AISLAN GOMIDE FOINA

**MONITORAÇÃO DE REDE DE SENSORES COM
TRANSPONDERS**

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

São Paulo

2007

AISLAN GOMIDE FOINA

**MONITORAÇÃO DE REDE DE SENSORES COM
TRANSPONDERS**

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

Área de Concentração:
Sistemas Eletrônicos

Orientador:
Francisco Javier Ramirez Fernandez

São Paulo
2007

FICHA CATALOGRÁFICA

Foina, Aislan Gomide

Monitoração de rede de sensores com transponders / A. G. Foina. -- São Paulo, 2007.

91 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Middleware 2.RFID 3.Monitoração de processos I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

A meus pais, Paulo e Luciana, que sempre foram meus professores do dia-a-dia; a meu irmão, Ariel, cujas divergências sempre nos rendiam horas de discussão; e a meu avô, Rubens, o culpado pela minha paixão pela Engenharia.

AGRADECIMENTOS

Após dois longos anos de dias e noites de trabalho imensurável, dificuldades para conseguir equipamentos, tempo para conciliar todas as tarefas pessoais e profissionais e as vezes falta de energia para continuar acordado para mais uma hora de trabalho, muitas foram as pessoas que colaboraram com este trabalho direta ou indiretamente. Tantos palpites, comentários, sugestões, críticas e brigas fizeram com que esta dissertação atingisse o seu formato final.

O primeiro que merece agradecimento é meu orientador e amigo, Prof. Dr. Francisco Javier Ramirez, que sua experiência e conhecimento colaboraram de forma única com este trabalho, sem falar das longas e duradouras discussões sobre temas sobre engenharia e pesquisa.

Aos meus pais, que também tiveram uma colaboração de extrema importância pelo apoio emocional, conselhos e sugestões. Agradeço pela revisão técnica realizada pelo meu pai, Paulo Rogério Foina, e pela revisão gramatical feita pela minha mãe, Luciana de Mello Gomide.

Ao meu irmão Ariel que sempre me rendeu longas conversas sobre assuntos aleatórios; à minha avó Anita que me sempre me deu abrigo e carinho quando precisei; ao meu falecido avô Rubens, por todos os ensinamentos sobre engenharia que ele me passou; à minha avó Wanda e avô Herval, que apesar do tempo e da distância que nos separa, são sempre pessoas importantes na minha vida.

À Jú, amiga e “irmã”, às vezes até inimiga, mas sempre ao meu lado tentando enriquecer o meu vocabulário e me tornar uma pessoa mais culta. Graças a seu conhecimento em inglês, sempre fez ótimas traduções e correções.

Aos meus grandes amigos, especialmente Mateus, Emille, Henrique, Cris, Alvinho, Rafael Máquina, Mari, Neto, Drudi, Baixinho e Julia, que tiveram uma enorme importância na minha vida, me apoiaram enquanto este texto ainda estava no início e comemoraram comigo a cada etapa pessoal e profissional concluída.

À FAPES, CNPq e todas as outras instituições de fomento à pesquisa, pelos seus apoios em minha vida acadêmica, tentando diminuir o grande obstáculo criado pelo Brasil para a pesquisa e desenvolvimento.

Ao LME, inclusive as pessoas que lá trabalham, por ter me acolhido quando eu era um simples estudando de 2º ano e que esteve me apoiando durante toda a minha vida acadêmica.

A todas as empresas que colaboraram com equipamentos, informações, apoio e contatos, entre elas a Korth, a GTP e a Activewave.

RESUMO

Este trabalho apresenta os resultados obtidos no desenvolvimento de um sistema capaz de estabelecer um controle adequado do fluxo da informação e a supervisão de uma rede de sensores. O sistema se caracteriza pelo alto grau de flexibilidade possuindo uma camada de comunicação de dispositivos, uma camada de gerenciamento de regras de negócio para controlar o fluxo de processos, uma camada de interface homem - máquina e de interface com outros sistemas.

A camada de comunicação com os dispositivos é responsável pela interface entre os mesmos e o núcleo do sistema, integrando, de forma transparente, diferentes equipamentos, tecnologias e fabricantes. A interface do usuário foi projetada em um único módulo para facilitar modificações sem comprometer o funcionamento geral do sistema. A interface com outros sistemas é feita por meio de *drivers* de comunicação, permitindo qualquer tipo de integração. O núcleo do sistema faz todo o controle de eventos, do fluxo do processo e geração alarmes, assim como recebimento e envio das informações da camada de dispositivos e da camada de interface.

São descritas as diversas camadas da interface e sua implementação analisando as possíveis aplicações do sistema, juntamente com um estudo de caso do controle do processo de descarga de carga a granel no porto de Santos, usando tecnologia RFID. Os resultados obtidos nessa aplicação são descritos e comprovam a utilidade prática do sistema. Apresentam-se inicialmente alguns conceitos básicos necessários ao entendimento deste projeto como: sistemas distribuídos, estrutura *webservice*, linguagem XML e de tecnologias passíveis de integração com o sistema.

ABSTRACT

This paperwork presents the results obtained with the development of a system capable of establishing a proper control of information flow and supervision of a sensors network. The system characterizes itself due to its great flexibility degree by having a devices communication layer, a business management layer to do the process flow control, a man-machine interface and interface with other systems layer.

The devices communication layer is responsible for the interface between other devices and the system core, integrating in a transparent way, different equipment, technologies and manufactures. The user's interface was designed in a single module to facilitate modifications without compromising the system's general functioning. The interface with other systems is made throughout communication drivers, allowing any type of integration. The system core makes a control of all events, process flow and alarm generation, as well as receiving and sending information from the devices layer and interface layer.

The several interface layers and its implementation are described analyzing the system's possible applications along with a case study of loading discharge in a bulk process control at Santos Port using RFID technology. The results obtained with this application are described and prove the practical utility of the system. Some necessary basic concepts to understand this project are presented initially as: distributed systems, web service structure, XML language and technologies subjected to integration with the system.

SUMÁRIO

Resumo.....	7
Abstract.....	8
Índice de Figuras.....	11
Tabela de Abreviaturas.....	13
Capítulo 1 - Introdução	15
1.1. Objetivo.....	16
1.2. Sistemas de monitoramento	16
1.3. Organização do Texto	17
Capítulo 2 - Fundamentos Básicos.....	18
2.1. Middleware	18
2.2. Sistemas distribuídos	18
2.3. Estrutura webservice.....	20
2.4. Linguagem XML	20
2.5. Tecnologias de aquisição de dados	22
2.5.1. Transdutores Inteligentes	22
2.5.2. Sistemas de RFID	23
Capítulo 3 - Estrutura do sistema.....	25
3.1. Controle de dispositivos.....	26
3.2. Interface do usuário.....	26
3.3. Interligação com aplicações.....	27
3.4. Núcleo do sistema.....	28
Capítulo 4 - Implementação do sistema	30
4.1. DMS.....	30
4.1.1. Estrutura e funcionamento	30
4.1.2. Protocolo XML	32
4.2. MCS	33
4.2.1. Simulador.....	34
4.2.2. Leitor Feig Eletronic	35
4.2.3. Protocolo serial	35
4.2.4. Confecção da antena	37

4.2.5.	Leitor Korth KT-38.....	39
4.2.6.	Leitor ActiveWave ativo.....	40
4.3.	Core.....	41
4.3.1.	Fluxo de comunicação interna	41
4.3.2.	Estrutura Webservice	42
4.3.3.	Banco de dados	43
4.4.	Parms.....	50
4.5.	ECS	53
4.6.	ALMS	56
4.7.	LSI.....	58
4.7.1.	Integração por envio de arquivos em uma pasta compartilhada	59
4.8.	Command.....	60
4.8.1.	Tela de Consulta	60
4.8.2.	Tela de cadastro	61
4.8.3.	Tela do mapa.....	61
Capítulo 5 -	Aplicações práticas do Middleware.....	63
5.1.	Processo de descarga de cargas a granel.....	63
5.2.	Implementação do fluxo no sistema	67
5.3.	Tecnologias Utilizadas.....	70
5.4.	Resultados	73
Capítulo 6 -	Conclusões e planos futuros	77
6.1.	Planos Futuros.....	77
Anexo A -	Especificação dos equipamentos utilizados	79
Anexo B -	Mensagens XML usadas no estudo de caso	83
B.1.	DMSAuthenticationRequest	83
B.2.	DMSAuthenticationResponse.....	83
B.3.	DMSConfigurationRequest.....	84
B.4.	DMSConfigurationResponse	84
B.5.	LSIPDAMessage.....	85
B.6.	DMSPDAMessage.....	85
B.7.	LSIWeightRequest.....	86
B.8.	LSIWeightResponse	86
Bibliografia		88

ÍNDICE DE FIGURAS

Figura 1 – Arquitetura básica de um sistema distribuído	19
Figura 2 - Modelo geral de um sensor inteligente	23
Figura 3 - Componentes do sistema de RFID	24
Figura 4 - Estrutura modular do sistema.....	25
Figura 5 – Módulo de controle dos dispositivos (DMS).....	26
Figura 6 - Módulo de interface com o usuário.....	27
Figura 7 - Módulo de interligação com outros sistemas (LSI)	27
Figura 8 - Módulo principal do sistema.....	28
Figura 9 - Etapas de início do DMS.....	31
Figura 10 - Descrição do protocolo serial do leitor ID.ISC.MR100.....	36
Figura 11 - Fluxo de informações no DMS	37
Figura 12 - Seqüência de comandos para uso da API e do leitor da ActiveWave.....	41
Figura 13 - Fluxo de Informações no Core.....	41
Figura 14 - Arquivo de propriedades do JDBC com drivers MySQL, Postgres, ODBC e SQLServer.....	44
Figura 15 - Tabelas de informações de baixo nível	44
Figura 16 - Tabelas de parametrização	45
Figura 17 - Tabelas de histórico do sistema.....	46
Figura 18 - Tabelas de instalação dos equipamentos.....	47
Figura 19 - Tabelas de cadastro do sistema	49
Figura 20 - Tabela de configuração do sistema	50
Figura 21 - Diagrama de divisão de um processo de exame médico.....	51
Figura 22 - Tabela dos processos.....	51
Figura 23 - Tabela das tarefas.....	51
Figura 24 - Tabela de atividades.....	52
Figura 25 - Tabela de instalações.....	52
Figura 26 - Tabela de inserção dos eventos.....	53
Figura 27 - Tabela da fila de tarefas.....	54
Figura 28 - Registro histórico de atividades.....	55
Figura 29 - Tabela de registro histórico.....	55
Figura 30 - Tela de controle.....	57

Figura 31 - Tela de controle com o Pop-Up	58
Figura 32 - Tela de consulta.....	61
Figura 33 - Tela de cadastro de unidades.....	61
Figura 34 - Tela do mapa.....	62
Figura 35 - Fluxo de descarga de carga a granel	63
Figura 36 - (a) Caminhão caçamba basculando; (b) Caminhão graneleiro na moega.	64
Figura 37 - Fluxo completo de descarga.....	66
Figura 38 - Processo cadastrado no Parms	67
Figura 39 - Tarefa de pesagem bruta	67
Figura 40 - Tarefa de descarga.....	67
Figura 41 - Tarefa de pesagem tara.....	68
Figura 42 – Diagrama de estados do processo.....	69
Figura 43 - Tela de acompanhamento dos estados do processo	70
Figura 44 - Apresentação dos caminhões que estão no estado fila pesagem da tara...	70
Figura 45 - Alarme de violação de lacre eletrônico.....	71
Figura 46 - Tag cadeado	72
Figura 47 - Local de instalação dos leitores na balança.....	73
Figura 48 - Gráfico dos alarmes gerados	74
Figura 49 - Gráfico da economia gerada pelo sistema.....	74
Figura 50 - Gráfico de desempenho do sistema.....	75
Figura 51 - Imagens das variáveis do sistema operando com 60 caminhões simultâneos	76
Figura 52 - Fotos do leitor ID.ISC.MR100 e da antena montada.....	79
Figura 53 - Especificação do leitor ID.ISC.MR100	80
Figura 54 - Fotos e especificação do leitor KT-38	81
Figura 55 - Fotos e especificação do leitor da ActiveWave	82

TABELA DE ABREVIATURAS

ALMS:	<i>alarm management subsystem</i> (sistema de tratamento de alarmes)
API:	<i>application program interface</i> (aplicativo de interface de programa)
BCC:	<i>business component composer</i> (compositor de componentes de negócio)
BMI:	<i>business model interface</i> (interface de modelo de negócios)
BPA:	<i>business performance analysis subsystem</i> (sistema de análise de performance do negócio)
CCO:	Centro de Controle Operacional
Command:	<i>control, monitoring, management and dispatch subsystem</i> (sistema de controle, monitoração, gerência e despacho)
CPU:	<i>central processing unit</i> (unidade central de processamento)
CSW:	<i>core system framework</i> (núcleo do sistema)
DLL:	<i>dynamic-link library</i> (biblioteca de ligação dinâmica)
DMS:	<i>device management subsystem</i> (sistema de administração de dispositivos)
DTD:	<i>document type definition</i> (definição de tipo de documento)
ECS:	<i>event control subsystem</i> (sistema de controle de eventos)
FIFO:	<i>first in, first out</i> (primeiro a entrar, primeiro a sair)
GPS:	<i>global positioning system</i> (sistema de posicionamento global)
HF:	<i>high frequency</i> (frequência alta)
HTML:	<i>hypertext markup language</i> (linguagem de formatação de hipertexto)
ISO:	<i>international organization for standardization</i> (organização internacional de padronização)
JDBC:	<i>Java database connectivity</i> (conectividade Java a base de dados)
JNI:	<i>Java native interface</i> (Interface Nativa do Java)
LF:	<i>low frequency</i> (baixa frequência)
LSI:	<i>legacy system integration subsystem</i> (sistema de integração com sistema legado)
MCS:	<i>middleware component subsystem</i> (sistema de componentes de interligação)
Parms:	<i>system parameters configuration subsystem</i> (sistema de configuração de parâmetros do sistema)
PEPS:	primeira entrada primeira saída
RF:	radio frequência

RFID: *radio-frequency identification* (identificação por radiofrequência)

SGML: *standard generalized markup language* (linguagem padronizada de marcação genérica)

SQL: *structured query language* (linguagem de consulta estruturada)

SW: *software* (programa)

UHF: *ultra-high frequency* (ultra-alta frequência)

URL: *uniform resource locator* (localizador de recursos)

USB: *universal serial bus* (barramento serial universal)

VPN: *virtual private network* (rede virtual privada)

W3C: *world wide web consortium* (consórcio da rede mundial de informação)

XML: *extensible markup language* (linguagem marcação extensiva)

CAPÍTULO 1 - INTRODUÇÃO

Em vários contextos existe a necessidade de monitorar uma rede de sensores, por exemplo, a posição de um objeto ou uma pessoa dentro de um determinado ambiente, controlando uma série de regras de negócio e gerando alarmes, caso as regras não sejam cumpridas. Essa necessidade pode ser gerada por razões de segurança, prevenindo a movimentação indevida de ativos de alto valor agregado, ou pode ser gerada por razões logísticas, analisando os perfis, tempos e gargalos de um dado fluxo de processo. Um sistema que gerencia esses tipos de informações deve ser flexível o suficiente para atender os mais diversos cenários e se adaptar aos diferentes processos existentes (LEGNER, 2006).

O interesse crescente por novas tecnologias e arquiteturas para o desenvolvimento de sistemas distribuídos é uma consequência dessa necessidade de flexibilidade. Na automação dos sistemas produtivos, por exemplo, essa tendência pode ser claramente vista, pois os *softwares* de controle são organizados como conjuntos de componentes com alto grau de cooperação (LEWIS, 2001). Este é um dos principais motivos pelos quais os softwares de supervisão e de controle estão tornando-se altamente complexos e possuindo um grande número de módulos com elevado grau de interação entre si, sendo que esses módulos normalmente são os elementos de *hardware* e *software* do sistema (SANTOS FILHO, 2000). Os mesmos princípios podem ser aplicados aos sistemas de monitoramento e gerenciamento de processos.

