20% a 70% de seu acionamento. A identificação do degrau no gráfico representa apenas o instante da ocorrência e não a amplitude do sinal.

Como pode ser observado no gráfico da Figura 115, a resposta da planta de controle de nível em malha não apresentou aspecto de um S, o que inviabiliza neste caso a aplicação do primeiro método de Ziegler-Nichols.

Mesmo se for considerado que o ponto de inflexão está próximo às oscilações no início da resposta da planta, este ponto de inflexão teria uma reta tangente que passaria muito próxima do instante de aplicação do degrau. Desta forma, o primeiro período para este

método seria próximo da origem, o que também inviabiliza sua aplicação, dado a imprecisão em projetar uma reta tangente nesta região.

Figura 115: Gráfico com a resposta da planta para uma excitação em degrau.



Fonte: Elaborada pelo autor.

Também se tentou aplicar o segundo método de Ziegler-Nichols para sintonia de controladores PID. No entanto, apenas com a ação proporcional do controlador PID da aplicação cliente, não se conseguiu obter uma oscilação sustentada da planta em malha fechada.

Devido ao tempo de resposta razoavelmente lento da planta, não se conseguiu realizar muitas tentativas de estabilização crítica da planta. Foram observadas algumas constantes proporcionais que não conseguiam provocar nenhum aumento do nível. Outros valores para a constante proporcional conseguiam estabilizar o nível com erro estacionário bastante elevado e, ainda, alguns valores da constante proporcional deixavam o sistema instável, operando em saturação e corte do acionamento da bomba centrífuga.

Assim, acredita-se que esta planta de controle de nível possa ser estabilizada criticamente. Entretanto, nos experimentos realizados, não se encontrou a constante proporcional crítica deste sistema.

Por fim, o controlador PID da aplicação de controle de nível foi sintonizado experimentalmente por uma série de tentativas realizadas com a aplicação cliente, conforme os procedimentos descritos a seguir:

1 – Utilizando-se apenas da ação proporcional, procurou-se encontrar a constante proporcional que estabilizasse o processo com o menor erro estacionário. Esta constante foi encontrada, entretanto, o processo apresentou significativo erro estacionário nesta configuração;

2 – Mantida a constante proporcional, procurou-se encontrar uma constante integrativa que provocasse uma resposta transitória no mesmo período percebido na resposta transitória da planta controlada pelo CLP. Encontrada a constante integrativa, verificou-se resposta transitória com amplitude bem superior à resposta da planta controlada pelo CLP;

3 – Mantida a constante integrativa encontrada anteriormente, aumentou-se a constante proporcional, provocando maior amortecimento da resposta transitória, até chegar a amplitudes próximas às percebidas na resposta transitória da planta controlada pelo CLP;

4 – Mantida a nova constante proporcional, encontrada pelo procedimento anterior, fez-se um pequeno ajuste da constante integrativa para afinar o período de resposta transitória.

Com os procedimentos descritos anteriormente, chegou-se aos valores 500 e 40 para as constantes proporcional e integral, respectivamente. Como pôde ser observado, buscou-se obter o mesmo comportamento da planta controlada pelo CLP, o que justifica a não utilização da ação derivativa nesta sintonia do controlador PID.

Registrado estes parâmetros de sintonia do controlador PID, obtidos pelos procedimentos descritos anteriormente, procurou-se aplicar um pouco de ação derivativa para verificar a influência desta ação no comportamento da planta.

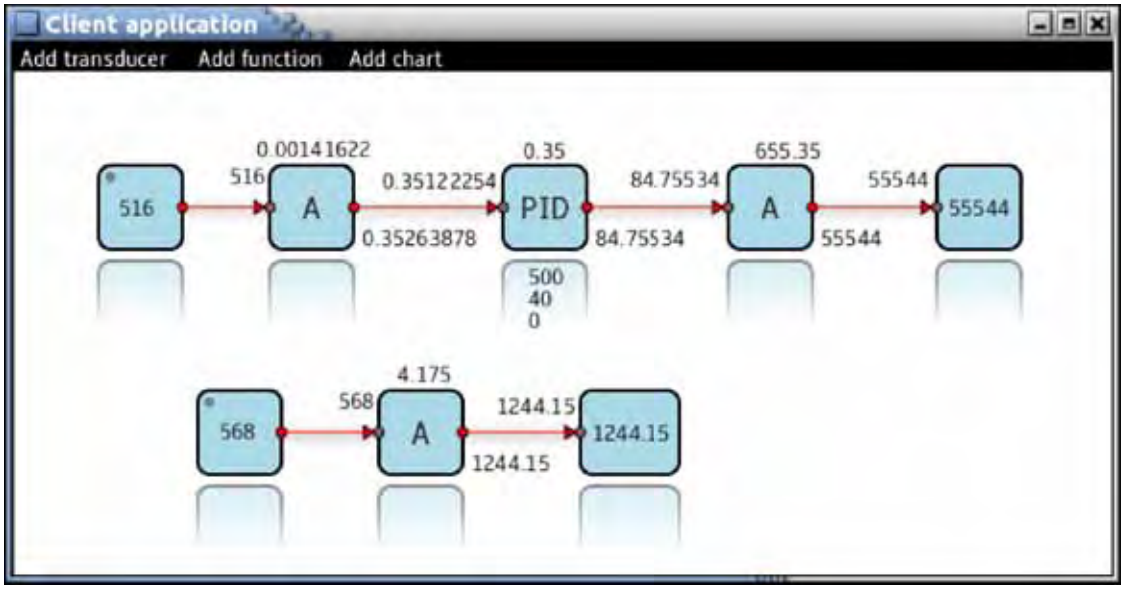
Escolhido um valor para a constante derivativa, aumentou-se um pouco a constante proporcional e foi diminuída a constante derivativa, de forma a não distanciar o comportamento da variável controlada, da resposta já obtida pela sintonia anterior. Desta forma, chegou-se aos valores 700, 20 e 100, para as constantes proporcional, integral e derivativa, respectivamente.

6.7.3 Resultados

Na Figura 116 é apresentada a interface da aplicação cliente composta para o controle de nível e medição de vazão da planta de controle de processos.

A primeira linha de blocos fecha a malha da planta de controle de nível entre o transmissor de pressão e a bomba centrífuga, conforme já descrito na seção anterior. A diferença neste caso foi apenas os parâmentros de sintonia do controlador PID, obtidos experimentalmente, conforme também apresentado na seção anterior.

Figura 116: Interface da aplicação para controle de nível e medição de vazão.



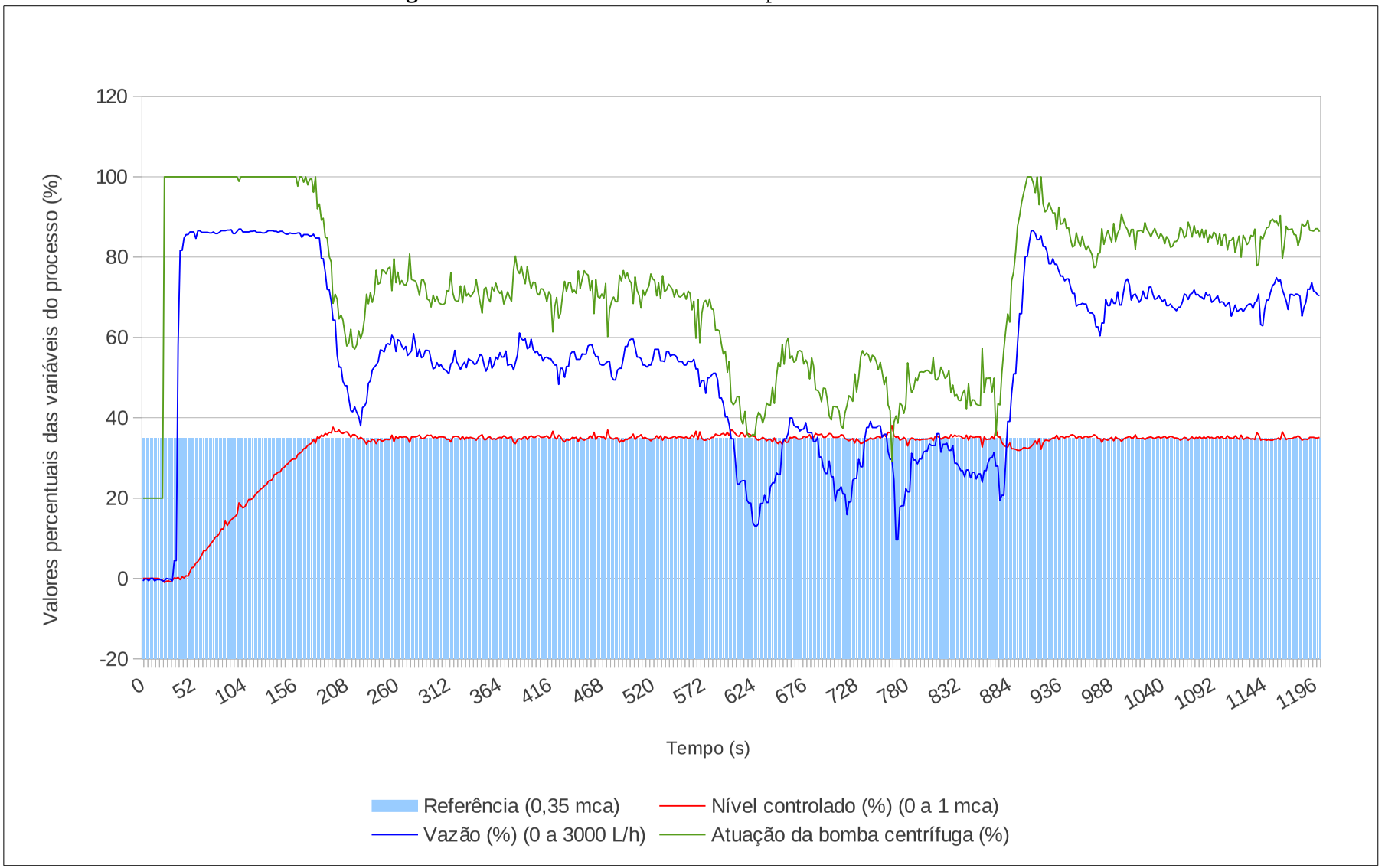
Fonte: Elaborada pelo autor.

A segunda linha de blocos realiza a entrada dos dados do canal AD que está ligado o transmissor de vazão, faz a correção de escala dos dados de entrada recebidos e envia os dados corrigidos para um serviço de registro de dados, respectivamente. Além do serviço de registro de dados, também foram realizados registros locais pela aplicação cliente.

Na Figura 117 é apresentado um gráfico com as curvas de nível, vazão e atuação da bomba centrífuga. Nesta operação foram utilizados os valores 500, 40 e 0 para as constantes proporcional, integral e derivativa, respectivamente.

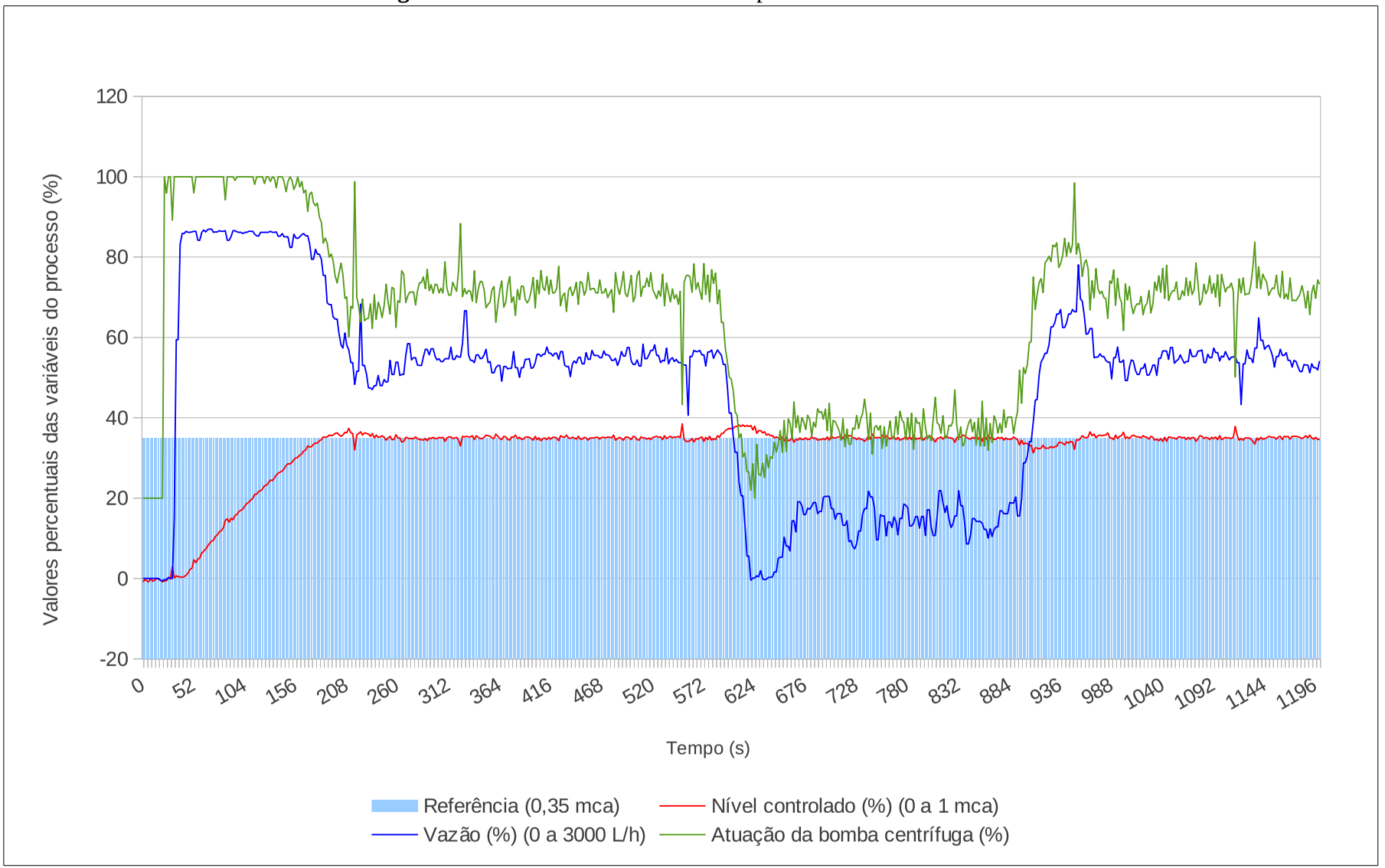
Na Figura 118 é apresentado um gráfico com as curvas de nível, vazão e atuação da bomba centrífuga. Desta vez foram utilizados os valores 700, 20 e 100 para as constantes proporcional, integral, respectivamente, utilizando também a ação derivativa, conforme descrito na seção anterior.