Do ponto de vista do *hardware* envolvido, os mais variados equipamentos podem ser utilizados para realizar esse monitoramento. Qualquer tipo de sistemas de aquisição de dados, ou rede de sensores, pode ser integrado ao sistema (CHOI, 2004) (VIEGAS, 2003). Entre eles, inclui-se a identificação por radiofrequência, ou RFID, que utiliza comunicação de rádio a curto alcance para enviar uma identificação (NI, 2003) (KALEJA, 1999). Outro equipamento possível de se utilizar é o sistema de posicionamento global, ou GPS, que acoplado a um modem de celular ou de satélite, permite sua localização em qualquer lugar do mundo (BAKHURU, 2005).

Para que um *software* possa realizar sua função sem ficar vinculado a alguma tecnologia específica é necessário que exista uma camada de gerenciamento de dispositivos, conhecida como *middleware* (XIONG, 2001). Essa camada isola a comunicação dos dispositivos com o restante do sistema, permitindo que o equipamento utilizado seja transparente para o sistema.

Baseado nessas informações este trabalho apresenta o desenvolvimento de um sistema de estrutura modular de supervisão de uma rede de sensores e gerenciamento dos eventos gerados por eles, com o intuito de controlar o processo no ambiente monitorado pelos sensores. Além dessa funcionalidade primordial, o sistema deve gerar relatórios e alarmes quando necessário, sendo que a tecnologia utilizada para esse rastreamento se integra de forma transparente devido a um *middleware*.

1.1. Objetivo

O objetivo deste projeto é desenvolver um *middleware* para gerenciamento de uma rede de sensores que possua um sistema de supervisão de dispositivos, com o propósito de controlar as regras de negócios configuradas no sistema, publicar relatórios e gerar alarmes caso ocorra o descumprimento de alguma regra parametrizada.

O desenvolvimento deste sistema foi feito utilizando majoritariamente leitores de RFID como dispositivos de aquisição de dados para controlar regras de acessos e permanência em cômodos ou ambientes fechados. Porém o sistema se mostrou bastante flexível para gerenciar outros tipos de dispositivos tais como rastreadores veiculares e sensores inteligentes.

1.2. Sistemas de monitoramento

Atualmente existem diversos *middlewares* para gerenciamento de redes de sensores (SUN, 2006) (IBM, 2006) (XIONG, 2001). Porém eles são focados somente na interligação da camada de dispositivos com a camada de aplicativos, ou seja, esses sistemas não executam nenhuma análise nos dados que são enviados de um lado para o outro. Como foi bem comentado por Kermer sobre *middlewares* de RFID, a maturação da tecnologia está bem além da camada do *middleware* (KERMER, 2004). Ou seja, os sistemas de gerenciamento dos dispositivos devem ser muito mais que simples aplicações desempenhando o papel de ponte de dados (LEAVER, 2004).

A maioria dos *middleware* atualmente disponíveis, poucos se diferenciaram de um mero sistema de tradução e troca de mensagens. Molla apresenta um comparativo dedicado somente a *middlewares* direcionados para rede de sensores, e nenhum possui a inteligência necessária para realizar um gerenciamento de processo (MOLLA, 2006). Uma proposta publicada por Liu, mostrando as vantagens de se agregar um *middleware* a um sistema de gerenciamento de processos de negócios é inovadora, mas não foi desenhado para gerenciar dispositivos físicos, e sim para operar com outros sistemas usando tecnologia de *webservices* (LIU, 2003). Da mesma maneira, os sistemas de gerenciamento de processos não possuem a conectividade necessária para estabelecer uma comunicação com outras plataformas e nem com equipamentos (WU, 2005).

Assim, a contribuição deste trabalho está concentrada no desenvolvimento da junção dos dois sistemas, o *middleware* voltado para rede de sensores e um sistema de gerenciamento de processos, para fechar essa lacuna entre os dois tipos de sistemas. Isso viabiliza a interligação de uma rede de sensores (RFID, sensores inteligentes, etc) com um sistema final inteligente que receba e processe os dados, gerando relatórios, alarmes e garantindo a execução correta do processo sob supervisão e, se for o caso, permitindo a conexão com outros sistemas.

1.3. Organização do Texto

O texto a seguir está dividido da seguinte forma: O primeiro capítulo oferece uma breve introdução do contexto no qual se insere o sistema desenvolvido neste trabalho; o segundo capítulo explica o funcionamento das tecnologias envolvidas no projeto, começando pelos fundamentos de *software* e finalizando com algumas das tecnologias integradas ao sistema; o capítulo 3 mostra a estrutura do sistema proposto neste projeto e explica a função de cada módulo e seus sub-módulos; o capítulo 4 detalha a metodologia usada na implementação de cada um dos módulos e sub-módulos descritos no capítulo anterior, explicando a funcionalidade de cada campo das tabelas da base de dados e como elas se relacionam; o capítulo 5 apresenta as possíveis aplicações do sistema incluindo um estudo de caso de monitoramento de caminhões com cargas a granel, e como o sistema foi configurado para tratar este fluxo; no capítulo 6 se incluem as conclusões obtidas com experiência na realização deste projeto e finalmente é incluída uma série de anexos técnicos mostrando detalhes de implementação e equipamentos usados neste projeto, e suas especificações.

CAPÍTULO 2 - FUNDAMENTOS BÁSICOS

2.1. *Middleware*

Middleware é uma nomenclatura utilizada normalmente para se referir a uma classe de *software* cujo objetivo é servir de ligação entre sistemas construídos separadamente (SANZ, 2003). Ele é a parte do *software* que fica entre o sistema operacional e o sistema distribuído. Ou em outras palavras, “*Middleware* é um conjunto de serviços, independentes de negócio, que permitem a interação entre aplicações e usuários finais através de uma rede de computadores. Em essência, *middleware* é o *software* que reside acima da rede e abaixo dos *softwares* de aplicações de negócio” (UMAR, 97).

Com a sua utilização, é possível prover uma interoperabilidade entre as interfaces de um sistema, de forma simples, sem a necessidade de alterações nos códigos mais importantes do sistema caso alguma das interfaces seja alterada, somente pequenas alterações no *middleware* já resolveriam a incompatibilidade.

O *middleware* é relativamente novo no mundo da computação. Ele ganhou popularidade na década de 80 como uma solução para problemas de como ligar novas aplicações a antigos sistemas legado, apesar do que o termo tem sido usado desde 1968. Ele também tem facilitado o processamento distribuído – conexão de múltiplas aplicações para criar uma aplicação maior, normalmente através de uma rede de dados. Informações mais detalhadas sobre *middleware* podem ser encontradas no trabalho de Bray (BRAY, 2005).

2.2. *Sistemas distribuídos*

O conceito de sistemas distribuídos é um paradigma de programação focando no desenho de um sistema que seja distribuído, aberto, escalável, transparente e tolerante a falha. Esse paradigma é um resultado natural do uso de computadores para formar uma rede. Um sistema distribuído é descentralizado e trabalha paralelamente, usando dois ou mais computadores se comunicando sobre uma rede para completar uma tarefa ou objetivo comum. Os tipos de *hardware*, linguagem de programação, sistema operacional e outros recursos desses computadores podem variar drasticamente. A tendência de se adotar sistemas distribuídos está se tornando

crescente devido às ferramentas de programação que estão sendo disponibilizadas e o crescente aumento da banda de comunicação entre os sistemas, permitindo uma comunicação muito mais rápida entre os componentes envolvidos (HECK, 2003). Um sistema distribuído é similar a um sistema que é executado em um cluster, com a diferença que pode existir uma grande separação geográfica dos recursos.

Na literatura, sistemas distribuídos têm diversas definições. Tanenbaum, por exemplo, definiu da seguinte forma: “Sistema Distribuído é uma coleção de entidades que processam informações de modo independente; para os usuários isto deve ser transparente, ou seja, deve parecer um sistema único (*single virtual system* (PUTMAN, 2001)) que atende às suas necessidades” (TANENBAUM, 2002). Essa definição mostra a grande necessidade de interoperabilidade existente nos sistemas distribuídos. Na Figura 1 pode ser vista uma representação de um sistema distribuído sendo executado em diversos computadores que se comunicam através de uma rede de dados.

Figura 1 – Arquitetura básica de um sistema distribuído

As vantagens de se usar essa arquitetura são varias. Os sistemas distribuídos são abertos, ou seja, já estão predispostos a se comunicar com outros sistemas devido a sua necessidade de se comunicar internamente. Eles utilizam os protocolos de *webservice* para serem estendíveis e escaláveis, ou seja, caso seja demandado pela aplicação, mais recursos computacionais (computadores) podem ser agregados ao sistema distribuído sem que isso impacte no funcionamento do sistema como um todo, e sem a necessidade de alterar códigos fonte.

2.3. Estrutura webservice

A W3C define *webservice* como um software desenhado para suportar interoperabilidade e interação máquina-a-máquina sobre uma rede. Outros sistemas interagem com o *webservice* de maneiras prescrita por sua descrição usando mensagens SOAP, normalmente enviadas usando HTTP com uma mensagem XML em conjunto com outros padrões relacionados à Web (WSA, 2004). Essa definição engloba muitos sistemas diferentes, mas o uso comum deste termo se refere a sistemas usando mensagens XML para troca de mensagens.

O *webservice* define um meio padrão de interoperabilidade entre diferentes aplicações, sendo executadas em uma variedade de plataformas e/ou *frameworks*. A arquitetura *webservices* é uma arquitetura de interoperabilidade, ela identifica os elementos globais de uma rede global de serviços que são requeridos para garantir a interoperabilidade entre os *webservices*. O trabalho de Cabrera mostra informações mais detalhadas sobre *webservices* (CABRERA, 2004).

2.4. Linguagem XML

XML é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. É um subtipo de SGML (linguagem padronizada de marcação genérica) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet. Entre linguagens baseadas em XML incluem-se XHTML (formato para páginas Web), RDF, SMIL, MathML (formato para expressões matemáticas), NCL, XBRL, XSIL e SVG (formato gráfico vetorial).

Estimulado pela insatisfação com os formatos existentes (padronizados ou não), um grupo de empresas e organizações que se autodenominou *World Wide Web Consortium* (W3C), começou a trabalhar em meados da década de 1990 em uma linguagem de marcação que combinasse a flexibilidade da SGML com a simplicidade da HTML. O princípio do projeto era criar uma linguagem que pudesse ser lida por *software*, e integrar-se com as demais linguagens. Sua filosofia seria incorporada por vários princípios importantes:

- Separação do conteúdo da formatação;
- Simplicidade e legibilidade, tanto para humanos quanto para computadores;

- Possibilidade de criação de tags sem limitação;
- Criação de arquivos para validação de estrutura (chamados DTDs);
- Interligação de bancos de dados distintos;
- Concentração na estrutura da informação, e não na sua aparência

Um banco de dados pode, através de uma aplicação, escrever em um arquivo XML e um outro banco distinto pode ler estes mesmos dados.

Como trata-se de um formato texto-puro, XML pode ser criado e editado em qualquer editor de textos moderno, suportando ainda a maioria das codificações de caractere (como ISO-8859-1 e UTF-8). Conhecendo-se a sintaxe de XML e o DTD, pode-se escrever documentos XML manualmente tão válidos quanto os gerados por programas automatizados.

O exemplo abaixo demonstra a sintaxe flexível do XML sendo usada para descrever uma receita de pão:

```
<?xml version="1.0" encoding="UTF-8"?>
<Receita nome="pão" tempo_de_preparo="5 minutos"
tempo_de_cozimento="1 hora">
  <título>Pão simples</título>
  <ingrediente quantidade="3"
unidade="xícaras">Farinha</ingrediente>
  <ingrediente quantidade="7"
unidade="gramas">Fermento</ingrediente>
  <ingrediente quantidade="1.5" unidade="xícaras"
estado="morna">Água</ingrediente>
  <ingrediente quantidade="1" unidade="colheres de
chá">Sal</ingrediente>
  <Instruções>
    <passo>Misture todos os ingredientes, e dissolva
bem.</passo>
    <passo>Cubra com um pano e deixe por uma hora em
um local morno.</passo>
    <passo>Misture novamente, coloque numa bandeja e
asse num forno.</passo>
  </Instruções>
</Receita>
```

Onde temos na 1ª linha:

```
<Receita nome="pão" tempo_de_preparo="5 minutos"
tempo_de_cozimento="1 hora">
```

"Receita" é o nome principal do documento. A semelhança entre XML e HTML é grande, na 1ª linha é aberto a *tag* (não confundir com *tag* de RFID)

Receita e na última linha ela é fechada, como em HTML, e assim se estende por todo o exemplo.

Em Becker é citado o uso crescente do XML como formato de codificação de mensagens e a sua utilização como meio de integração entre aplicações (BECKER, 2000). Também é considerada a necessidade de integração dos vários sistemas não interferir de modo negativo nas aplicações já existentes nos mesmos.

2.5. Tecnologias de aquisição de dados

Diversas tecnologias de rede de sensores podem ser utilizadas com o sistema desenvolvido. Algumas das mais significativas são transdutores inteligentes e dispositivos de RFID, devido a seu grau de inovação e popularização desses dispositivos no meio acadêmico.

2.5.1. Transdutores Inteligentes

Transdutores (sensores e atuadores) servem a uma grande variedade de necessidades industriais, manufatura, automotiva, etc. Desde que o mercado de transdutores ficou muito diversificado, os seus fabricantes vêm buscando formas de construir sensores inteligentes, baratos, sem fio e ligados em rede (BERNI, 2004). Muitas redes de controle de sensores estão disponíveis no mercado, cada uma com uma vantagem e desvantagem para uma dada aplicação (MA, 2004). O acréscimo de um microprocessador nos transdutores para manejar a comunicação nessas redes já dá um nível de inteligência a esses dispositivos, permitindo-lhes fazer análises dos seus dados medidos e, às vezes, até permite a tomada de decisão de um atuador, baseado na informação enviada diretamente por um sensor próximo, sem passar por algum sistema central de controle.

O sensor inteligente é um simples sensor que trata sua própria aquisição e converte essa informação para um resultado calibrado em unidades do atributo físico medido. Por exemplo, um termopar tradicional simplesmente fornece uma tensão analógica de saída. O voltímetro é responsável por receber essa tensão e transformá-la em temperatura medida através de uma série de algoritmos complexos e de uma conversão analógica para digital. Um sensor inteligente deve fazer isso tudo internamente e simplesmente fornecer o valor da temperatura como um dado. Sensores inteligentes não fazem julgamentos do dado coletado, a não ser que ele fuja

do alcance do sensor. Na Figura 2 está apresentado um modelo geral de um sensor inteligente.

Padrões vêm surgindo para definir a comunicação dos transdutores inteligentes com a rede e entre si. Um dos padrões mais citados na atualidade é o IEEE 1451 (IEEE, 2000), que define uma coleção de padrões de interface para sensores e atuadores inteligentes. Essa coleção de padrões tem a finalidade de facilitar o trabalho dos fabricantes de transdutores ao produzirem dispositivos inteligentes e de permitir a interface desses dispositivos em uma rede, sistema e instrumentos, apenas incorporando tecnologias de sensoriamento e de rede existentes e emergentes.

Figura 2 - Modelo geral de um sensor inteligente

2.5.2. Sistemas de RFID

Nos sistemas de RFID normalmente há três tipos de componentes básicos, como pode ser visto na Figura 3: a antena, o leitor e o transponder ou *tag*. O transponder é um chip, que submetido à ação de um campo eletromagnético de uma dada frequência, é energizado, gerando um sinal de resposta. A antena é responsável pela recepção do sinal do *tag* e pela transmissão dos comandos aos *tags*. O leitor é responsável pela interpretação do sinal recebido pela antena, e possui interfaces de comunicação RS232, RS485, USB ou Ethernet, para ser ligada a um computador. Assim o leitor gera um sinal numa frequência específica (LF, HF, UHF ou Wifi) e a antena transmite esse sinal numa dada área. Se nesta área estiver presente algum transponder, o mesmo irá transmitir um sinal de resposta que será recebido pela

antena e processado pelo leitor. Finalmente, o leitor passa esses dados processados para o computador via interface de comunicação.

Figura 3 - Componentes do sistema de RFID

O transponder pode ser do tipo passivo ou ativo. Os passivos não fazem uso de bateria, eles usam as ondas eletromagnéticas emitidas pela antena para alimentar seu circuito. Os ativos possuem uma pequena bateria interna com vida útil limitada para gerar a energia necessária para alimentar o circuito e enviar o sinal. Os sistemas de RFID com transponders passivos apresentam um custo unitário do *tag* menor que os sistemas que usam transponders ativos. Quando o sistema é voltado para controle de cadeia de suprimentos, pastas de processos e outros, o custo unitário deve ser baixo, assim neste projeto priorizam-se os transponders passivos. Já para aplicações onde o alcance e funcionalidades mais avançadas são requeridos, a escolha recai sobre os sistemas com *tags* ativos.

CAPÍTULO 3 - ESTRUTURA DO SISTEMA

Para atender os requisitos de flexibilidade e parametrização exigidos pelas aplicações mais modernas, o sistema proposto neste projeto possui uma estrutura modular com toda a sua comunicação interna baseada no protocolo XML. A estrutura geral do sistema proposto neste trabalho é mostrada na Figura 4. Inspirado em alguns trabalhos, como Curino e Shoshani, foi feita a divisão do sistema em três camadas: camada de aquisição de dados (verde clara), camada de tratamento dos dados (lilás) e camada e interfaces externas (azul) (CURINO, 2005) (SHOSHANI, 2002) (CHEN, 2003). A estrutura é baseada no modelo das 3 camadas, comumente usado em sistemas *web* (MEIJER, 2001).

Figura 4 - Estrutura modular do sistema

As setas no desenho mostram o caminho que a informação percorre desde sua origem em algum dispositivo até a sua chegada na interface gráfica e/ou seu envio para outros sistemas. Assim, conforme mostrado, o *software* possui quatro grandes módulos distintos, vistos um a um a seguir.

3.1. Controle de dispositivos

O módulo denominado Controle de Dispositivo (*Device Management Subsystem*) – DMS, conforme a descrição da Figura 5, é o responsável pela interface entre os dispositivos diversos e o núcleo do sistema. Ele integra, de forma transparente os equipamentos de diferentes marcas, tecnologias e fabricantes, e ainda permite a convivência com outras tecnologias de coleta de dados (como RFID e código de barra ao mesmo tempo, por exemplo).

Figura 5 – Módulo de controle dos dispositivos (DMS)

Este módulo permite ainda fazer a transição de processos controlados por outras tecnologias (código de barras, por exemplo) para um novo controle introduzindo uma tecnologia nova (RFID, por exemplo) de maneira gradual e com pouco impacto nos processos de negócio já existente. Esse gradualismo reduz a curva de aprendizagem dos funcionários envolvidos nesses processos acelerando a implantação da nova tecnologia. Pode-se também utilizar dois ou mais diferentes equipamentos simultaneamente para controlar o mesmo processo, como por exemplo, utilizar *tags* de RFID passivos com de curto alcance (aproximadamente 3 metros) e utilizar *tags* de RFID ativos de longo alcance (até 100 metros), e tudo isso sendo gerenciado pelo mesmo aplicativo de supervisão.

3.2. Interface do usuário

Toda a interface do usuário com o sistema foi projetada em um único módulo mostrado na Figura 6, para que possam ser feitas implementações em diversos idiomas (como os asiáticos), ou alterações no *layout* das telas de apresentação sem alterar o funcionamento do sistema como um todo. Este módulo contém dois subsistemas: Command e o BPA. O subsistema *Control, Monitoring, Management and Dispatch Subsystem* – Command é o responsável pela interface homem-máquina

de todo o sistema incluindo a apresentação da posição dos alvos no mapa, as informações de status, controles de alarmes, parametrização, etc. O subsistema *Business Performance Analysis Subsystem* – BPA é o responsável por gerar os relatórios de alertas, alvos que entraram em situação suspeita mais vezes, horários onde ocorre maior número de incidentes, entre outros.

Figura 6 - Módulo de interface com o usuário

3.3. Interligação com aplicações

O módulo chamado de Integração com Aplicações (*Legacy System Integration Subsystem*) – LSI, mostrado na Figura 7, implementa a interface de comunicação de dados para com outros sistemas legado, tais como sistemas de segurança ou de controle de estoque, para viabilizar a integração com o sistema de controle de processo.

Figura 7 - Módulo de interligação com outros sistemas (LSI)

3.4. Núcleo do sistema

O principal módulo do sistema é formado por quatro grandes módulos relacionados abaixo, conforme a descrição da Figura 8:

Figura 8 - Módulo principal do sistema

Controle de Sistema: Este sub-módulo possui dois subsistemas, o *System Parameters Configuration Subsystem* (Parms) e o *Middleware Component Subsystem* (MCS). O Parms é o responsável por todas as parametrizações possíveis do sistema, é nele que ficam guardados os parâmetros e as funcionalidades de cada parâmetro, como por exemplo, zonas liberadas, zonas proibidas, atitude a ser tomada em cada nível de alerta, etc. O MCS é a biblioteca de dispositivos, onde o módulo de controle de dispositivos (DMS) acessa para analisar que tecnologia está sendo usada.

Controle de Eventos: Este módulo possui também outros dois subsistemas, o *Event Control Subsystem* (ECS) - e o *Alarm Management Subsystem* (ALMS). O ECS é o subsistema responsável pelo tratamento de eventos. Cada evento gerado no sistema é analisado e armazenado na sua base de dados. Dependendo dos parâmetros e do tipo de evento o ALMS pode ser acionado. O ALMS é o subsistema responsável pelos diversos alarmes. Ele é o encarregado de acionar uma sirene específica, enviar mensagens para a segurança do prédio ou até acionar uma chamada para a polícia se houver um sistema de *call center* ligado ao módulo de interligação com aplicações (LSI).

Componentes de Negócio: Neste módulo estão os subsistemas responsáveis pela adaptação da aplicação para os diferentes requisitos de negócio. O *Business Model Interface* (BMI) é o subsistema onde se configuram as interfaces com aplicações que serão usadas, os tipos de dispositivos utilizados e a que eles estão ligados, etc. E o *Business Component Composer* (BCC) é onde se define o fluxo de

informações, quais atitudes tomar perante um evento de risco, e caso o estado de alarme continue, que outras atitudes serão tomadas, etc.

Núcleo do Sistema: Este é o módulo central da aplicação. O *Core System Framework* (CSW), é encarregado de garantir o funcionamento do sistema 24 horas por dia, sete dias por semana. Ele verifica se todos os módulos estão sendo operando, executa rotinas de teste para verificar o correto funcionamento dos mesmos e caso seja diagnosticado alguma falha, ele reinicia o módulo. Se o erro persistir, ele dispara um evento e pode até disparar um alarme.

CAPÍTULO 4 - IMPLEMENTAÇÃO DO SISTEMA

Neste capítulo estão descritas as implementações e a metodologia adotada em cada módulo, resultados deste projeto de pesquisa.

4.1. DMS

Como citado anteriormente, o sistema foi projetado para ser independente de tecnologia, ou seja, ele pode trabalhar com diferentes fabricantes de equipamentos de diferentes tecnologias sem ser necessário alterar seu código. O único trabalho que precisará ser feito, caso se deseje integrar outro equipamento ao sistema, será desenvolver o seu *driver* no MCS. Feito isso, o novo dispositivo passa a trabalhar de maneira transparente, e pode também trabalhar simultaneamente com outras marcas e modelos de equipamentos. Toda essa funcionalidade é feita por um módulo de gerenciamento de dispositivos chamado de DMS. O DMS é a única parte do sistema que é monolítico, já todo o resto do sistema é feito em estrutura *webservice*. Ele pode também ser executado em diversos computadores diferentes simultaneamente, cada um gerenciando um pequeno grupo de dispositivos locais como se fosse o cliente e o resto do sistema, o servidor. Pra isso, cada instância do DMS possui um identificador (ID) próprio atribuído em seu arquivo de configuração, que permite ao Núcleo diferenciá-los entre si.

4.1.1. Estrutura e funcionamento

A sua estrutura foi desenhada de forma a se conectar aos diversos tipos de equipamentos por meio de um *driver* localizado numa biblioteca de *drivers* chamada MCS, explicada detalhadamente mais abaixo. Como início foram adotados leitores de RFID para serem integrados no sistema, devido à sua disponibilidade no laboratório. Para o caso do RFID, os *drivers* são classes que implementam a classe abstrata *RFIDReader*, que por sua vez possui 3 métodos: um método de *start* com todas as tarefas de inicialização e conexão do leitor; um método de leitura que retorna um objeto *RFIDReaderData* com o ID do leitor, ID da antena e ID dos *tags* lidos; e um método de *ping*, que consiste em enviar pequenas e simples mensagens, forçando

uma resposta do dispositivo para verificar se o leitor está ativo e funcionando corretamente (essa resposta é um valor booleano verdadeiro ou falso).

Quando o DMS é iniciado, ele executa a seqüência de tarefas mostrada na Figura 9. Sua primeira tarefa é um pedido de autenticação ao Núcleo por meio de uma mensagem formato XML, para saber se essa instância do DMS é legítima ou não e qual é a sua configuração. Após a resposta positiva do Núcleo, o DMS envia uma mensagem XML de requisição de configuração para saber quais são os tipos de leitores a que ele deve se conectar e quais *drivers* do MCS ele tem que utilizar. Essa mensagem XML é recebida pelo Núcleo, que faz uma consulta ao Parms para saber as configurações dessa instância do DMS, e retorna a relação de leitores e *drivers* que o DMS vai utilizar. Essas mensagens podem ser vistas com detalhe no Anexo B.

Figura 9 - Etapas de inicio do DMS

Sabendo quais os leitores e quais *drivers* utilizar, o DMS instancia os *drivers* criando um *thread* independente para cada dispositivo e inicia um *loop* infinito de requisições de leitura para os diferentes leitores. Simultaneamente, quando esse *loop* principal é iniciado, um *thread* de diagnóstico é iniciado também, e sua função é ficar enviando comandos de *ping* aos diferentes leitores para ver se estão funcionando

corretamente. Caso seja detectada uma falha, o objeto do *driver* do leitor é destruído e re-instanciado. Novamente um *ping* lhe é enviado, e caso o problema persista, então o *thread* de diagnóstico retira o leitor da lista de leitores ativos e mantém esse *loop* de instanciar, enviar um *ping* e destruir até que o leitor volte a responder, colocando-o novamente na lista de leitores ativos.

4.1.2. Protocolo XML

A mensagem XML de comunicação entre o módulo DMS e o módulo Core foi definida conforme a Tabela 1.

<!ELEMENT	RFIDReaderPacketSend	(Reader*)>
<!ELEMENT	Reader	(Time, TagList)>
<!ATTLIST	Reader	
	Id	CDATA #REQUIRED>
	Antenna	CDATA #REQUIRED>
<!ELEMENT	Time	(#CDATA)>
<!ELEMENT	TagList	(Size, TagId*)>
<!ELEMENT	Size	(#CDATA)>
<!ELEMENT	TagId	(#CDATA)>
<!ATTLIST	TagId	
	item	CDATA #REQUIRED>

Tabela 1 – XML entre o DMS e o Núcleo

O atributo ID do campo *Reader* indica qual o identificador do leitor que está enviando o pacote de leitura; o atributo *Antenna* indica o identificador da antena. Na maioria dos equipamentos utilizados, os identificadores da antena e do leitor possuem o mesmo valor, salvo os casos em que o equipamento permite a conexão de mais de uma antena, como os leitores passivos que operam na frequência de 915MHz e alguns leitores ativos. Cada antena envia uma lista de *tags* que possui um campo informando quantos *tags* foram lidos e, para cada um dos *tags*, o seu identificador único, *Unique ID*, e um campo de índice de todos os *tags* lidos com seu valor variando de 1 ao número de *tags* do pacote.

Assim, o formato geral da mensagem XML de leitura, que contém as informações lidas pelos tags, segue o seguinte padrão:

```
<?xml version="1.0" encoding="UTF-8"?>
<RFIDReaderPacketSend>
```



```

<Reader Id="1" Antenna="1">
  <Time>07/19/2006 20:00:10.656</Time>
  <TagList>
    <Size>8</Size>
    <TagId item="1">E00700000188F030</TagId>
    <TagId item="2">E00700000188F031</TagId>
    <TagId item="3">E007000017F29173</TagId>
    <TagId item="4">E007000017F28FB6</TagId>
    <TagId item="5">E007000001BDE4B7</TagId>
    <TagId item="6">0000000006018C3A</TagId>
    <TagId item="7">0000000006018C3B</TagId>
    <TagId item="8">0000000006018C3C</TagId>
  </TagList>
</ReaderAntenna>
</RFIDReaderPacketSend>

```

Portanto, para cada pacote de informações enviado pela leitora RFID e recebido pelo DMS, uma mensagem em formato XML deste é gerada e enviada para o Core, onde é recebido pelo módulo de tratamento de eventos. Neste módulo, as informações contidas na mensagem serão gravadas no banco de dados e adicionadas da posição da antena que leu os tags. A posição da antena é obtida a partir da consulta ao módulo Pams sobre a posição da antena que possui o ID informado na mensagem.

4.2. MCS

A biblioteca de *drivers* usada pelo DMS para se conectar com as mais diferentes tecnologias chama-se *Middleware Component Subsystem* – MCS. O MCS na verdade consiste em uma coletânea de classes que herdam uma classe abstrata. Para o caso de equipamentos de RFID, essa classe chama `RFIDReader` e possui a seguinte declaração:

```

public abstract class RFIDReader {

    protected boolean readerStatus;
    protected int readerId;

    public abstract int startReader ();
    public abstract boolean pingReader ();
    public abstract RFIDReaderData readData ();
}

```

O método `startReader()` tem o papel de inicialização do dispositivo. Alguns equipamentos de RFID exigem o envio de alguns parâmetros de configuração na inicialização, como é o caso do leitor da Feig. Então, todos esses comandos e

parâmetros devem ser enviados quando o método `startReader()` é chamado, o que normalmente acontece no construtor do *driver* do leitor.

O próximo método, `pingReader()` é o método chamado na rotina de auto-diagnóstico do DMS, onde é enviado um comando simples de interrogação e aguardada a resposta. Caso a resposta não retorne ou retorne com um valor não esperado, o método `pingReader()` retorna um valor “falso”, o que faz o DMS destruir a instância do *driver* e iniciar novamente. Se o leitor responder o *ping* com a resposta esperada, o método retorna um valor “verdadeiro”.

Finalmente, o método `readData()` é onde ocorre a leitura dos tags. Esse método retorna um objeto `RFIDReaderData`, que contém a seguinte declaração:

```
public class RFIDReaderData {  
  
    protected int idReader;  
    protected int idAntenna;  
    protected String[] idTag;  
    protected Date time;  
  
    public RFIDReaderData() {  
        time = new Date(System.currentTimeMillis());  
    }  
}
```

Como citado anteriormente, o `RFIDReaderData` possui 4 atributos: o `idReader`, que possui o valor do ID do leitor que realizou a leitura; o `idAntenna`, que possui o valor do ID da antena; o `idTag`, que é um vetor dos tags lidos; e o `time`, que registra a hora exata em que a leitura foi realizada. Essas quatro variáveis têm todas as informações necessárias para que o DMS gere a mensagem XML para o Núcleo. Assim, as classes `RFIDReader` e `RFIDReaderData` formam o modelo para o desenvolvimento dos *drivers* do MCS.

4.2.1. Simulador

Antes de iniciar os testes com os equipamentos, foi desenvolvido um simulador de uma rede de leitores para validar as rotinas do sistema. Este simulador tem implementado todas os métodos da classe `RFIDReader`, como se fosse um *driver* convencional do MCS, com a diferença que ele não se conecta a nenhum equipamento. A simulação é baseada num arquivo de configuração que contém os ID do *tag* e o ID do leitor para um número de segundos após a instância do *driver* do simulador. A seguir pode ser visto um trecho do arquivo de definição da simulação:

```
tag2=E001000000000001  
rdr2=4  
num2=22
```

```
tag3=E001000000000001  
rdr3=15  
num3=35
```

Cada grupo de três linhas consiste em um pacote. A 1ª linha é o ID do *tag*, a 2ª é o ID do leitor, e a 3ª linha é o número de segundos após a execução da função `startReader()` que deverá ser aguardado para que este pacote seja enviado. O simulador possibilita envio simultâneo, permitindo testar todas as funcionalidades como se fosse uma rede efetiva de leitores, bastando apenas escrever todo o *script* de simulação no formato descrito. O simulador foi uma ferramenta poderosa nos testes desse sistema.