Figura 117: Gráfico com variáveis do processo com controlador PI.



Fonte: Elaborada pelo autor.

Figura 118: Gráfico com variáveis do processo com controlador PID.



Fonte: Elaborada pelo autor.

Os gráficos apresentados nas Figuras 110, 117 e 118 foram produzidos com os dados do processo de controle de nível controlado pelo CLP da planta de controle de processos, controlado pela aplicação cliente com ação PI e controlado pela aplicação cliente com ação PID, respectivamente.

Para melhor comparação, os três experimentos de controle de nível foram realizados sob condições e períodos semelhantes, com uma perturbação provocada pelo fechamento parcial da válvula manual, que permite a vazão do tanque superior para o inferior, conforme descrito na seção 6.6.3.

Comparando-se os três gráficos verifica-se a significativa semelhança no comportamento da variável controlada (nível). No controle realizado pela aplicação cliente com ação PI pode ser observado pequeno ruído de alta frequência da curva da variável controlada.

No entanto, se observadas as curvas de atuação da bomba centrífuga é possível perceber significativa diferença nos três casos. Nas aplicações controladas pela aplicação cliente se percebe significativa oscilação da atuação da bomba se comparado ao comportamento percebido pelo controle realizado pelo CLP. Sendo que no controle com a aplicação cliente com ação PID nota-se oscilação em maior frequência da atuação da bomba centrífuga.

Quanto à curva de vazão, percebe-se nos três casos que seu comportamento é diretamente proporcional à atuação da bomba centrífuga, refletindo as mesmas oscilações já observadas desta.

Uma hipótese da elevada oscilação do atuador da planta (bomba centrífuga), excitado pelo controlador PID, é que as sintonias obtidas tenham adotado valores muito elevados para a ação proporcional. Talvez seja possível procurar por sintonias com valores menores para a constante proporcional que diminua essa oscilação e preserve o comportamento da variável controlada.

Outra hipótese levantada para a oscilação da saída do controlador PID é a diferença de resolução do ADC do CLP, com a resolução do ADC do *gateway* de transdutores. Enquanto as entradas ADs do CLP possuem resolução de 12 bits, as entradas ADs do *gateway* de transdutores são de 10 bits. Esta diferença pode estar acarretando a maior oscilação da saída do controlador PID da aplicação cliente, por apresentar maior erro entre dois valores adjacentes.

6.8 CONCLUSÕES

Neste capítulo foi possível desenvolver elementos da arquitetura proposta, para efetivamente demonstrar exemplos de aplicações sobre a mesma.

Enquanto algumas das implementações evidenciam as diversas possibilidades do emprego da arquitetura para publicação e universalização do acesso a transdutores inteligentes, devido a sua característica aberta e ao emprego de padrões consagrados, outras aplicações, como as de controle de processos, efetivamente demonstram e validam a arquitetura, como uma possibilidade eficaz para supervisão e controle de processos.

As implementações de controle de processos também vislumbram a possibilidade de controle a custos reduzidos, se comparados a soluções industriais clássicas. Entretanto, questões de segurança precisam ser investigadas, principalmente quanto a falhas e indisponibilidade da rede de dados.

Também fica evidente o alto nível de abstração para o desenvolvedor da aplicação cliente, que pode se utilizar de técnicas de programação sobre SOA, sem, necessariamente, se conhecer detalhes físicos dos transdutores empregados.

Por sua vez, o desenvolvedor dos transdutores inteligentes e circuitos de condicionamento e acionamento, podem deixar todo o tratamento de dados e lógicas de controle para serem realizados em software, como serviços ou na aplicação cliente.

6.9 PROPOSTAS DE TRABALHOS FUTUROS

Ao término das implementações realizadas neste capítulo, algumas dúvidas e motivações sugerem frentes de pesquisas e desenvolvimentos futuros, como as que seguem:

- Migração da aplicação cliente para JavaFX 2, 3, ou outra tecnologia;
- Armazenamento do *script* do *gateway* de transdutores em memória Flash interna do microcontrolador;
- Emprego de outras plataformas de hardware como *gateway* de transdutores;
- Desenvolvimento de plataforma de hardware própria para o *gateway* de transdutores;
- Implementação de controle PID digital com *setpoint* variante no tempo;
- Implementação de funções de divisão de sinais;
- Desenvolvimento de mais recursos e funções na aplicação cliente;

- Desenvolvimento de outros serviços para a arquitetura proposta;
- Investigação da diferença de comportamento do controlador PID do CLP, da planta de controle de processos, com o controlador PID da aplicação cliente;
- Investigação das possíveis causas da oscilação acentuada da atuação da bomba centrífuga, do processo de controle de nível com a aplicação cliente, em comparação ao controle com o CLP da planta de controle de processos;
- Verificação da possibilidade de integração da arquitetura proposta com ferramentas livre, como a Xcos[29] do SciLab[30];
- Aplicação da arquitetura proposta em outros estudos de caso;
- Investigação do desempenho computacional, das taxas de comunicação e do consumo de energia elétrica;
- Aplicação da arquitetura proposta em estudos de caso com desempenho crítico.