4.2.2. Leitor Feig Eletronic

O sistema em vigor foi desenvolvido para utilizar o leitor ID.ISC.MR100. Sua tecnologia opera na frequência de HF (13.56 MHz) e é compatível com os *tags* da Philips (I-Code 1), da Texas Instruments (Tag-it HF) e com o padrão ISO15693. Este leitor tem capacidade de leitura de 30 *tags* simultâneos por segundo, e transmite seus dados através da porta Serial (RS232).

Inicialmente, deve-se enviar uma série de parâmetros de configuração para o leitor, primeiramente enviando um *reset* para o CPU e para o circuito RF, depois a ativação da emissão de RF para que finalmente possam ser feitas leituras de *tags*. Essas etapas, juntamente com a parte de abertura da porta serial, escolha do *baudrate* e paridade, são feitas na chamada do método `startReader()`. As leituras dos *tags* são feitas pela execução do método `readData()` que é síncrono, ou seja, a sua execução só é concluída após o envio da resposta do resultado, e a rotina `pingReader()` envia um pedido de versão do *firmware* para verificar se o leitor está respondendo os comandos ou não.

4.2.3. Protocolo serial

Um aspecto crítico observado na instalação do sistema de RFID está associado à ausência das informações básicas da descrição do protocolo serial utilizado pelo leitor ID.ISC.MR100 da Feig Eletronic. Para enviar todos os comandos necessários seria preciso uma documentação completa do protocolo serial de comunicação com o

leitor, o que não ocorreu. Para suprir essa deficiência foi preciso documentar o protocolo antes do início dos trabalhos de integração do sistema RFID com a aplicação. Baseado nos estudos do programa original do fabricante, foi feita a seguinte descrição detalhada na Figura 10:

Comandos

```
Baud rate detection: 06 00 52 00 FC A8
CPU reset: 05 00 63 13 51
Software version: 05 00 65 25 34
RF reset: 05 00 69 49 FE
RF field ON: 06 00 6A 01 17 C1
RF field OFF: 06 00 6A 00 9E D0
Reset QUIET bit: 05 00 1B DC AE
Read: 07 00 B0 01 00 CE 93
```

Erros

```
Unknown Command: 06 00 1C 80 82 F0
No Transponder in Reader Field: 06 00 B0 01 5C 63
```

Resposta

```
Reading: 2F 00 B0 00 04 03 00 XX XX XX XX XX XX XX XX 03
00 XX XX XX XX XX XX XX 03 00 XX XX XX XX XX XX XX XX
03 00 XX XX XX XX XX XX XX XX
```

Obs: Onde estão as cadeias de XX é onde aparece o ID dos *tags*, e cada ID é separado por “03 00”, e o primeiro byte de cada comando ou resposta é o de bytes do pacote total. O segundo byte sempre é “00”.

Figura 10 - Descrição do protocolo serial do leitor ID.ISC.MR100

Para utilizar o leitor é necessário uma sequência de comandos em determinada ordem: (a) primeiramente tem que reiniciar o CPU; (b) depois inserir uma configuração para iniciar o sistema de controle de RF; (c) ativar o campo para preparar para a leitura; e finalmente (d) entrar num ciclo de leitura dos *tags* no campo. Se a ordem não for obedecida, o leitor não realiza as leituras corretamente. O fluxo de informações a ser seguido está apresentado em forma de diagrama na Figura 11.

Figura 11 - Fluxo de informações no DMS

Vendo isso em código serial:

1# 05 00 63 13 51	Reset (comando)
06 00 63 00 86 07	Reset OK(resposta)
2# 05 00 69 49 FE	Field Reset
06 00 69 00 F6 FA	Field Reset OK
3# 06 00 6A 01 17 C1	RF Field ON
06 00 6A 00 9E D0	RF Field ON OK

Neste momento é necessário fazer uma repetição constante do comando de leitura para realizar a varredura dos *tags* que estão no alcance da antena. Ou seja:

4# 07 00 B0 01 00 CE 93	Le tags
11 00 B0 00 01 03 00 XX XX XX XX XX XX XX XX	
5# 07 00 B0 01 00 CE 93	Le tags
25 00 B0 00 03 03 00 XX XX XX XX XX XX XX XX 03 00 XX	
XX XX XX XX XX XX XX 03 00 XX XX XX XX XX XX XX XX	

4.2.4. Confecção da antena

Outra característica deste projeto está relacionada ao fato de que a antena de RF não é um componente intrínseco ao leitor, sendo um acessório que necessita ser adquirido à parte. Então para reduzir os custos do sistema, uma antena foi projetada e confeccionada sobre uma superfície de madeira com uma fita de cobre. Ela foi calibrada usando dois resistores (um variável e o outro fixo) e um capacitor, que foram calculados a partir da documentação disponibilizada pela Texas Instrument com fórmulas descritas abaixo. O tamanho da fita especificado foi o tamanho necessário para cobrir toda a superfície da tábua com uma margem de 2 cm da borda, para evitar que possíveis danos à madeira pudesse afetar a fita de cobre.

$$f = \frac{1}{2\pi\sqrt{LC}} \quad [1]$$

Fórmula 1 – Cálculo da frequência de ressonância

$$L_{\mu H} = Side \times 0.008 \left[LN \left(\frac{Side \times 1.414}{2 \times Diameter} \right) + 0.379 \right] \quad [2]$$

Fórmula 2 – Cálculo da indutância

Com as dimensões necessárias para se montar a antena sobre a superfície de madeira, a indutância encontrada foi de 1,36 μ H.

$$C_{RES} = \frac{1}{\omega^2 L} \quad [3]$$

Fórmula 3 – Cálculo do Capacitor

Assim, o $C_{res} = 101,29$ pF

$$Q = \frac{R}{2\pi f L} \quad [4]$$

Fórmula 4 – Cálculo do Q

$$R_{PAR} = 2\pi f L Q \quad [5]$$

Fórmula 5 – Cálculo da resistência paralela

Com um $Q = 40$, o $R_{par} = 4635 \Omega$ e para o $Q = 120$, o $R_{par,antenna} = 13904 \Omega$

$$R = \frac{1}{\frac{1}{R_{par}} - \frac{1}{R_{par,antenna}}} \quad [6]$$

Fórmula 6 – Cálculo da resistência

Resultando em $R = 6952 \Omega$. Enfim, os valores dos componentes usados foram: um capacitor de 101pF e um resistor de 6K8 ligado a um resistor variável para fazer ajuste fino.

4.2.5. Leitor Korth KT-38

Com a colaboração da empresa Korth RFID, situada em São Carlos, foi concedido um leitor KT-38 utilizado normalmente para controle de acesso por cartão de proximidade. Devido a sua simplicidade de ligação e utilização, esse leitor foi muito útil para fazer os testes do sistema. O seu protocolo de comunicação com o computador é extremamente simples. Quando um *tag* é aproximado do leitor, o mesmo realiza a leitura automaticamente e envia pela porta serial o ID do *tag*. Assim não é necessário o envio de comandos de configuração, leitura e outros, pois basta simplesmente alimentar o leitor e realizar as leituras.

Desta forma, no método de `startReader()` estão somente os comandos de porta serial: abertura da porta, definição da velocidade e paridade e configuração do modo de leitura. Como esse leitor não utiliza nenhum protocolo de comunicação, não foi possível desenvolver o método `pingReader()`, fazendo com que sua implementação retorne “verdadeiro” sempre que ele for chamado. Já o método de `readData()` tem uma peculiaridade. Como as informações lidas pelo leitor chegam a qualquer momento, assincronamente e sem a necessidade de um comando de leitura, foi necessário criar uma fila PEPS, ou FIFO, das leituras realizadas pelo leitor. Essa fila é uma instância da classe `ReadDataList`, que é uma implementação da classe `ArrayList` do Java para objetos do tipo `RFIDReaderData`. Quando o leitor realiza uma leitura, as informações vão pela porta serial até o *driver* e lá, essa leitura é inserida na fila. Assim, sempre que o método de leitura `readData()` for chamado, será retirado da fila o objeto `RFIDReaderData` da leitura mais antiga existente e enviado ao DMS, ou retornará “nulo” caso não tenha sido realizada nenhuma leitura nesse período de tempo. Abaixo está parte do código de implementação usado do desenvolvimento do *driver* KT38.

```
public class KT38Reader extends RFIDReader{

    private ReadDataList readData;

    public KT38Reader() {
        readData = new ReadDataList();
        this.startReader();
    }

    public boolean pingReader() {
        return true;
    }
}
```

```

public RFIDReaderData readData() {
    if(readData.isEmpty())
        return null;
    else
        return readData.getReaderData();
}
}

```

4.2.6. Leitor ActiveWave ativo

O foco deste projeto é voltado para *tags* passivos, porém, devido às dificuldades tecnológicas, os leitores ideais para serem usados aqui (leitores 915 MHz com alcance de 10 metros ou mais) ainda são muito caros e de grande dificuldade de importação. Então, o que mais se aproxima de um leitor de 915MHz de longo alcance é um leitor de *tags* ativos, que possui alcances de até 100 metros e com um custo muito menor. Após uma longa negociação com uma empresa de Miami chamada ActiveWave e o seu representante nacional, foi possível adquirir 3 leitores de *tags* ativos e um conjunto de *tags* para os testes do sistema. Esse equipamento é conectado ao sistema por rede Ethernet, permitindo o uso simultâneo dos três leitores em rede e dispensando cabos seriais, e para permitir uma comunicação *full-duplex* o dispositivo opera em duas frequências: 433MHz para envio de informações ao *tag* e 915MHz para recebimento da resposta do *tag*.

O que diferencia esse equipamento dos outros utilizados anteriormente, no ponto de vista de integração com o DMS, é o seu protocolo de comunicação fechado e proprietário, sendo que toda a comunicação deve ser feita utilizando uma API disponibilizada pelo fabricante. Então todos os envios de comando e recebimentos de informações devem ser feitos através dessa API. O problema é que todo o sistema é feito em Java e a API é uma biblioteca (DLL) com funções de C++, nativamente incompatíveis entre si.

Para utilizar bibliotecas C++ em Java, é necessário fazer uma DLL Java com todas as interfaces da biblioteca em C++ utilizando uma outra biblioteca chamada JNI – *Java Native Interfaces* (MOISE, 2005). Assim, é possível fazer as chamadas das funções da biblioteca em C++ pelo Java. A comunicação com os leitores através da API se dá de maneira assíncrona, ou seja, uma função de envio de comando não retorna a resposta imediatamente, e quando a resposta chega, ela é tratada por uma função de *callback*. Os leitores podem ser configurados para ficar enviando comandos

de leitura pelo *software* e lendo os *tags* que estão no seu alcance (modo de funcionamento similar ao leitor da Feig) ou podem ser configurados para funcionar de maneira autônoma (modo de funcionamento similar do leitor da Korth). No funcionamento autônomo, o utilizado neste projeto, cada leitura de *tag* faz um chamado à função de *callback* com as informações do *tag* lido.

Para preparar a API e o leitor para o seu uso, é necessário executar a sequência de comandos descrita na Figura 12. Inicialmente deve registrar as funções de *callback* dos eventos dos *tags* (`rfTagEventRegister()`) e dos eventos dos leitores (`rfReaderEventRegister()`). Após as duas funções registradas, é feita uma varredura na rede através da função `rfScanNetwork()` para localizar os leitores ativos e retornar seus respectivos endereços IP. Com essas informações então é executada a função `rfOpenSocket()` para criar uma conexão TCP/IP entre o leitor e a API. Após estabelecida a conexão, a função `rfReaderReset()` é executada enviando um comando de *reset* para todos os leitores conectados. Se o leitor estiver funcionando corretamente, após o *reset* cada um dos leitores irá enviar um pacote de *PowerUp* contendo o seu ID e iniciará sua leitura continua autônoma por *tags* no seu raio de leitura, e para cada leitura realizada, a função de *callback* registrada inicialmente para os eventos dos *tags*.

Figura 12 - Sequência de comandos para uso da API e do leitor da ActiveWave

4.3. Core

4.3.1. Fluxo de comunicação interna

A cada leitura realizada pelo DMS, uma mensagem XML chega ao Core. Esta informação percorre o seguinte caminho descrito na Figura 13 para ser processada:

Figura 13 - Fluxo de Informações no Core

Após todo este fluxo de dados, as informações contidas no sistema ficam atualizadas e prontas para serem lidas pelo Command, onde ocorre a montagem do mapa e o posicionamento do *tag* sobre ele.

A parte de parametrização possui uma estrutura que será explicada mais detalhadamente na parte do subsistema de tratamento de evento, mas consiste basicamente na seguinte estrutura: todos os processos (check-up de saúde, por exemplo) são compostos por tarefas (ida ao hospital, atendimento médico, exame de sangue, exame de urina, etc) e cada tarefa possui uma série de atividades a serem desenvolvidas (entrada na sala de exame, bloqueio da veia, inserção da agulha e retirada do sangue). Então, cada processo possui uma ou mais tarefas, e cada tarefa pode conter uma ou mais atividades. Rotinas podem ser agregadas às atividades, chamadas aqui de módulos de execução, permitindo assim o envio de alguma mensagem pelo sistema de integração (LSI) ou um tratamento específico para aquele evento.

Quando um *tag* é lido pela 1ª vez, um módulo de inicialização, anexado à 1ª atividade do processo, gera um *thread* único de tratamento deste *tag* que só se encerra após a conclusão do processo, e na base de dados é gerada uma fila das atividades a serem realizadas. Se alguma dessas atividades não for realizada ou for realizada fora da ordem, então um alarme é enviado ao ALMS. A partir daí, o *tag* pode voltar a se movimentar pela área somente após o operador do sistema fazer uma intervenção pelo Command e finalizar o alarme.

4.3.2. Estrutura Webservice

Como citado inicialmente, o *webservice* é um *software* desenhado para suportar interoperabilidade e interação máquina-a-máquina sobre uma rede. Para implementar esse recurso no sistema, optou-se pelo uso da *web* como meio de integração dos diversos módulos. O recurso utilizado na estruturação do sistema é o de *webservice*, utilizando *servlets*. *Servlets* são APIs do Java que geram endereços *web* feitos para receber informações pelos métodos POST e GET do HTTP, normalmente usados para a comunicação entre aplicativos via *web* e executados sob um servidor de aplicação (MURRAY, 2006). No caso, aqui foi usado o conjunto Tomcat e Apache como servidor de aplicação. Isso faz com que toda a comunicação entre os módulos ocorra pela *web*, de maneira distribuída. Assim, pode-se dividir cada parte do programa em módulos, instalá-los em computadores diferentes e distribuir

esses computadores em diferentes redes. Se todos os computadores estiverem interligados através de um acesso comum (uma VPN, por exemplo), o sistema vai funcionar da mesma forma como se todo ele estivesse centralizado em um único servidor. A vantagem disso é que em aplicações críticas, o sistema se torna escalável, ou seja, caso o servidor esteja sobrecarregado, pode-se adicionar quantos servidores forem necessários para trabalharem juntos e dividirem a carga de trabalho, para assim conseguirem um desempenho melhor.

4.3.3. Banco de dados

O sistema incorpora os mais modernos recursos de programação Java. Um deles é o padrão DAO, que permite que o programador utilize os comandos típicos de banco de dados, sem a necessidade de escrever consultas em SQL (SULLIVAN, 2003). Com esse recurso, somado ao recurso de JDBC (uma API do Java que realiza a conectividade do aplicativo com os bancos de dados), o sistema passa não só a ser independente de banco de dados, como também se torna mais flexível e com mais abrangência de uso. Alterando um arquivo de propriedade como o descrito abaixo na Figura 14, o sistema pode utilizar desde os bancos de dados gratuitos, como o MySQL, até os bancos de dados comerciais mais comuns, como o Oracle. A base de dados usada no desenvolvimento deste trabalho foi a base Oracle XE, de licença gratuita para finalidades estudantis.

```

drivermysql=com.mysql.jdbc.Driver
urlmysql=jdbc:mysql://127.0.0.1:3306/mrp
usermysql=root
pwdmysql=

driverpostgresql=org.postgresql.Driver
urlpostgresql=jdbc:postgresql://127.0.0.1:5432/mrp
userpostgresql=postgres
pwdpostgresql=postgres

driveraccess=sun.jdbc.odbc.JdbcOdbcDriver
urlaccess=jdbc:odbc:Mrp
useraccess=sa
pwdaccess=sa

driversqlserver=com.microsoft.jdbc.sqlserver.SQLServerDriver
urlaccess=jdbc:microsoft:sqlserver://10.1.1.51:1433;databaseName=Mrp;selectMethod=cursor
usersqlserver=sa
pwdsqlserver=sa

```

Figura 14 - Arquivo de propriedades do JDBC com drivers MySQL, Postgres, ODBC e SQLServer

Estando a comunicação com a base de dados de forma bem controlada pelas camadas de gerenciamento, foi feita a estruturação das tabelas do Core e de todo o sistema de gerenciamento de eventos e parametrização. As tabelas foram divididas em grupos pelo tipo de informação que elas contêm. Todas as tabelas têm o seu nome inicial pelas letras “mrp”, que é a sigla do nome do projeto, para que elas possam ser facilmente reconhecidas no meio das outras tabelas da base de dados. O primeiro grupo de tabelas é o grupo de catalogação das informações essenciais de baixo nível para o funcionamento do sistema, visto na Figura 15.

mrp_Tasks		mrp_Activities		mrp_Modules		mrp_XMLMessages	
PK	<u>TaskCd</u>	PK	<u>ActivityCd</u>	PK	<u>ModuleCd</u>	PK	<u>XMLMessageCd</u>
	TaskNm LastChangeDt Authorizer		ActivityNm LastChangeDt Authorizer		ModuleNm LastChangeDt Authorizer		XMLMessageNm XMLMessage LastChangeDt Authorizer

Figura 15 - Tabelas de informações de baixo nível

A tabela mrp_XMLMessages tem como conteúdo a relação de todas as mensagens XMLs do sistema, tanto as usadas internamente pelo DMS e pelo Core,

como as utilizadas pelo LSI para comunicação com outros sistemas. Da mesma forma, as tabelas `mrp_Tasks` e `mrp_Activities` contêm a coleção de tarefas e atividades cadastradas no sistema, que podem ser adicionadas a um processo. No subsistema de gerenciamento de eventos, pode-se parametrizar o sistema para executar uma rotina quando uma tarefa é cumprida. Essas rotinas foram chamadas de Módulos de Programa, e a tabela `mrp_Modules` possui a relação de todos os Módulos de Programa disponíveis para uso no sistema..

O grupo seguinte de tabelas, apresentado na Figura 16 são as tabelas do subsistema de parametrização, o qual será abordado detalhadamente adiante, juntamente com a definição de todas as regras de permissão de acesso, tempo de permanência, ordem dos eventos, etc. Na tabela `mrp_Processes` estão cadastrados os processos do sistema (por exemplo check-up médico, acompanhamento de processos judiciais, etc). Esses processos possuem uma série de tarefas com suas regras de seqüência, tempo de execução, etc. Essas informações estão na tabela `mrp_ProcTasks` e exemplos de tarefas, para um processo de acompanhamento de processos judiciais, seriam a criação do processo, encaminhamento para o advogado, julgamento do juiz e encerramento do processo. Agora, cada uma das tarefas contém uma ou mais atividades a serem executadas. No caso da tarefa de encaminhamento para o advogado, o processo tem que realizar as tarefas de desarmarquivar, sair da sala de arquivo, entrar na expedição e entregar para o advogado, e cada uma dessas atividades estão cadastradas na tabela `mrp_ProcActivities`, juntamente com os módulos a serem executados na conclusão de cada atividade, no campo `ActivityMd..`

Figura 16 - Tabelas de parametrização

Com a operação do sistema, o histórico de informações geradas deve ser armazenado para análise posterior, e na Figura 17 estão apresentadas as tabelas responsáveis por isso. Cada atividade cumprida pelo *tag* é armazenada na tabela *mrp_ActivitiesRecord*. As tarefas enfileiradas na fila de tarefas a serem executadas pelos *tags*, quando o ele é lido pela 1ª vez, ficam na tabela *mrp_TasksQueue*. Após a conclusão dessas tarefas, o registro é retirado desta tabela e armazenado na tabela de histórico, chamada *mrp_Journal*. A tabela *mrp_LoginLog* registra todos os acessos de usuários no sistema, mantendo assim o histórico de acesso para auditoria futura. E finalmente, a tabela *mrp_Actions* contém os registros dos tipos de intervenções possíveis de serem feitas manualmente pelo operador do Command, como fechar um alarme, direcionar um *tag* para outra tarefa, etc.

mrp_ActivitiesRecord	
PK	<u>OrganizationCd</u>
PK	<u>TagData</u>
PK	<u>ProcessCd</u>
PK	<u>ProcTaskCd</u>
PK	<u>ProcActivityCd</u>
PK	<u>ActivityDt</u>
	ModuleNm MsgGroupCd MsgCd ExternalNr ComplementURL InstallationCd UserCd

mrp_Journal	
PK	<u>OrganizationCd</u>
PK	<u>ProcessCd</u>
PK	<u>ProcTaskCd</u>
PK	<u>MsgGroupCd</u>
PK	<u>MsgCd</u>
	PlaceCd ExternalNr ObjectTp ObjectSN ActionCd TagData LastChangeDt Authorizer

mrp_TasksQueue	
PK	<u>OrganizationCd</u>
PK	<u>ProcessCd</u>
PK	<u>ProcTaskCd</u>
PK	<u>TagData</u>
PK	<u>ExternalNr</u>
PK	<u>SeqCd</u>
	ObjectTp ObjectSN ScheduledTime ActionCd LastChangeDt Authorizer

mrp_LoginLog	
PK	<u>LoginDt</u>
	UserCd LogoutDt

mrp_Actions	
PK	<u>ActionCd</u>
	ActionDesc StartDt EndDt UserCd Authorizer

Figura 17 - Tabelas de histórico do sistema

A seguir, na Figura 18, estão as tabelas de cadastro dos equipamentos utilizados no sistema. A tabela *mrp_Tags* contém todos os tags cadastrados no sistema. Na tabela *mrp_Devices* estão cadastrados todos os leitores de RFID utilizados, com a informação de a qual DMS ele pertence contida no campo *DMSCd*. A próxima tabela, a *mrp_Places*, contém as informações sobre os locais onde os leitores estão instalados. Essas informações são, por exemplo, uma recepção, a sala de

exames de raios-X, entrada principal do prédio, etc. Agora, juntando a informação do equipamento de RFID com a informação de um local físico no prédio, cria-se o que é chamado de Instalação. As Instalações estão cadastradas na tabela `mrp_Installation`, e a cada Instalação tem uma atividade vinculada. No caso de uma sala de raios-X, teriam as informações de um leitor de RFID passivo no `mrp_Devices`, as informações da sala no `mrp_Place`, bem como no `mrp_Installation` teria a chave primária dos registros do leitor de RFID, da sala de raio-x e teria a atividade exame. Tudo isto inseridos num único registro, informando que naquele local físico existe aquele leitor com aquele ID e acontece aquela atividade.

Figura 18 - Tabelas de instalação dos equipamentos

Outra tabela importante desse grupo é a tabela `mrp_Happenings`, que é o local onde todos os eventos de leitura de RFID são inseridos primeiramente. Então, a

cada leitura realizada por algum leitor, é feito um *insert* nesta tabela com as informações do *tag* e da instalação onde ele foi lido. Nesta tabela foi colocado um *trigger* que envia uma mensagem ao módulo de gerenciamento de eventos informando que um novo evento foi gravado na tabela.

O grupo de tabelas seguinte (Figura 19) são as tabelas de cadastro de informações menos vitais, que não são obrigatórias para o sistema operar. Entre essas informações estão as tabelas de usuários cadastrados, *mrp_Users*, com seus respectivos endereços cadastrados na tabela *mrp_Addresses* e suas informações da empresa a qual pertencem inseridas na tabela *mrp_Organizations*. As tabelas *mrp_Mensagens* e suas relacionadas existem simplesmente para desvincular o sistema de um idioma específico. Todas as mensagens do sistema são inseridas nessa tabela. Então, para cada idioma configurado no sistema existe a mesma frase em cada língua inserida nessa tabela, assim o sistema pode trabalhar tanto em inglês quanto em português, por exemplo, sem a necessidade de alterar código fonte.

Figura 19 - Tabelas de cadastro do sistema

Enfim, a ultima tabela do sistema, `mrp_SysControl`, vista na Figura 20, é a tabela onde são armazenadas as configurações do sistema, que são usadas pelo Core no gerenciamento do funcionamento correto de todos os módulos, e de configuração dos outros módulos. Por exemplo, o campo `SharedFolder` contem a informação de onde será gravado o XML de integração que será visto mais detalhadamente quando for explicado o módulo de integração com outros sistemas.

mrp_SysControl	
PK	<u>OrganizationCd</u>
	InstallationDt NextUpdateDt EndOfLicenseDt ServerUTC ShutdownCd LocaleCd DialectCd SharedFolder DMSURL LegacyURL LastChangeDt Authorizer

Figura 20 - Tabela de configuração do sistema

Com esse conjunto de 28 tabelas foi possível realizar todas as implementações dos módulos de acordo com o especificado. Pode ser visto que em quase todas as tabelas existe sempre dois campos `LastChangeDt` e `Authorizer`. O `LastChangeDt` registra a data da última alteração desse registro na tabela, e o `Authorizer` registra o nome do usuário que fez essa alteração ou inserção da informação. Esse é um dos diversos recursos de segurança colocados no sistema para garantir o correto funcionamento com certo grau de proteção.

4.4. *Parms*

O módulo de parametrização chamado de *Parameters Configuration Subsystem* ou simplesmente *Parms*, é o módulo que contém todos os parâmetros de zonas permitidas, dos momentos em que as mesmas são proibidas, bem como os momentos em que elas são acessíveis e os tempos de permanência permitidos. Alguns modelos foram estudados, porém todos apresentavam implementações complexas e tornavam a parametrização mais complexa ainda (HUTCHENS, 1997) (MASUDA, 2003). Assim, para conseguir agrupar essas características em uma forma simples e facilmente configurável, foi definida no sistema a parametrização da forma vista na Figura 21, que atende à maioria das aplicações para a qual o sistema foi desenvolvido. Para armazenar essa estrutura foi criada uma tabela na base de dados que contém os processos usados no sistema, a `mrp_Processes` vista na Figura 22.

Figura 21 - Diagrama de divisão de um processo de exame médico

A tabela possui uma chave primária com o código do processo (`ProcessCd`), e seus campos são o nome do processo (`ProcessNm`) e o status do projeto, que pode ser ativo ou inativo (`ProcessSt`).

Em cada processo é configurada a ordem das etapas a serem seguidas, as diferentes opções de caminho e as regras de cada etapa. Para facilitar a montagem do processo, foi definido que um processo é um conjunto de tarefas. Cada uma dessas tarefas está gravada na tabela `mrp_ProcTasks` (Figura 23). Cada tarefa possui um código como chave

mrp_Processes	
PK	<u>ProcessCd</u>
	ProcessNm ProcessSt LastChangeDt Authorizer

Figura 22 - Tabela dos processos

primária (`ProcTaskCd`) e o código do processo a qual pertence (`ProcessCd`). A tarefa possui também um nome (`ProcTaskNm`) e o seu número na sequência de

mrp_ProcTasks	
PK	<u>ProcessCd</u>
PK	<u>ProcTaskCd</u>
	ProcTaskNm SeqCd TimingLastTk AlertMsgLastCd TimingThisTk AlertMsgThisCd TimingNextTk AlertMsgNextCd LastChangeDt Authorizer

Figura 23 - Tabela das tarefas

tarefas (`SeqCd`). Às tarefas foi atribuída a parte de controle de tempo de execução ou permanência, e para realizar esse controle existem os campos `TimingLastTk`, `TimingThisTk` e `TimingNextTk`, que são os intervalos de tempo permitidos entre o fim da última tarefa e início da nova tarefa, bem como a duração desta última tarefa, seu fim e o início da seguinte, respectivamente. Para cada limite de tempo desses excedido, existe um parâmetro de qual mensagem de alerta será apresentada. Então para o tempo entre o fim da última tarefa e o início da mesma, o campo `AlertMsgLastCd` representa o código da

mensagem armazenada na tabela `mrp_Messages`, visto anteriormente na Figura 19,

assim como o `AlertMsgThisCd` representa a mensagem de alarme para o campo `TimingThisTk` e o `AlertMsgNextCd` para o campo `TimingNextTk`.

Dando continuidade à hierarquia do processo, foi definido também que cada tarefa é composta por uma ou mais atividades e que a cada atividade seria atribuído um módulo de execução. As atividades do sistema ficam gravadas na tabela `mrp_ProcActivities`, como visto na Figura 24. A cada atividade atribui-se o código do processo e da tarefa à qual ela pertence (`ProcessCd` e `ProcTaskCd` respectivamente), o seu código de identificação (`ProcActivityCd`) e o código de sequência de execução da atividade. Além desses valores, a atividade possui também um nome (`ProcActivityNm`), o módulo de execução que será executado e o XML (armazenados na tabela `mrp_Modules` e `mrp_XMLMessages` como visto na Figura 15) que poderá ser enviado pelo sistema de integração com outras aplicações.

mrp_ProcActivities	
PK	<u>ProcessCd</u>
PK	<u>ProcTaskCd</u>
PK	<u>ProcActivityCd</u>
PK	<u>SeqCd</u>
	ProcActivityNm ActivityMd ActivityXMLMsg LastChangeDt Authorizer

Figura 24 - Tabela de atividades

Então um processo foi definido como um conjunto de tarefas, e as tarefas como um conjunto de atividades, e os controles de tempo são feitos por tarefas e as atividades têm a propriedade de poder executar um módulo de execução. Agora, para fazer essa ligação entre o mapeamento do processo e as leituras dos tags do RFID, foram definidas as instalações, que são armazenadas numa tabela chamada `mrp_Installation` (Figura 25). As instalações fazem a conexão entre o leitor de

mrp_Installations	
PK	<u>InstallationCd</u>
	ActivityCd PlaceCd DeviceCd DeviceId PlaceDeviceURL EnableDisable LastChangeDt Authorizer

Figura 25 - Tabela de instalações

RFID, o local físico e ao mapeamento do processo. Uma instalação possui: um código de identificação (`InstallationCd`); o código da atividade que ocorre naquele local (`ActivityCd`); o código do local físico que a instalação representa; e o código do dispositivo (`PlaceCd` e `DeviceCd`, ambos pertencentes as tabelas `mrp_Places` e `mrp_Devices` da Figura 18). A instalação possui ainda a identificação única do equipamento instalado neste local; o URL da imagem que será apresentada no Command quando essa instalação for

visualizada e o status da instalação, ativa ou inativa (`EnableDisable`).

Então, recordando, na instalação fica cadastrada a atividade que ocorre num dado local, na atividade fica o módulo de execução que será executado e a tarefa à qual ela pertence, na tarefa ficam parametrizados os tempos e o código do processo à qual ela pertence. Desta forma então, os tempos são parametrizados pelas tarefas, as propriedades únicas de cada zona são executadas dentro dos módulos de execução, e se tem as parametrizações de zona automaticamente quando se define a ordem das atividades, sendo que cada zona possui uma atividade definida.

4.5. ECS

No sistema existe um módulo responsável pelo gerenciamento de eventos, chamado *Event Control Subsystem* – ECS. O ECS é o encarregado de receber os eventos gerados pelos equipamentos de aquisição e de tratá-los, de acordo com o parametrizado no Parms. Todos os dados gerados pelos dispositivos são gravados em uma tabela chamada `mrp_Happenings` (Figura 26). Esta tabela é onde são gravadas as informações recebidas pelo XML do DMS.