29 Ferramenta de simulação de sistemas de controle.
30 Software livre de computação numérica.

7 CONCLUSÕES GERAIS

No primeiro capítulo concebeu-se a proposta deste trabalho, desenvolvida nas pesquisas dos capítulos seguintes 2, 3 e 4, que muito contribuíram para o esclarecimento e amadurecimento de conceitos, ideias e concepções.

No capítulo 5 foi documentada a proposta de arquitetura para publicação e universalização do acesso a transdutores inteligentes, sendo resultado não apenas dos capítulos anteriores, mas também do capítulo 6. As implementações desenvolvidas no capítulo 6 contribuíram significativamente para evidenciar as possibilidades de emprego da arquitetura proposta e também para o seu refinamento.

Espera-se que a continuidade do desenvolvimento de elementos da arquitetura e aplicações sobre ela permita o contínuo aprimoramento da proposta. No entanto, diante da característica de sistema aberto desta arquitetura, desde a sua concepção, era de se esperar infinitas possibilidades de desenvolvimento, sobre as mais diversas áreas em que se deseja monitorar ou controlar transdutores.

Diante da diversidade de possibilidades, os exemplos de controle de temperatura e nível apresentados validam a arquitetura proposta e sua eficácia no monitoramento e controle de processos.

Desta forma, a arquitetura proposta publica e universaliza o acesso a transdutores inteligentes, simplificando e democratizando o acesso aos transdutores, a profissionais de diversas áreas, além de permitir a tradução e integração entre os padrões implementados e quaisquer outros que venham a ser implantados em *gateways* de tradução.

A publicação dos transdutores inteligentes em diversos padrões, dentro de uma arquitetura aberta, por padrões consagrados, é uma forma de libertar os transdutores, os publicando ao mundo. Entende-se que esta libertação é a principal contribuição deste trabalho. Espera-se que seguida tal proposta, os transdutores inteligentes sejam dispositivos mais presentes e populares no cotidiano de todos.

Também são contribuições pontuais deste trabalho, entre outras: o mapeamento RESTful do *gateway* de transdutores; a racionalização de recursos de hardware dos transdutores inteligentes, com transferência de necessidades computacionais para serviços em nuvem; a proposta de *gateways* de tradução de padrões que, embora a proposta não seja inédita, talvez seja na forma de serviços em nuvem; o modelo de dados sugerido para a aplicação cliente; a aplicação cliente para composição de aplicações, com interface simples e

interativa; e a aproximação das áreas do conhecimento de instrumentação e tecnologia da informação.

Por fim, concluí-se que o encerramento deste trabalho marca o início de uma infinidade de possibilidades de trabalhos futuros, elencados ou não nas seções específicas para este fim, de cada um dos capítulos apresentados anteriormente.

8 TRABALHOS PUBLICADOS E ACEITOS

RIBEIRO, A. A. L; SILVA, A. C. R. Architecture for publication and universal access to smart transducers. In: IEEE INTERNATIONAL INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE - I2MTC, 2012, Graz, Áustria. **Proceedings...** New York: IEEE, 2012. p. 2361-2364. (Publicado).

RIBEIRO, A. A. L; SILVA, A. C. R. Application of process control over SOA for smart transducers. In: IEEE SENSORS APPLICATIONS SYMPOSIUM – SAS, Feb. 2013, Galveston, Texas, USA. **Proceedings...** New York: IEEE, 2013. (Aceito).

RIBEIRO, A. A. L; SILVA, A. C. R. Controle distribuído com transdutores orientados a serviços. In: INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER EDUCATION – ICECE, 8., Mar. 2013, Luanda, Angola. **Proceedings...** [S.l.]: COPEC, 2013. (Aceito).

REFERÊNCIAS

2EI ENGENHARIA. **KPME-10 – Guia do usuário.** Versão 1.10. São Paulo: 2EI - Eletrônica Embarcada para Internet, 2007. 64 p.

2EI ENGENHARIA. **Loja virtual.** São Paulo: 2EI - Eletrônica Embarcada para Internet, c2005. Disponível em: <http://www.2ei.com.br/loja/>. Acesso em: 15 jul. 2010.

ADOBE SYSTEMS. **Adobe AIR 3**. San Jose, 2011a. Disponível em: <http://www.adobe.com/products/air.html>. Acesso em: 29 nov. 2011.

ADOBE SYSTEMS. **Flex**. San Jose, 2011b. Disponível em: <http://www.adobe.com/products/flex.html>. Acesso em: 29 nov. 2011.

AKYILDIZ, I. F.; SU, W.; SANKARASUBRAMANIAM, Y.; CAYIRCI, E. Wireless sensor networks: a survey. **Computer Networks**, Amsterdam, n. 38, p. 393-422, 2002.

ALTERA CORPORATION. **Cyclone II device handbook**. San Jose, 2008. vol. 1, 470 p.

ALTERA CORPORATION. **DE2 Development and education board:** user manual. Version 1.4. San Jose, 2006. 72 p.

ALTERA CORPORATION. **DE2 Development and education board**. San Jose, 2011. Disponível em: <http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html>. Acesso em: 8 nov. 2011.

APACHE SOFTWARE FOUNDATION. **Web Services – Axis**. [S.l.], 2005. Disponível em: <http://ws.apache.org/axis/>. Acesso em: 14 nov. 2011.

APACHE SOFTWARE FOUNDATION. **Apache Axis2/Java**. [S.l.], 2011. Disponível em: <http://axis.apache.org/axis2/java/core/>. Acesso em: 14 nov. 2011.

APACHE SOFTWARE FOUNDATION. **Apache Axis2/C**. [S.l.], 2009. Disponível em: <http://axis.apache.org/axis2/c/core/>. Acesso em: 14 nov. 2009.

ATMEL CORPORATION. **AVR®32 32-bit Microcontroller – AT32AP7000**. AVR32 AP7000 datasheet. San Jose, 2009. 973 p.

AVRFREAKS WIKI. **Documentation: NGW**. [S.l.], 2009. Disponível em: <http://www.avrfreaks.net/wiki/index.php/Documentation:NGW>. Acesso em: 8 nov. 2011.