Para cada evento gerado é criado um registro com um código único (`HappeningCd`) e um *timestamp* (`HappeningDt`). Pelo ID do equipamento que gerou o XML é possível descobrir a instalação em que ele está cadastrado, e assim, o código da instalação também é armazenado nesta tabela (`InstallationCd`). Junto com essas informações é registrado também o código do *tag* (`TagCd`). O campo `TagData` foi colocado prevendo um uso futuro do DMS a *tags* com memória de gravação, e o

mrp_Happenings	
PK	<u>HappeningCd</u>
PK	<u>HappeningDt</u>
	InstallationCd
	TagCd
	TagData
	Content
	MsgGroupCd
	MsgCd
	UserCd

Figura 26 - Tabela de inserção dos eventos

campo `Content`, prevendo compatibilidade com recursos de mensagens vindo do leitor, como potência do sinal recebido para cálculo de distância entre o *tag* e a antena, para o caso de equipamentos de RFID. Já os campos `MsgGroupCd` e `MsgCd` existem para caso o DMS precise enviar alguma mensagem, dentre as mensagens existentes no `mrp_Messages` ao Core ou a algum módulo do sistema.

Quando um *tag* é lido pela 1ª vez no sistema, através de um recurso da base de dados chamado *Advanced Task Queue*, é gerada uma fila de tarefas com todas as tarefas a serem executadas pelo *tag*. Essas filas são gravadas em uma tabela chamada `mrp_TasksQueue` vista na Figura 27. Cada registro possui as informações da

organização à qual pertence (*OrganizationCd* da tabela *mrp_Organization*, Figura 19), o código do processo ao qual a tarefa pertence (*ProcessCd*), o código da tarefa (*ProcTaskCd*), o dado que deve estar gravado no *tag* nesta tarefa

mrp_TasksQueue	
PK	<u>OrganizationCd</u>
PK	<u>ProcessCd</u>
PK	<u>ProcTaskCd</u>
PK	<u>TagData</u>
PK	<u>ExternalNr</u>
PK	<u>SeqCd</u>
	ObjectTp
	ObjectSN
	ScheduledTime
	ActionCd
	LastChangeDt
	Authorizer

Figura 27 - Tabela da fila de tarefas

(*TagData*), o número externo do *tag*, *ExternalNr*, localizado na tabela *mrp_Tags*, e o número da sequência da tarefa *SeqCd*. O registro possui também as informações do objeto ao qual o *tag* está anexado, bem como o tipo do objeto (*ObjectTp*) e o seu número serial (*ObjectSN*). Possui também o tempo previsto de conclusão da tarefa (*ScheduledTime*), baseado nas informações do *Parms*, e o código da atividade, que pode ser utilizado para corrigir algum alarme gerado nesta tarefa (*ActionCd*). Ao mesmo tempo em que é gerada essa fila de tarefas, é gerado um *thread* no motor do ECS,

responsável por gerenciar todas as atividades realizadas pelo *tag*, e por gerar os alarmes, caso alguma atividade for realizada fora do parametrizado.

Assim, com a chegada do XML enviado pelo DMS, o *servlet* responsável em receber o XML o interpreta, consultando as tabelas de instalações (*mrp_Installations*) para descobrir à qual instalação aquele leitor pertence, bem como as tabelas de *tags* (*mrp_Tags*) para descobrir qual o código dos *tags* cadastrados e à qual deles pertence aquele *TagId*. Com essas duas informações, é inserido um registro na tabela *mrp_Happenings*.

A tabela *mrp_Happenings* possui um recurso específico de base de dados chamado de gatilho ou *trigger*. Esse gatilho está configurado para ser disparado sempre que um registro é armazenado nessa tabela *mrp_Happenings*. Esse gatilho, quando disparado, aciona o *thread* responsável pelo *tag* no motor do ECS, para fazer o tratamento das informações gravadas no *mrp_Happenings*. Esse tratamento consiste em consultar a tabela das instalações para verificar qual a atividade relacionada àquela instalação. Sabendo a atividade, então o motor consulta as tabelas do *Parms* para ver se a atividade a ser iniciada possui um código de sequência (*SeqCd*) maior que a última atividade realizada pelo *tag*. Sendo maior, o motor do ECS executa o módulo de execução atribuído àquela atividade, caso exista. Depois de concluída a atividade, as suas informações são salvas na tabela

mrp_ActivitiesRecord, vista na Figura 28. São salvas as informações do código da organização, do código do processo e da tarefa à qual a atividade pertence como também o seu código próprio (OrganizationCd, ProcessCd, ProcTaskCd e ProcActivityCd respectivamente), assim como as informações gravadas na memória do *tag* e o *timestamp* de execução da atividade. Junto a essas informações é salvo também o nome do módulo de execução chamado pela atividade (ModuleNm), as mensagens geradas (MsgGroupCd e MsgCd), o número do *tag* (ExternalNr), um URL de uma imagem de registro da atividade para ser visto na

mrp_ActivitiesRecord	
PK	<u>OrganizationCd</u>
PK	<u>TagData</u>
PK	<u>ProcessCd</u>
PK	<u>ProcTaskCd</u>
PK	<u>ProcActivityCd</u>
PK	<u>ActivityDt</u>
	ModuleNm
	MsgGroupCd
	MsgCd
	ExternalNr
	ComplementURL
	InstallationCd
	UserCd

Figura 28 - Registro histórico de atividades

interface gráfica (ComplementURL), o código da instalação onde ocorreu a atividade (InstallationCd) e o código do usuário que registrou a atividade (UserCd). Essas informações das atividades realizadas são salvas nessa tabela para que se possa fazer uma auditoria futura dos eventos ocorridos no sistema. Depois disso, o *thread* consulta a tabela mrp_TasksQueue para analisar se a tarefa em execução foi concluída, e caso tenha sido, o motor do ECS retira o registro da fila de tarefas e atualiza as informações do histórico de registro na tabela mrp_Journal (Figura 29) juntamente com as

informações da tarefa realizada. Nesta tabela de histórico são salvas todas as informações contidas no registro do mrp_TaskQueue, junto com a informação do local onde foi realizada a tarefa (PlaceCd), já que pela estrutura do sistema, um mesmo tipo de tarefa pode ser realizada em diferentes locais. Agora, caso algum dos parâmetros seja divergente com o parametrizado pelo Parms, então um alarme é gerado, e a mensagem parametrizada no Parms na tabela de mapeamento das tarefas (Figura 23) é enviada ao módulo de alarmes a ser visto mais a frente. O alarme pode ser gerado caso a atividade relativa ao evento gerado pelo leitor de RFID não seja a próxima atividade esperada na fila de tarefas, ou caso o tempo de execução da tarefa seja

mrp_Journal	
PK	<u>OrganizationCd</u>
PK	<u>ProcessCd</u>
PK	<u>ProcTaskCd</u>
PK	<u>MsgGroupCd</u>
PK	<u>MsgCd</u>
	PlaceCd
	ExternalNr
	ObjectTp
	ObjectSN
	ActionCd
	TagData
	LastChangeDt
	Authorizer

Figura 29 - Tabela de registro histórico

maior que o tempo parametrizado na tabela das tarefas do processo. Essa mensagem enviada ao ALMS é gravada no histórico (MsgGroupCd e MsgCd), e caso não seja gerado nenhum alarme, então a mensagem de execução correta é colocada.

4.6. ALMS

No que se diz respeito à parte de alarmes, existe um módulo dedicado somente à essa função de gerenciamento deles, chamado de *Alert Management Subsystem* - ALMS. O ALMS recebe as mensagens de alarme do ECS quando algum evento veio fora da ordem programada ou se houver algum estouro de tempo entre os intervalos parametrizados no Parms. O ALMS, por si, só trabalha com alarmes visuais e sonoros na tela do operador. Porém quando aliado ao módulo de integração com outras aplicações, o sistema de alarmes pode gerar alertas, tais como, chamadas telefônicas, acionamento de travas de porta, acionamento de algum motor elétrico, etc.

A interface gráfica do sistema, a ser vista detalhadamente mais a frente, possui três telas de visualização: a tela de cadastro e configuração, onde são cadastrados os tags, locais, alterados parâmetros do PARMS, etc; a tela de status do sistema, onde é apresentado o mapa, normalmente uma planta baixa, com as posições dos objetos portando os tags e a sua respectiva localização; e a tela de controle, ou centro de controle operacional – CCO (Figura 30), onde são gerados os alarmes, onde pode ser realizado o fechamento das exceções e onde se pode realizar mudança na ordem das tarefas programadas para um dado *tag*. Nessa tela de controle, ou CCO, é onde o ALMS lança seus alarmes na forma de um *pop-up* na tela. Cada *tag* que gerar uma exceção gera um *pop-up* na tela, conforme visto na Figura 31, e só poderá continuar sua movimentação normal caso o alarme seja fechado pelo operador.

Figura 30 - Tela de controle

No ponto de vista do sistema, o ECS, quando recebe um evento de algum dispositivo, consulta a tabela `mrp_TasksQueue` para ver qual é a próxima atividade esperada para aquele *tag*. Caso a atividade executada pelo evento recebido do equipamento não seja a atividade programada, então o motor do ECS envia uma mensagem de quebra de fluxo do processo para o ALMS, e assim, é enviado um *popup* para a interface gráfica.

Tarefas normais

Tarefas de exceções

Tarefas de alerta

Tarefas de alarme

Ver ações em aberto

Fecha exceção

Fecha alarme/alerta

Mudança de password

Tipo de alerta: Todos

Número de horas: 08

Ir

Código da mensagem	Descrição da mensagem	Descrição da mensagem	Código do processo	Código da tarefa	Número externo	Data da última modificação
0002	Quebra de sequência da operação.		Processo		200	2006-08-04 12:09:18.0

Action Msg#555

Ext. #:

200

Id:

1216

Tag Data:

NA2006080412091864783917341

Task Cd:

Quebra de sequência da operação.

null

Action code:

555

Description:

Save

Figura 31 - Tela de controle com o Pop-Up

Neste *pop-up* estão apresentadas as informações do *tag* que gerou o alarme, o tipo de alarme gerado, e outros campos previstos para uso futuro. O *pop-up* se mantém até que seja tomada uma ação de fechar o alarme através de uma senha de administrador, fazendo com que esse alarme deixe de estar ativo e passe a fazer parte do histórico de alarmes.

4.7. LSI

O módulo de integração com outras aplicações, chamado de *Legacy Systems Integration Subsystem* – LSI permite que o sistema de monitoramento se interligue com outras aplicações. Pela definição, o LSI deve permitir diversos tipos de integração, assim como o DMS permite diversos dispositivos diferentes simultaneamente. Então, as estruturas do LSI e do DMS são similares, fazendo com que o LSI tenha uma classe pai abstrata de todos os *drivers* de interligação chamada de *LSIDriver*, como pode ser visto abaixo, só que mais simples que a classe do DMS:

```
public abstract class LSIDriver {

    public boolean waitAnswer;
```

```

    public abstract int sendMessage(String message);
    public abstract String getMessage();
}

```

Nessa estrutura, são necessários somente dois métodos para implementar um *driver*, o método `sendMessage()` para envio de mensagens, e o `getMessage()` para recebimento. A classe `LSIDriver` possui também um atributo booleano `waitAnswer`, que indica se o LSI deve esperar uma resposta e executar o método `getMessage()` ou não.

Para ativar a integração, deve-se configurar o parâmetro LSI no Parms com o nome dos *drivers* de interligação ativos. Na definição das zonas, dos alarmes e dos eventos, deve-se também colocar o campo de interligação e o nome da mensagem, para que naquele ponto seja enviado um pacote de informações para o LSI. O *driver* do LSI recebe a mensagem e executa o método `sendMessage()`, colocando a mensagem como parâmetro. Caso seja configurado para aguardar uma resposta, então a função `getMessage()` também será executada e a resposta será enviada de volta para o core.

4.7.1. Integração por envio de arquivos em uma pasta compartilhada

Como a maioria dos aplicativos comerciais de controle de acesso é escrito em linguagens estruturadas como C ou Pascal, então a forma de integração possível para a maioria dos aplicativos mais antigos é a gravação de arquivos-mensagem em uma pasta compartilhada pelas duas aplicações. Para isso, a implementação do método `sendMessage()` cria um arquivo com nome “Oyyyymmddhhmmss.xml” (a letra O é de *output*, yyyy é ano, mm é mês, dd é dia, hh é hora, mm é minuto e ss é segundo do *timestamp* de criação do arquivo) na pasta configurada no Parms, e seu conteúdo possui o seguinte XML:

```

<?xml version="1.0" encoding="UTF-8"?>
<LSIOutputMessage Id="433">
    <Message> Mensagem </Message>
</LSIOutputMessage>

```

O atributo `Id` do *tag* `LSIMessage` é simplesmente um índice incremental sequencial e que deve ser repetido na mensagem de resposta caso exista.

E a resposta aguardada deve ter a nomenclatura “Iyyyymmddhhmmss.xml” (a letra I é de *input* e as outras letras têm o mesmo significado do pacote de envio) criado no mesmo diretório com o seguinte formato:

```
<?xml version="1.0" encoding="UTF-8"?>
<LSIInputMessage Id="433">
    <Message> Mensagem </Message>
</LSIInputMessage>
```

Assim, o método `getMessage()` fica realizando uma busca no diretório até detectar um arquivo com o nome iniciado por “I” e que tenha um atributo Id igual ao atributo Id do arquivo enviado.

4.8. Command

A interface *web* do sistema passou por diversas alterações, porém graças à estrutura do sistema, essas alterações não interferiram no seu funcionamento. As mensagens existentes nas telas são extraídas dinamicamente da tabela `mrp_Messages`, de forma que diferentes idiomas podem ser cadastrados nesta tabela para que automaticamente a interface gráfica passe a utilizar esse novo idioma. A interface gráfica do sistema é dividida em 3 telas: As telas de consulta e cadastro, onde são atualizados os parâmetros do sistema; A tela da planta, onde é apresentada a posição do *tag* no mapa; e a Tela do CCO, onde são mostrados os alarmes, apresentada anteriormente na parte de alarmes.

4.8.1. Tela de Consulta

Na Figura 32 pode ser vista uma das telas de consulta das tabelas do Parmis. Como pode ser visto, a consulta pode ser feita baseada em qualquer um dos campos existentes na tabela. Aqui está apresentada a consulta à tabela `mrp_Happenings`, sendo que as opções de consulta são por tipo de evento, data, conteúdo, código da instalação, e muitos outros, sendo que esta característica se repete para todas as tabelas do sistema. Desta forma, é possível fazer qualquer consulta às tabelas sem precisar escrever *queries*, e nem precisar de acesso à base de dados.

Usuário: lgalhego
Tempo de login: 8:42:56

ORG. TAGS INSTALAÇÕES PROCESSOS SYSCONTROL DEVICES LOCAIS USUARIOS ENDERÇOS CONSULTAS

LOG OFF

Consulta de happenings

Código do evento: DMS0001 Data: equals Contêúdo: equals Código da instalação:

Código do Tag: Dados do Tag: equals

Código do grupo de mensagens: código de mensagem: Código do usuário: Ir

Figura 32 - Tela de consulta

4.8.2. Tela de cadastro

A tela mostrada na Figura 33 é onde são cadastradas as unidades que serão lidas pelo sistema. Como especificado e, devido ao uso do protocolo XML, o sistema é independente de tecnologia. Assim, pode-se escolher a melhor tecnologia para cada projeto e ainda utilizar diversos tipos de dispositivos ao mesmo tempo.

Usuário: lgalhego
Tempo de login: 8:42:56

ORG. TAGS INSTALAÇÕES PROCESSOS SYSCONTROL DEVICES LOCAIS USUARIOS ENDERÇOS CONSULTAS

LOG OFF

Manutenção de Tags

Código do Tag: 1216 Ir

Nome: equals Id do Tag: equals Ir

Código	Id do Tag	Nome	Tecnologia	URL do Tag	Bar code	Número externo	Número interno	Data de inclusão	Data de saída	Razão	Data da última modificação	Autorizador	Manutenção
1216	216	teste216	RFID		1216	1216	1216	0006-05-10 00:00:00.0			2006-05-10 14:44:15.0	lgalhego

Incluir novo Tag

Figura 33 - Tela de cadastro de unidades

4.8.3. Tela do mapa

Conforme mostrado na Figura 34, pode ser visto o mapa do local controlado cadastrado e a localização das leitoras sobre a planta baixa da área. No exemplo mostrado, um tag previamente cadastrado no sistema, e que é referenciado por um ícone de uma carteira, foi lido pela antena que está localizada no estacionamento do local.

Figura 34 - Tela do mapa

CAPÍTULO 5 - APLICAÇÕES PRÁTICAS DO MIDDLEWARE

Para analisar o real desempenho e uso do sistema, foi focado o desenvolvimento de um caso de aplicação prática. Com o apoio de uma empresa de transporte rodoviário de cargas a granel que já desenvolvia um projeto no laboratório, todo o sistema foi parametrizado para atender um fluxo de descarga de caminhões com cargas a granel. Assim, colocando *tags* nos caminhões, leitores de RFID nos locais de pesagem e descarga, bem como equipando os fiscais de descarga com PDAs ligados em rede via Wifi, se torna possível controlar o fluxo de descarga seguido pelos caminhões e os tempos entre cada etapa, assim como visualizar alarmes em caso de quebra de fluxo ou atrasos (FOINA, 2006).

5.1. *Processo de descarga de cargas a granel*

O processo de descarga de cargas a granel possui três etapas, conforme Figura 35.

Figura 35 - Fluxo de descarga de carga a granel

De acordo com o fluxo então, primeiramente o caminhão realiza uma pesagem de seu peso bruto (peso do caminhão mais peso da carga) numa balança rodoviária, que é basicamente uma plataforma de pesagem maior que o caminhão, que pesa o cavalo e a carroceria usando células de carga. Após a pesagem bruta, o caminhão se desloca aos locais apropriados de descarga. Caso o caminhão seja do tipo caçamba (Figura 36a), com sistema de caçamba basculante, então o caminhão se dirige a um armazém. Caso o caminhão seja um caminhão graneleiro (Figura 36b), então ele se desloca a um local chamado moega, que possui uma plataforma basculante que levanta o caminhão e escorre sua carga. Um fiscal de descarga em cada um dos locais

portando o PDA realiza a verificação da placa, nota fiscal, tipo de carga, etc do caminhão. Após a descarga, esse fiscal informa no PDA se a descarga foi total ou se a descarga foi incompleta, ou parcial, redirecionando o caminhão para outro local de descarga. Neste mesmo PDA, caso o caminhão que esteja entrando tenha realizado alguma quebra de fluxo, uma mensagem de alarme é apresentada ao fiscal.

Figura 36 - (a) Caminhão caçamba basculando; (b) Caminhão graneleiro na moega

Após a descarga da carga nos locais, o caminhão se dirige novamente para a balança para fazer a sua pesagem tara, que consiste simplesmente no peso do caminhão vazio. Então, tendo o peso bruto (peso do caminhão mais peso da carga) e o peso tara (peso do caminhão), a diferença entre eles resulta no peso da carga descarregada.

$$P_{bruto} - P_{tara} = (P_{caminhão} + P_{carga}) - P_{caminhão} = P_{carga}$$

Fórmula 7 - Cálculo do peso da carga

Assim, o peso da carga é comparado com o descrito na nota fiscal do frete e validado. Existem algumas situações previstas no fluxo normal, porém ocorrem com menor frequência, como a descarga parcial e a raspagem.

A descarga parcial ocorre quando por algum motivo, o caminhão não conseguiu realizar a descarga total da carga no local designado, ou por defeito do caminhão, por defeito da moega ou por algum outro motivo. Assim, o caminhão deve

sair do local de descarga e dirigir-se para outro local de descarga (outra moega ou armazém), reiniciando o processo de descarga.

Já a raspagem ocorre quando o cálculo do peso da carga resulta em um valor menor do que o descrito na nota fiscal, ou seja, o peso descarregado medido está menor que o peso esperado, gerando o que chamado de “quebra”. Esse evento normalmente ocorre em dias úmidos, quando o açúcar fica grudado nas paredes da carroceria, gerando uma medição do peso tara maior do que o peso do caminhão. Pode ser causado também quando o caminhão é muito antigo e possui buracos na carroceria, deixando vazar o produto durante a viagem até o porto. Assim, se o motorista desejar, o caminhão se desloca novamente para o local de descarga e realiza novamente o processo de descarga, caso contrário, a diferença de peso entre o medido e o descrito na nota é descontada do pagamento da transportadora.

Então, o fluxo de descarga completa fica da seguinte forma visto na Figura 37.

Figura 37 - Fluxo completo de descarga

5.2. Implementação do fluxo no sistema

Baseado no fluxo apresentado foi gerada uma configuração do Parms com um processo de descarga visto na Figura 38. Esse processo consiste em três tarefas: pesagem bruto; descarga; e pesagem tara.

Figura 38 - Processo cadastrado no Parms

Cada uma dessas tarefas possui três atividades. A tarefa de pesagem bruta (Figura 39) possui como atividades a entrada na balança, captura do peso bruto e a saída da balança.

Figura 39 - Tarefa de pesagem bruta

Já a descarga (Figura 40) possui a entrada no local de descarga, a descarga em si (confirmação do PDA) e saída do local.

Figura 40 - Tarefa de descarga

E finalmente, a tarefa pesagem tara (Figura 41) possui as atividades de entrada na balança, captura peso tara e saída da balança.

Figura 41 - Tarefa de pesagem tara

As atividades de entrada e saída serão eventos gerados pelos leitores de RFID e pelos *tags* fixados nos caminhões. Os eventos de captura de peso serão gerados pelo LSI através da integração com o sistema legado da empresa, assim como as informações de necessidade de raspagem. As informações de descarga final e parcial são fornecidas pelos fiscais de descarga através do PDA.

Durante todo o processo de descarga, o caminhão pode assumir sete estados (Figura 42): aguardando na fila da pesagem bruta, realizando pesagem bruta, aguardando na fila para descarga, descarregando, aguardando na fila para pesagem tara, realizando a pesagem tara e finalizando o fluxo.

O estado do caminhão está diretamente relacionado com qual tarefa ele está realizando. As tarefas cadastradas são estados, e o intervalo entre o fim de uma tarefa e o início da tarefa seguinte são outros estados. Por exemplo, se o caminhão já realizou a tarefa de pesagem bruta, porém ainda não iniciou a tarefa de descarga, então ele se encontra no estado de fila para descarga.

Figura 42 – Diagrama de estados do processo

A informação sobre qual estado o caminhão se encontra e sobre quantos caminhões estão em cada estado é vital para o departamento de logística, de forma que o chefe da operação possa saber quantos caminhões faltam para descarregar, pois o tamanho das filas é baseado nessas informações, bem como a decisão de fazer mais ou menos pedido de caminhões para descarregar. Assim, o Command foi desenhado para, em vez de apresentar o mapa com a localização do caminhão, apresentar a contagem do número de caminhões em cada estado, permitindo a visualização dos caminhões que se encontram em cada etapa do processo (Figura 43). Selecionado algum dos estados, o Command apresenta todos os veículos que estão naquele estado, juntamente com informações extras como placa, carga, tipo e o tempo em que o caminhão se encontra neste estado. Na Figura 44 pode ser visto o exemplo da relação de caminhões que se encontram na fila de pesagem tara naquele instante.

Figura 43 - Tela de acompanhamento dos estados do processo

Figura 44 - Apresentação dos caminhões que estão no estado fila pesagem da tara

5.3. Tecnologias Utilizadas

Neste caso, devido às distâncias envolvidas e à hostilidade do meio onde os tags de RFID estavam sujeitos, foram utilizados leitores de RFID ativos da ActiveWave, descritos mais detalhadamente no Anexo A. Essa tecnologia de tag possui a capacidade de ligação de um sensor de abertura, permitindo o envio do status do sensor junto com os pacotes de identificação do tag. Assim, foi desenvolvido um encapsulamento especial para o tag, em formato de cadeado e com uma chave de abertura que permite o tag saber se o cadeado está aberto ou está fechado. Caso esse

cadeado seja aberto durante o processo de descarga, um alarme de violação do lacre eletrônico é mostrado na tela do CCO (Figura 45).

Figura 45 - Alarme de violação de lacre eletrônico

Devido aos problemas gerados pelo metal nas transmissões de RF, o cadeado teve metade se sua estrutura feita de plástico e outra metade metálica (DOBKIN, 2005). O corpo, que contém a eletrônica do tag, é a parte plástica, deixando o sinal de rádio entrar e sair do cadeado praticamente sem atenuação. A parte que se usa para fixar o cadeado foi feita de metal para prover a resistência necessária e agüentar os impactos com o caminhão. Esse cadeado foi dimensionado para se encaixar às barras laterais dos caminhões, como pode ser visto na Figura 46.

Figura 46 - Tag cadeado

Os leitores foram instalados nas entradas e saídas das balanças, moegas e armazéns, de forma a realizar leituras dos *tags* quando os caminhões entram e saem dos locais. Na Figura 47 pode ser visto o local de instalação dos leitores na balança de pesagem.

Justamente com os equipamentos de RFID, foram usados PDAs para enviar informações do sistema para o fiscal juntamente com dados do caminhão fornecidos pelo sistema legado da empresa, como placa e carga por exemplo, e para enviar informações do fiscal para o sistema informando se a descarga foi final ou parcial. As informações são enviadas do PDA para o sistema através de uma mensagem XML encapsulada no protocolo HTTP para o *driver* do PDA no DMS, que possui um *webserver* simplificado para realizar essa transação. E para o PDA receber informações do sistema, outra mensagem XML também encapsulada em HTTP é enviada através do LSI para o aplicativo do PDA. As mensagens são descritas detalhadamente no Anexo B.

Figura 47 - Local de instalação dos leitores na balança

5.4. Resultados

Os resultados obtidos neste estudo de caso referem-se à área de segurança e à área de logística.

De acordo com a análise do número de alarmes medidos em 2 meses de operação do sistema, como visto na Figura 48, 102 alarmes operacionais foram gerados por defeitos na moega, quebra de caminhões ou manobras erradas, e 16 alarmes dos 118 alarmes totais gerados ocorreram em situações de possível furto de carga. Sabendo que cada caminhão possui uma carga no valor de R\$ 9 mil reais em produto, os alarmes de segurança preveniram então o furto de R\$ 144 mil reais em cargas furtadas. Fazendo a extrapolação deste cálculo para um ano, temos um total de R\$ 864 mil reais de produto não furtado. Então, com um ano completo de operação do sistema e sabendo que o total de perda de produto na empresa analisada é de R\$ 1,5 milhões de reais, chega-se à redução aproximada de 60% do total de carga perdida (Figura 49).

Gráfico dos Alarmes Gerados

Figura 48 - Gráfico dos alarmes gerados

Volume desviado

Figura 49 - Gráfico da economia gerada pelo sistema

No ponto de vista de logística, o departamento logístico apresentou que a quantidade de caminhões que devem estar em processo de descarga simultâneo, para evitar ociosidade dos locais de pesagem e descarga, deve ser de 40 caminhões. Com as informações de tamanho da fila e quantidade de caminhões em cada etapa, o departamento de operação conseguiu gerenciar as filas de forma a manter esse volume de veículos a maior parte do tempo. As informações de tempo que cada caminhão gastou em cada etapa permitirão a esse departamento encontrar o ponto de gargalo e

definir metas de redução de tempo em cada etapa. Do ponto de vista de desempenho, o sistema deve ser capaz de gerenciar mais de 40 processos de descarga simultâneos. Um gráfico de desempenho foi gerado (Figura 50) como resultado dos testes de stress, mostrando que o sistema suporta a descarga de até 50 caminhões sem gerar atrasos nos pacotes tratados.

Figura 50 - Gráfico de desempenho do sistema

Testes de sobrecarga foram feitos e o sistema ainda se mostrou eficiente gerenciando 60 processos simultâneos de descarga, como visto na Figura 51. Apesar dos picos de 100% no consumo do processamento, ainda existem algumas quedas curtas do processamento, mostrando que o sistema teve breves momentos de ociosidade. O equipamento utilizado foi um servidor Pentium Xeon 3.67GHz, com 512MB de memória e o Windows Server 2003 Service Pack 1.

Figura 51 - Imagens das variáveis do sistema operando com 60 caminhões simultâneos

CAPÍTULO 6 - CONCLUSÕES E PLANOS FUTUROS

No decorrer de todo esse trabalho, foram apresentadas as tecnologias envolvidas nas mais recentes tendências do desenvolvimento de sistemas. A estrutura proposta explora as características de modularidade e implementa uma estrutura distribuída, assim como um *middleware* para gerenciar dispositivos e uma camada de análise de regras de negócio. Desta forma, o sistema possui grande flexibilidade, escalabilidade e disponibilidade.

Com todos os dados apresentados e os resultados do estudo de caso prático do sistema, o sistema pode ser de grande importância tanto para análises logísticas quanto para segurança. Sua estrutura também permite rápidas alterações de contexto sem a necessidade de se reescrever código, somente alterando os parâmetros do processo controlado e, se necessário, desenvolvendo alguns *drivers* para o DMS e para o LSI. Essas características mostram que este sistema é uma importante ferramenta de análise de rede de sensores e para controle de processos.

Testes do mesmo sistema com outros processos de diferentes características, como acompanhamento de processos judiciais, análise de dados gerados por sensores inteligentes *wireless* de controle de solo, e gerenciamento de frotas de veículos, permitirão a validação da estrutura do sistema e sua flexibilidade.

6.1. Planos Futuros

O sistema já se mostrou operacional e estável, porém mesmo com todos os recursos e técnicas envolvidos, ele ainda possui algumas limitações. Entre eles está o caminho percorrido pela informação na sua estrutura. O dado é gerado na camada de DMS, passa pelo Núcleo e vai para a parte de interligação com aplicações. Porém, o caminho contrário, apesar de suportado pela estrutura, não foi implementado, impedindo que o LSI possa gerar um evento e o DMS receber alguma ordem vinda do Núcleo. Por esse motivo, a implementação da comunicação com o PDA no estudo de caso exigiu um *driver* no DMS e um no LSI.

No DMS foram implementados somente *drivers* para rede de sensores baseadas em leitores de RFID. Como plano futuro tem-se o desenvolvimento de

drivers para aplicações em computação pervasiva e rede de sensores inteligentes, devido a sua crescente importância e material gerado nestas linhas de pesquisa.

Outro recurso que está no *roadmap* do sistema é o desenvolvimento de uma ferramenta visual de diagrama de blocos para realizar o mapeamento do processo e carregar, de forma automática, esse processo nas tabelas do PARMS. Assim, o sistema pode ser configurado para os diferentes processos sem a necessidade de algum especialista em base de dados.

Finalmente, outro objetivo futuro seria portar a camada de *middleware*, o DMS, para uma arquitetura de computador diferente, com características *multi-thread* para possibilitar o gerenciamento de centenas de dispositivos ao mesmo tempo com um desempenho muito maior.