BUTEK, R.; GALLARDO, N. **Web services hints and tips: JAX-RPC versus JAX-WS, Part 2**. Armonk: IBM Corporation, 2006. Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-tip-jaxwsrpc2.html>. Acesso em: 14 nov. 2011.

BUYYA, R. et al. Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. **Future Generation Computer Systems**, Amsterdam, v. 25, p. 599–616, jun. 2009.

CHUNYE, G. et al. The characteristics of cloud computing. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING WORKSHOPS – ICPPW, 39., 2010, San Diego, California, USA. **Proceedings**..., Piscataway: IEEE, 2010. p. 275-279.

CLARKE, J.; CONNORS, J.; BRUNO, E. **JavaFX: desenvolvendo aplicações de internet ricas**. Rio de Janeiro: Alta Books, 2010. 325 p.

COETZEE, L.; EKSTEEN, J. The internet of things – promise for the future? an introduction. In: IST-AFRICA, May 2011, Gaborone, Botswana, Africa do Sul. **Proceedings**... New York: IEEE, 2011. 9 p.

COOK, W. R.; BARFIELD, J. Web services versus distributed objects: a case study of performance and interface design. In: IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES – ICWS, Sept. 2006, Chicago, USA. **Proceedings**... Piscataway: IEEE, 2006. p. 419-426.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to algorithms**. 2. ed. Cambridge: MIT Press, 2001.

DELIN, K. A.; JACKSON, S. P. The sensor web: a new instrument concept. In: SYMPOSIUM ON INTEGRATED OPTICS, Jan. 2001, San Jose. **Proceedings**... Bellingham: SPIE, 2001. 9 p.

DELL. **Site oficial Dell - Notebooks, computadores e netbooks**. 2010. Disponível em: <http://www.dell.com.br>. Acesso em: 15 jul. 2010.

DENNING, P. **Annotated list of web services specs**. 2004. Disponível em: <http://lists .w3.org/Archives/Public/www-ws-arch/2004Feb/0022.html>. Acesso em: 14 nov. 2011.

DIGIKEY CORPORATION. **Electronic components distributor - DigiKey Corp.** 2012. Disponível em: <http://www.digikey.com>. Acesso em: 20 dez. 2010.

EVANS, T. D. **NetBIOS, NetBEUI, NBF, NBT, NBIPX, SMB, CIFS Networking.** [S.l.]: DocBook, 2003. 73 p.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. 2000. 162 f. Dissertação (Doutorado em Informação e Ciências da Computação) – Instituto de Ciências da Computação, Universidade da Califónia, Irvine, 2000.

FSF – FREE SOFTWARE FOUNDATION. **GNU General Public License**, ver. 3. June 2007.

FSF – FREE SOFTWARE FOUNDATION. **GNU Classpath**. 2009. Disponível em: <http://www.gnu.org/software/classpath/>. Acesso em: 14 nov. 2011.

GIRERD, C. et al. Ethernet network-based DAQ and smart sensors for the OPERA long-baseline neutrino experiment. In: IEEE NUCLEAR SCIENCE SYMPOSIUM CONFERENCE RECORD (NSS 2000), Lyon, France, 2000. **Proceedings**..., Piscataway: IEEE, 2000. v. 2, p. 111-115.

GUERRA, W. A. **Implementação de controle proporcional, integral e derivativo digital em controladores lógico programáveis.** 2009. 40 f. Monografia (Especialização em Engenharia de Instrumentação) – Centro de Tecnologia e Geociências, Universidade Federal de Pernambuco, Recife, 2009.

IEEE INSTRUMENTATION AND MEASUREMENT SOCIETY. **IEEE Standard for a smart transducer interface for sensors and actuators**: common functions, communication protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Std. 1451.0-2007, TC-9. Piscataway: IEEE, set. 2007. 335 p.

IEEE INSTRUMENTATION AND MEASUREMENT SOCIETY. **IEEE Standard for a smart transducer interface for sensors and actuators – Network Capable Application (NCAP) information model**, IEEE Std. 1451.1-1999, TC-9. Piscataway: IEEE, abr. 2000. 349 p.

JOSHI, N. N.; DAKHOLE, P. K.; ZODE, P. P. Embedded Web Server on Nios II Embedded FPGA Platform. In: INTERNATIONAL CONFERENCE ON EMERGING TRENDS IN ENGINEERING & TECHNOLOGY – ICETET, 2., 2009, Nagpur, India. **Proceedings**... Piscataway: IEEE, dez. 2009. p. 372-377.

JOSUTTIS, N. M. **SOA na prática**. Rio de Janeiro: Alta Books, 2008. 266 p.

JSON – JAVASCRIPT OBJECT NOTATION. **Introducing JSON**. Disponível em: <http://www.json.org/>. Acesso em: 14 nov. 2011.

KATO, Y. *Splish:* A visual programming environment for arduino to accelerate physical computing experiences. In: INTERNATIONAL CONFERENCE ON CREATING CONNECTING AND COLLABORATING THROUGH COMPUTING - C5, 8., 2010, La Jolla, CA, USA. **Proceedings**... Piscataway: IEEE, 2010. p. 3-10.

KUCZENSKI B.; LEDUC, P. R.; MESSNER, W. C. A platform for building PIC applications for control and instrumentation. In: AMERICAN CONTROL CONFERENCE – ACC, 2005, Portland, OR, USA. **Proceedings**... Piscataway: IEEE, 2005. v. 7, p. 5162-5168.

LAMMARSCH, T. et al. A comparison of programming platforms for interactive visualization in web browser based applications. In: INTERNATIONAL CONFERENCE INFORMATION VISUALISATION, 12., 2008, Londres, UK. **Proceedings**... Piscataway: IEEE, 2008. p. 194-199.

LI, H. et al. Jax-WS enabled digital ecosystems approach for coal mining applications. In: ANNUAL CONFERENCE OF THE IEEE INDUSTRIAL ELECTRONICS SOCIETY – IECON, 35., 2009, Porto, Portugal. **Proceedings**... Piscataway: IEEE, 2009. p. 3937-3941.

LIU, Y.; LIU, X-F.; MAO, J-H. Research on the integration of Silverlight and WebGIS based on REST. In: INTERNATIONAL CONFERENCE ON MODEL TRANSFORMATION – ICMT, 3., 2010, Ningbo, China. **Proceedings**... Piscataway: IEEE, Oct. 2010. 4 p.

LOUGHER, R. **JamVM**. 2010. Disponível em: <http://jamvm.sourceforge.net/>. Acesso em: 14 nov. 2011.

MERHI, Z. M.; ELGAMEL, M. A.; BAYOUMI, M. A. A lightweight collaborative fault tolerant target localization system for wireless sensor networks. **IEEE Transactions on Mobile Computing**, Piscataway, v. 8, n. 12, p. 1690-1704, Dec. 2009.

MICROCHIP TECHNOLOGY. **PIC18F6520/8520/6620/8620/6720/8720 Datasheet**. Chandler, 2004. 380 p.

MICROCHIP TECHNOLOGY. **PIC24FJ256GB110 Family Data Sheet**. Chandler, 2009. 352 p.

MICROSOFT CORPORATION. **COM - Component Object Model Technologies**. Redmond, 2011a. Disponível em: <http://www.microsoft.com/com/default.mspx>. Acesso em: 14 nov. 2011.

MICROSOFT CORPORATION. **Microsoft Silverlight**. Redmond, 2011b. Disponível em: <http://www.microsoft.com/silverlight/>. Acesso em: 29 nov. 2011.

MULLER, G. **System Architecting.** Version 1.8. Eindhoven : Buskerud University College, fev. 2012. 235 p.

NATIS, Y. V. **Service-Oriented Architecture Scenario**. Stamford: Gartner, abr. 2003. 6 p.

OASIS OPEN. Committee Specification 1. **Reference Model for Service Oriented Architecture 1.0**. Aug. 2006. 31 p.

OASIS OPEN. UDDI Committee Specification. **UDDI Version 2.04 API Specification**. July 2002. 93 p.

OBERMEYER, P.; HAWKINS, J. **Microsoft .NET Remoting:** a technical overview. Redmond: Microsoft Corporation, July 2001. Disponível em: <http://msdn.microsoft.com/en-us/library/ms973857.aspx>. Acesso em: 14 nov. 2011.

OGATA, K. **Engenharia de controle moderno**. 4. ed. São Paulo: Prentice-Hall, 2003. 788 p.

OGC – OPEN GEOSPATIAL CONSORTIUM. **OGC Reference Model.** Version 1.2. Wayland, Dec. 2011. 44 p. (Open Geospatial Consortium, OGC 08-062r7).

OGC – OPEN GEOSPATIAL CONSORTIUM. **OGC Sensor Web Enablement:** overview and high level architecture. Version 3. Wayland, Dec. 2007. 14 p.

OLIVEIRA, A. S.; ANDRADE, F. S. **Sistemas Embarcados:** hardware e firmware na prática. São Paulo: Érica, 2006.

OMG – OBJECT MANAGEMENT GROUP. **History of CORBA**. Needham, 2009. Disponível em: <http://www.omg.org/gettingstarted/history_of_corba.htm>. Acesso em: 17 abr. 2010.

OPC FOUNDATION. **OPC Unified Architecture Specification:** Part 1: overview and concepts. Release 1.01. Auburn Township, Ohio, fev. 2009. 28 p.

ORACLE CORPORATION. **Gestão de processos de negócios, arquitetura orientada a serviços e Web 2.0:** transformação da empresa ou rota de colisão? Redwood City, ago. 2008. 26 p. (White Paper Oracle).

ORACLE CORPORATION. **Java remote method invocation:** distributed computing for Java. Redwood City, 2011a. Disponível em: <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html>. Acesso em: 14 nov. 2011.

ORACLE CORPORATION. **JavaFX 2.0**. Redwood City, 2011b. Disponível em: <http://www.oracle.com/us/technologies/javafx/overview/index.html>. Acesso em: 29 nov. 2011.

ORACLE CORPORATION. **JavaFX Roadmap**. Redwood City, 2012. Disponível em: <http://www.oracle.com/technetwork/java/javafx/overview/roadmap-1446331.html>. Acesso em: 7 mai. 2012.

PROSYS ENGENHARIA. **Apostila de treinamento:** planta didática para controle de processos. Barueri, abr. 2011. 68 p. (POC03-F02-A).

RIBEIRO, A. A. L; SILVA, A. C. R. Architecture for publication and universal access to smart transducers. In: IEEE INTERNATIONAL INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE - I2MTC, 2012, Graz, Áustria. **Proceedings...** Piscataway: IEEE, 2012. p. 2361-2364.

ROSSI, S. R. et al. Open and standardized resources for smart transducer networking. **IEEE Transactions on Instrumentation and Measurement**, Piscataway, v. 58, n. 10, p. 3754-3761, Oct. 2009.

SAVOCHKIN A. A. et al. Universal system of monitoring and control on the basis of the NGN Concept. In: INTERNATIONAL CONFERENCE ON ULTRAWIDEBAND AND ULTRASHORT IMPULSE SIGNALS – UWBUSIS, 4., 2008, Sevastopol, Ukraine. **Proceedings**... Piscataway: IEEE, 2008. p. 113-115.

SCHERER, R. W.; KLEINSCHMIDT, J. H. A middleware architecture for wireless sensor networks using secure web services. **IEEE Latin America Transactions**, Piscataway, v. 9, n. 5, p. 815-820, Sept. 2011.

SCHULTE, W. R.; NATIS, Y. V. **Service oriented architectures:** part 1. Stamford: Gartner, Apr. 1996. 2 p.

SHELBY, Z. et al. **Constrained Application Protocol (CoAP) - draft-ietf-core-coap-13**. Orlando: The Internet Engineering Task Force – IETF, Dec. 2012. Disponível em: <http://tools.ietf.org/html/draft-ietf-core-coap-13>. Acesso em: 20 dez. 2012.

SHETH, A.; HENSON, C.; SAHOO, S. S. Semantic Sensor Web. **IEEE Internet Computing**, Piscataway, v. 12, n. 4, p. 78-83, July 2008.

SONG, E. **Open1451:** an IEEE 1451-based smart transducer standard implementation project repository. Gaithersburg: National Institute of Standards and Technology – NIST, 2004. Disponível em: <http://open1451.sourceforge.net/>. Acesso em: 14 nov. 2011.

SONG, E.; LEE, K. STWS: a unified web service for IEEE 1451 smart transducers. **IEEE Transactions on Instrumentation and Measurement**, Piscataway, v. 57, n. 8, p. 1749-1756, ago. 2008a.

SONG, E. Y.; LEE, K. Integration of IEEE 1451 smart transducers and OGC-SWE using STWS. In: IEEE SENSORS APPLICATIONS SYMPOSIUM – SAS, 2009, New Orleans, LA, USA. **Procedings**... Piscataway: IEEE, Feb. 2009. p. 298-303.

SONG, E. Y.; LEE, K. Understanding IEEE 1451 – Networked Smart Transducer Interface Standard. **IEEE Instrumentation and Measurement Magazine**, Piscataway, v. 11, n. 2., p. 11-17, Apr. 2008b.

SOUZA, V. A. S. M. **Uma arquitetura orientada a serviços para desenvolvimento, gerenciamento e instalação de serviços de rede**. 2006. 93 f. Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2006.

STALLINGS, W. **Computer organization and architecture**. 5. ed. New Jersey: Prentice-Hall. 2000. 748 p.

SUN MICROSYSTEMS. **JavaTM API for XML-based RPC JAX-RPC 1.1**, **JSR-101**. Version 1.1. Santa Clara, California, Oct. 2003. 167 p.

SUN MICROSYSTEMS. **The Java API for XML-Based Web Services (JAX-WS) 2.0**, **JSR-224**. Version 2.0. Santa Clara, California, Apr. 2006. 149 p.

SUN MICROSYSTEMS. **JAX-RS:** JavaTM API for RESTful Web Services, JSR-311. Version 1.0. Santa Clara, California, Sept. 2008. 49 p.

SURE ELECTRONICS. **PIC24 Web Server demo board user's guide.** Thane, 2010. 17 p.

SURE ELECTRONICS. **Sure Electronics' webstore**.Thane, 2012. Disponível em: <http://www.sureelectronics.net/>. Acesso em: 20 dez. 2012.

TANENBAUM, A. S. **Redes de computadores**. 4. ed. Rio de Janeiro: Campus. 2003. 945 p.

TORRES, G. Redes de computadores: curso completo. Rio de Janeiro: Axcel Books, 2001. 664 p.

TYAGI, S. **RESTful Web Services**. Santa Clara, California: Sun Microsystems, Aug. 2006. Disponível em: <http://www.oracle.com/technetwork/articles/javase/index-137171.html>. Acesso em: 14 nov. 2011.

VIEGAS, V.; PEREIRA, J. M. D.; GIRÃO, P. M. B. S. .NET Framework and Web Services: a profit combination to implement and enhance the IEEE 1451.1 Standard. **IEEE Transactions on Instrumentation and Measurement**, Piscataway, v. 56, n. 6, p. 2739-2747, Dec. 2007.

W3C – WORLD WIDE WEB CONSORTIUM. **Extensible Markup Language (XML) 1.0.** 5. ed. Cambridge: W3C Recommendation, Nov. 2008.

W3C – WORLD WIDE WEB CONSORTIUM. **SOAP Version 1.2 Part 1:** Messaging Framework. 2. ed. Cambridge: W3C Recommendation, Apr. 2007.

W3C – WORLD WIDE WEB CONSORTIUM. **Web Services Architecture**. Cambridge: W3C Note, Feb. 2004.

W3C – WORLD WIDE WEB CONSORTIUM. **Web Services Description Language (WSDL) 1.1**. Cambridge: W3C Note, Mar. 2001.

WOBSCHALL, D. Networked sensor monitoring using the universal IEEE 1451 Standard. **IEEE Instrumentation and Measurement Magazine**, Piscataway, v. 11, n. 2, p. 18-22, Apr. 2008.

ZEQIANG, C. et al. A flexible data and sensor planning service for virtual sensors based on web service. **IEEE Sensors Journal**, Piscataway, v. 11, n. 6, p. 1429-1439, jun. 2011.

ZUG, S.; DIETRICH, A.; KAISER, J. An architecture for a dependable distributed sensor system. **IEEE Transactions on Instrumentation and Measurement**, Piscataway, v. 60, n. 2, p. 408-419, Feb. 2011.

APÊNDICE A

MAPEAMENTO XML DA PLACA DB-DP11115

```
 1:    <?xml version="1.0" encoding="UTF-8"?>
 2:    <transducerGateway>
 3:        <id>1</id>
 4:        <description>General purpose node.</description>
 5:        <ADC>
 6:            <resolution>10 bits</resolution>
 7:            <ch0>605</ch0>
 8:            <ch1>1023</ch1>
 9:            <ch2>91</ch2>
10:            <ch3>0</ch3>
11:        </ADC>
12:        <digitalIOs>
13:            <d0>0</d0>
14:            <d1>1</d1>
15:            <d2>1</d2>
16:            <d3>1</d3>
17:            <d4>1</d4>
18:            <d5>1</d5>
19:            <d6>1</d6>
20:            <d7>1</d7>
21:        </digitalIOs>
22:        <switches>
23:            <sw0>close</sw0>
24:            <sw1>open</sw1>
25:            <sw2>open</sw2>
26:        </switches>
27:        <thermometer>
28:            <temperature>28.3</temperature>
29:            <unit>oC</unit>
30:        </thermometer>
31:        <pwm>
32:            <resolution>16 bits</resolution>
33:            <cycle>65535</cycle>
34:            <dutyCycle>0</dutyCycle>
35:        </pwm>
36:        <lcd>
37:            <text>Transducer GW 192.168.2.20</text>
38:            <maxSize>32</maxSize>
39:        </lcd>
40:        <uart>
41:            <message></message>
42:            <bufferSize>16</bufferSize>
43:        </uart>
44:    </transducerGateway>
```

APÊNDICE B

ARQUIVO WADL DO *GATEWAY* RESTFUL

```
 1: <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 2: <application xmlns="http://research.sun.com/wadl/2006/10">
 3:   <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedB
      y="Jersey: 1.8 06/24/2011 12:17 PM"/>
 4:   <resources base="http://localhost:8080/gatewayRESTful/webreso
      urces/">
 5:     <resource path="digitalios">
 6:       <method id="postXml" name="POST">
 7:         <request>
 8:           <representation mediaType="application/xml"/>
 9:         </request>
10:       </method>
11:       <method id="getXml" name="GET">
12:         <response>
13:           <representation mediaType="application/xml"/>
14:         </response>
15:       </method>
16:     </resource>
17:     <resource path="thermometer">
18:       <method id="getXml" name="GET">
19:         <response>
20:           <representation mediaType="application/xml"/>
21:         </response>
22:       </method>
23:     </resource>
24:     <resource path="lcd">
25:       <method id="postXml" name="POST">
26:         <request>
27:           <representation mediaType="application/xml"/>
28:         </request>
29:       </method>
30:       <method id="getXml" name="GET">
31:         <response>
32:           <representation mediaType="application/xml"/>
33:         </response>
34:       </method>
35:     </resource>
36:     <resource path="adc">
37:       <method id="postXml" name="POST">
38:         <request>
39:           <representation mediaType="application/xml"/>
40:         </request>
41:       </method>
42:       <method id="getXml" name="GET">
```

```
43:           <response>
44:             <representation mediaType="application/xml"/>
45:           </response>
46:         </method>
47:       </resource>
48:       <resource path="uart">
48:         <method id="postXml" name="POST">
50:           <request>
51:             <representation mediaType="application/xml"/>
52:           </request>
53:         </method>
54:         <method id="getXml" name="GET">
55:           <response>
56:             <representation mediaType="application/xml"/>
57:           </response>
58:         </method>
59:       </resource>
60:       <resource path="pwm">
61:         <method id="postXml" name="POST">
62:           <request>
63:             <representation mediaType="application/xml"/>
64:           </request>
65:         </method>
66:         <method id="getXml" name="GET">
67:           <response>
68:             <representation mediaType="application/xml"/>
69:           </response>
70:         </method>
71:       </resource>
72:       <resource path="switches">
73:         <method id="postXml" name="POST">
74:           <request>
75:             <representation mediaType="application/xml"/>
76:           </request>
77:         </method>
78:         <method id="getXml" name="GET">
79:           <response>
80:             <representation mediaType="application/xml"/>
81:           </response>
82:         </method>
83:       </resource>
84:     </resources>
85:   </application>
```

APÊNDICE C

ARQUIVO WSDL DO *GATEWAY WEB SERVICE PWM*

```
 1:  <?xml version='1.0' encoding='UTF-8'?>
 2:  <definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01
     /oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http:
     //www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoa
     p.org/ws/2004/09/policy"_xmlns:wsam="http://www.w3.org/2007/05
     /addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/w
     sdl/soap/" xmlns:tns="http://control/" xmlns:xsd="http://www.w
     3.org/2001/XMLSchema"_xmlns="http://schemas.xmlsoap.org/wsdl/"
     targetNamespace="http://control/" name="pwm">
 3:    <types>
 4:      <xsd:schema>
 5:        <xsd:import namespace="http://control/"_schemaLocation="
           http://localhost:8080/gatewayWS/pwm?xsd=1"/>
 6:      </xsd:schema>
 7:    </types>
 8:    <message name="getResolution">
 9:      <part name="parameters" element="tns:getResolution"/>
10:    </message>
11:    <message name="getResolutionResponse">
12:      <part name="parameters" element="tns:getResolutionResponse
         "/>
13:    </message>
14:    <message name="getCycle">
15:      <part name="parameters" element="tns:getCycle"/>
16:    </message>
17:    <message name="getCycleResponse">
18:      <part name="parameters" element="tns:getCycleResponse"/>
19:    </message>
20:    <message name="setCycle">
21:      <part name="parameters" element="tns:setCycle"/>
22:    </message>
23:    <message name="setCycleResponse">
24:      <part name="parameters" element="tns:setCycleResponse"/>
25:    </message>
26:    <message name="getDutyCycle">
27:      <part name="parameters" element="tns:getDutyCycle"/>
28:    </message>
29:    <message name="getDutyCycleResponse">
30:      <part name="parameters" element="tns:getDutyCycleResponse"
         />
31:    </message>
32:    <message name="setDutyCycle">
33:      <part name="parameters" element="tns:setDutyCycle"/>
34:    </message>
```

```
35:     <message name="setDutyCycleResponse">
36:       <part name="parameters" element="tns:setDutyCycleResponse"
          />
37:     </message>
38:     <portType name="Pwm">
39:       <operation name="getResolution">
40:         <input wsam:Action="http://control/Pwm/getResolutionRequ
          est" message="tns:getResolution"/>
41:         <output wsam:Action="http://control/Pwm/getResolutionRes
          ponse" message="tns:getResolutionResponse"/>
42:       </operation>
43:       <operation name="getCycle">
44:         <input wsam:Action="http://control/Pwm/getCycleRequest"
          message="tns:getCycle"/>
45:         <output wsam:Action="http://control/Pwm/getCycleResponse
          " message="tns:getCycleResponse"/>
46:       </operation>
47:       <operation name="setCycle">
48:         <input wsam:Action="http://control/Pwm/setCycleRequest"
          message="tns:setCycle"/>
49:         <output wsam:Action="http://control/Pwm/setCycleResponse
          " message="tns:setCycleResponse"/>
50:       </operation>
51:       <operation name="getDutyCycle">
52:         <input wsam:Action="http://control/Pwm/getDutyCycleReque
          st" message="tns:getDutyCycle"/>
53:         <output wsam:Action="http://control/Pwm/getDutyCycleResp
          onse" message="tns:getDutyCycleResponse"/>
54:       </operation>
55:       <operation name="setDutyCycle">
56:         <input wsam:Action="http://control/Pwm/setDutyCycleReque
          st" message="tns:setDutyCycle"/>
57:         <output wsam:Action="http://control/Pwm/setDutyCycleResp
          onse" message="tns:setDutyCycleResponse"/>
58:       </operation>
59:     </portType>
60:     <binding name="PwmPortBinding" type="tns:Pwm">
61:       <soap:binding transport="http://schemas.xmlsoap.org/soap/h
          ttp" style="document"/>
62:       <operation name="getResolution">
63:         <soap:operation soapAction=""/>
64:         <input>
65:           <soap:body use="literal"/>
66:         </input>
67:         <output>
68:           <soap:body use="literal"/>
69:         </output>
70:       </operation>
71:       <operation name="getCycle">
72:         <soap:operation soapAction=""/>
73:         <input>
74:           <soap:body use="literal"/>
```

```
 75:         </input>
 76:         <output>
 77:           <soap:body use="literal"/>
 78:         </output>
 79:       </operation>
 80:       <operation name="setCycle">
 81:         <soap:operation soapAction=""/>
 82:         <input>
 83:           <soap:body use="literal"/>
 84:         </input>
 85:         <output>
 86:           <soap:body use="literal"/>
 87:         </output>
 88:       </operation>
 89:       <operation name="getDutyCycle">
 90:         <soap:operation soapAction=""/>
 91:         <input>
 92:           <soap:body use="literal"/>
 93:         </input>
 94:         <output>
 95:           <soap:body use="literal"/>
 96:         </output>
 97:       </operation>
 98:       <operation name="setDutyCycle">
 99:         <soap:operation soapAction=""/>
100:         <input>
101:           <soap:body use="literal"/>
102:         </input>
103:         <output>
104:           <soap:body use="literal"/>
105:         </output>
106:       </operation>
107:     </binding>
108:     <service name="pwm">
109:       <port name="PwmPort" binding="tns:PwmPortBinding">
110:         <soap:address location="http://localhost:8080/gatewayWS/
           pwm"/>
111:       </port>
112:     </service>
113:   </definitions>
```