ANEXO A - ESPECIFICAÇÃO DOS EQUIPAMENTOS UTILIZADOS

advanced reader technologies

i-scan

Mid range reader
ID ISC.MR100-A/
-USB

Figura 52 - Fotos do leitor ID.ISC.MR100 e da antena montada

Technical data

Housing	Plastic
Colour	RAL 9018 (light grey)
Dimensions (LxWxH)	145 x 85 x 31 mm
Protection class	IP 30
Weight	170 g
Power supply	
- variant -A (RS232/RS485)	12-24 V DC +/- 15% via external power supply
- variant -USB	12 V via external power supply
Power consumption	approx. 6 VA
Operating frequency	13,56 MHz
Transmitting power	approx. 0,8 up to 1 W
Modulation factor	10% and 100% (via software adjustable)
Antenna connection	1 x SMA-female plug (50 Ω)
Reading distance	max. 40cm with ID ISC.ANT300/300
Interfaces	RS232 / RS485 (switchable) or USB
Signal generator	1 LED (multicoloured; red / green)
Processable transponders	I-CODE, Tag-it and ISO 15693
Temperature range	
- operation	-25° C up to 60° C
- storage	-25° C up to 85° C
FLASH	64 kByte (software may be updated via interface)

Figura 53 - Especificação do leitor ID.ISC.MR100

	Descrição	Min.	Méd.	Máx.	Un.
Vcc	Tensão de alimentação	12		18	V
Icc	Corrente em stand-by (Vcc=12V)		35		mA
IccL	Corrente em leitura (Vcc=12V)		80		mA

Ligações Gerais (Placa)

Pino	Descrição	Operador	Pino	Descrição	Operador
NA	Relê Normal Aberto	SAÍDA	TX	RS232 - TX TTL	SCI
NF	Relê Nor. Fechado	SAÍDA	RX	RS232 - RX TTL	SCI
C	Relê Comum	SAÍDA	BP	Aciona BEEP	Saida
TXO	RS232 - TX	SCI	SS	Slave Select	Sel. Master/Slave
RDI	RS232 - RX	SCI	5V	5 Vcc	Polariza SS
CP	Wiegand D0	WG	DT	Wiegand D1	WG

Capacidade dos contatos

	Descrição	Min.	Méd.	Máx.	Un.
Vcca	Tensão entre contatos em corrente alternada			250	V
Vccc	Tensão entre contatos em corrente contínua			24	V
Icca	Corrente alternada entre contatos			7	A
Iccc	Corrente contínua entre contatos			5	A

Figura 54 - Fotos e especificação do leitor KT-38

Functionality	Reads and writes RFID tags	
Multi-Tag Read Capability	Yes	
Transmit Frequency to Tag	433 MHz	
Receive Frequency from Tag	916 MHz, 927 MHz, or 868 MHz	
Range	30m (100 feet) to tag	
	85m (280 feet) from tag	
Host Communications	RS232	9600 - 115200 Baud
	Ethernet	10/100 Mbps
	WLAN (optional)	2.4 GHz, 5.2 GHz
Power	12Vdc, 1.5A	
Dimensions	without antennas	150 mm x 85 mm x 27 mm (5.9 in x 3.3 in x 1.1 in)
	with antennas	150 mm x 85 mm x 167 mm (5.9 in x 3.3 in x 6.6 in)
Weight	680 grams (1.5 lbs)	
Case Material	Impact resistant polystyrene with UL94-HB flammability rating	
Temperature	Operating	-35C to +50C (-31F to +122F)
	Storage	-40C to +85C (-40F to +185F)
Indicators	RF LED	On while receiving packet from tag.
	HOST LED	On while sending validated tag packet to Host.
	ACCESS LED	On while transmitting packet to tag.
	POWER LED	On when Reader is powered.
Connectors	Power	12Vdc, 1.5A
	Ethernet	RJ-45 female to Host
	MotionDetector	RJ-11 male
	Host Comm.	Same RJ-11 male to Host (DB9 female to Host optional)
	Input	Two contact sense inputs
	Output	Two isolated dry contact relay outputs

Figura 55 - Fotos e especificação do leitor da ActiveWave

ANEXO B - MENSAGENS XML USADAS NO ESTUDO DE CASO

Abaixo são descritas as mensagens XML utilizadas pelo sistema, e seus respectivos DTDs.

B.1. DMSAuthenticationRequest

Mensagem de requisição de autenticação do DMS.

```
<!--ELEMENT DMSAuthenticationRequest (DMSDriver)>
<!--ELEMENT DMSDriver (Time)>
<!--ATTLIST UniqueId CDATA #REQUIRED>
<!--ELEMENT Time (#CDATA)>
```

Através do atributo UniqueId o sistema sabe qual é o DMS que está requisitando autenticação.

```
<?xml version="1.0" encoding="UTF-8"?>
<DMSAuthenticationRequest>
  <DMSDriver UniqueId="216">
    <Time>07/19/2006 20:03:12.796</Time>
  </DMSDriver>
</DMSAuthenticationRequest>
```

B.2. DMSAuthenticationResponse

Resposta à requisição de autenticação do DMS.

```
<!--ELEMENT DMSAuthenticationResponse (DMSDriver)>
<!--ELEMENT DMSDriver (Time, Validity)>
<!--ATTLIST UniqueId CDATA #REQUIRED>
<!--ELEMENT Time (#CDATA)>
<!--ELEMENT Validity (#CDATA)>
```

O XML de resposta vem com um campo novo, chamado Validity, que informa até quando essa instância do DMS está autorizada a ser executada. Após esse momento, uma nova requisição de autenticação deve ser feita.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<DMSAuthenticationResponse>
  <DMSDriver UniqueId="216">
    <Time>07/19/2006 20:03:13.902</Time>
    <Validity>07/19/2007 20:03:13.902</Validity>
  </DMSDriver>
</DMSAuthenticationResponse>

```

B.3. *DMSConfigurationRequest*

Requisição de configuração do DMS.

<!ELEMENT	DMSConfigurationRequest	(DMSDriver)>
<!ELEMENT	DMSDriver	(Time)>
<!ATTLIST	UniqueId	CDATA #REQUIRED>
<!ELEMENT	Time	(#CDATA)>

Essa mensagem possui o identificador do DMS, *UniqueId*, que identifica o DMS para o sistema.

```

<?xml version="1.0" encoding="UTF-8"?>
<DMSConfigurationRequest>
  <DMSDriver UniqueId="216">
    <Time>07/19/2006 20:03:40.078</Time>
  </DMSDriver>
</DMSConfigurationRequest>

```

B.4. *DMSConfigurationResponse*

Resposta à requisição de autenticação do DMS.

<!ELEMENT	DMSConfigurationRequest	(DMSDriver)>
<!ELEMENT	DMSDriver	(Time, ReaderType, ReaderID)>
<!ATTLIST	UniqueId	CDATA #REQUIRED>
<!ELEMENT	Time	(#CDATA)>
<!ELEMENT	ReaderType	(#CDATA)>
<!ELEMENT	ReaderID	(#CDATA)>

A mensagem possui dois tags novos, o *ReaderType* e o *ReaderID*. O primeiro informa ao DMS qual será o *driver* do MCS que ele deverá carregar, e o segundo informa o ID que o dispositivo terá.

```

<?xml version="1.0" encoding="UTF-8"?>
<DMSConfigurationResponse>
  <DMSDriver UniqueId="216">
    <Time>07/19/2006 20:03:44.124</Time>
    <ReaderType>Simulator</ReaderType>
    <ReaderID>777</ReaderID>
  </DMSDriver>
</DMSConfigurationResponse>

```

```

    </DMSDriver>
</DMSConfigurationResponse>

```

B.5. LSIPDAMessage

Mensagem enviada do LSI para o PDA quando um caminhão entra no local de descarga.

<!ELEMENT	LSIPDAMessage	(Truck)>
<!ELEMENT	Truck	(Time, VehicleType, LicensePlate, LoadLicensePlate, Load)>
<!ATTLIST	Truck	
	PlaceID	CDATA #REQUIRED>
	Tag	CDATA #REQUIRED>
<!ELEMENT	Time	(#CDATA)>
<!ELEMENT	VehicleType	(#CDATA)>
<!ELEMENT	LicensePlate	(#CDATA)>
<!ELEMENT	LoadLicensePlate	(#CDATA)>
<!ELEMENT	Load	(#CDATA)>

A mensagem contém o número do tag, o local onde o caminhão entrou e as informações do caminhão, como placa, tipo, etc.

```

<?xml version="1.0" encoding="UTF-8"?>
<LSIPDAMessage>
  <Truck PlaceID="MoegaXX" Tag="216">
    <Time>09/13/2006 19:02:31.094</Time>
    <VehicleType>Caçamba</VehicleType>
    <LicensePlate>AAB-2912</LicensePlate>
    <LoadLicensePlate>HGA-4821</LoadLicensePlate>
    <Load>Adubo</Load>
  </Truck>
</LSIPDAMessage>

```

B.6. DMSPDAMessage

Mensagem enviada do PDA para o DMS quando um caminhão conclui a descarga.

<!ELEMENT	DMSPDAMessage	(Truck)>
<!ELEMENT	Truck	(Time, Unload)>

```

<!ATTLIST Truck
          PlaceID           CDATA #REQUIRED>
          Tag               CDATA #REQUIRED>
<!ELEMENT Time              (#CDATA)>
<!ELEMENT Unload            (#CDATA)>

```

A mensagem contém as informações do tag e do local de descarga, e um campo informando se a descarga foi parcial ou final.

```

<?xml version="1.0" encoding="UTF-8"?>
<DMSPDAMessage>
  <Truck PlaceID="MoegaXX" Tag="216">
    <Time>09/13/2006 19:12:55.252</Time>
    <Unload>Parcial</Unload>
  </Truck>
</DMSPDAMessage>

```

B.7. LSIWeightRequest

Essa mensagem é enviada pelo LSI para o aplicativo de pesagem da empresa, fazendo uma requisição do peso e das informações do caminhão que está na balança.

```

<!ELEMENT LSIWeightRequest      (Truck)>
<!ELEMENT Truck                (Time, Unload)>
<!ATTLIST Truck
          PlaceID           CDATA #REQUIRED>
          Tag               CDATA #REQUIRED>
<!ELEMENT Time              (#CDATA)>
<!ELEMENT State              (#CDATA)>

```

Essa mensagem contém um campo que informa se a pesagem que o caminhão está realizando é pesagem bruta ou pesagem tara.

```

<?xml version="1.0" encoding="UTF-8"?>
<DMSPDAMessage>
  <Truck PlaceID="Balanca1" Tag="216">
    <Time>09/13/2006 19:01:03.202</Time>
    <State>Bruto</State>
  </Truck>
</DMSPDAMessage>

```

B.8. LSIWeightResponse

Essa mensagem é a resposta do sistema de pesagem à requisição de peso feita pelo DMS.

```

<!ELEMENT LSIWeightResponse      (Truck)>
<!ELEMENT Truck                  (Time, Weight,
                                  VehicleType,
                                  LicensePlate,
                                  LoadLicensePlate,
                                  Load)>

<!--
  Truck
  PlaceID      CDATA #REQUIRED
  Tag          CDATA #REQUIRED
-->
<!ELEMENT Time                  (#CDATA)>
<!ELEMENT Weight                (#CDATA)>
<!ELEMENT VehicleType           (#CDATA)>
<!ELEMENT LicensePlate          (#CDATA)>
<!ELEMENT LoadLicensePlate      (#CDATA)>
<!ELEMENT Load                  (#CDATA)>

```

Nessa mensagem vem todas as informações logísticas que o sistema da empresa possui do caminhão, juntamente com o peso capturado na balança.

```

<?xml version="1.0" encoding="UTF-8"?>
<DMSPDAMessage>
  <Truck PlaceID="Balanca1" Tag="216">
    <Time>09/13/2006 19:02:02.813</Time>
    <Weight>16530</Weight>
    <VehicleType>Caçamba</VehicleType>
    <LicensePlate>AAB-2912</LicensePlate>
    <LoadLicensePlate>HGA-4821</LoadLicensePlate>
    <Load>Adubo</Load>
  </Truck>
</DMSPDAMessage>

```

BIBLIOGRAFIA

ARAKAKI, J.; “Técnicas de degeneração no projeto do controle de sistemas produtivos”, Tese (Doutorado), Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

BASTOS, M. R.; “Arquitetura ODP-CIM aplicada na previsão distribuída da carga do sistema elétrico de potência”, Tese (Mestrado), Escola Politécnica, Universidade de São Paulo, São Paulo, 2006.

BAKHURU, K.; “A seamless tracking solution for indoor and outdoor position location” In: Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on Volume 3, Pages: 2029 – 2033, 11-14 Sept. 2005.

BECKER, D.; “Standards-Based Approach Integrates Utility Applications”, In: Computer Applications in Power, IEEE, Volume: 13 Issue: 4, Pages: 13 – 20, October 2000.

BERNI, C. C.; “Implementação em Hardware / Software de um sensor virtual utilizando algoritmo de identificação nebulosa” Tese (Mestrado), Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

BRAY, M.; “Middleware”, <http://www.sei.cmu.edu/str/descriptions/middleware.html>, December 2005, última visita em 01/12/2006.

CABRERA, L. F.; KURT, C.; BOX, D.; “An Introduction to the Web Services Architecture and Its Specifications”, Microsoft Web Services Technical Articles, October 2004.

CHEN, S.; GULATI, S.; HAMID, S.; HUANG, X.; LUO, L.; MORISSEAU-LEROY, N.; POWELL, M. D.; ZHAN, C.; ZHANG, C.; “A three-tier System Architecture Design and Development for Hurricane Occurrence Simulation” In: Information Technology: Research and Education, 2003. Proceedings International Conference, Pages: 113 – 117, 11-13 Aug. 2003.

CHOI, S.; KIM, B.; PARK, J.; KANG, C.; EOM, D.; “An Implementation of Wireless Sensor Network for Security System using Bluetooth” In: IEEE Transactions on Consumer Electronics, Vol. 50, No. 1, Pages: 236 - 244, February 2004.

CURINO, C.; GIANI, M.; GIORGETTA, M.; GIUTI, A.; “TinyLIME: Bridging Mobile and Sensor Networks through Middleware”, 2005.

DOBKIN, D.M.; WEIGAND, S.M. “Environmental effects on RFID tag antennas” In: Microwave Symposium Digest, 2005 IEEE MTT-S International 12-17 June 2005.

FOINA, A. G.; RAMIREZ-FERNANDEZ, F. J.; “Sistema de Proteção e Supervisão de Cargas a Granel com Tecnologia RFID”, In: IBERSENSOR 2006, Anais do IBERSENSOR 2006, 2006.

HECK, S. B.; WILLS, L. M.; VACHTSEVANOS, G. J.; “Software Technology for Implementing Reusable, Distributed Control Systems”, In: IEEE Control Systems Magazine Vol. 23, No. 3, Pages: 21-36, February, 2003.

HUTCHENS, K.; OUDSHOORN, M.; MACIUNAS, K.; “Web-based software engineering process management” In: System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference, Pages: 676 - 685 vol.1, 7-10 January 1997.

IBM; “IBM Software - WebSphere RFID Premises Server - Product Overview”, http://www-306.ibm.com/software/pervasive/ws_rfid_premises_server/, último acesso em 01/12/2006.

IEEE Std 1451.1, "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model", IEEE Inc., New York, EUA, 2000.

KALEJA, M.M.; HERB, A.J.; RASSHOFFER, R.H.; FRIEDSAM, G.; BIEBL, E.M.; “Imaging RFID system at 24 GHz for object localization” In: Microwave Symposium Digest, 1999 IEEE MTT-S International, Volume: 4, Pages: 1497 – 1500, 13-19 June 1999.

KERNER, S. M.; “RFID Middleware Battleground Emerges” <http://www.internetnews.com/bus-news/article.php/3401611>, August 30, 2004.

LEAVER, S.; MENDELSON, T.; OVERBY, C. S.; YUEN, E. H.; “Evaluating RFID Middleware”, August 13, 2004.

LEGNER, C.; THIESSE, F.; “RFID-based maintenance at Frankfurt airport” In: Pervasive Computing, IEEE Volume 5, Issue 1, Pages: 34 – 39 Jan.-March 2006.

LEWIS, R.; “Modeling control systems using IEC 61499” In: IEE Control Engineering series 59, 2001.

LIU, S.; WEI, J.; XU, W.; “Diversifying and improving business process management systems by service cooperation middleware” In: Quality Software, 2003. Proceedings. Pages: 396 – 399, 6-7 November 2003.

MA, Y.; AYLOR, J. H.; “System lifetime optimization for heterogeneous sensor networks with a hub-spoke technology” In: Mobile Computing, IEEE Transactions on Volume 3, Issue 3, Pages: 286-294, July-Aug. 2004.

MASUDA, T.; “Process management and control for heterogeneous domain models” In: Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium, Pages: 127 – 130, 24-28 March 2003.

MEIJER, E.; VELZEN, D.; “Haskell Server Pages - Functional Programming and the Battle for the Middle Tier”, Electronic Notes in Theoretical Computer Science 41 No. 1, 2001.

MOISE, D. L.; WONG, K.; “Extracting and representing cross-language dependencies in diverse software systems” In: Reverse Engineering, 12th Working Conference on 7-11 Nov. 2005.

MOLLA, M. M.; AHAMED, S. I.; “A survey of middleware for sensor network and challenges” In: Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on 14-18 August 2006.

NI, L. M.; LIU, Y.; LAU, Y. C.; PATIL, A. P.; “LANDMARC: indoor location sensing using active RFID” In: Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference, Pages: 407-415, 23-26 March, 2003.

PUTMAN, J. R.; “Architecting with RM-ODP”, Prentice Hall PTR, 2001.

SANTOS FILHO, D. J.; “Aspectos do projeto do controle de sistemas produtivos”, Tese (Livre Docência), Escola Politécnica, Universidade de São Paulo, São Paulo, 2000.

SANZ, R.; “Corba for Control Systems”, Annual Reviews in Control, Vol. 25, pp 169-181, 2001.

SULLIVAN, S.; “Advanced DAO programming”, <http://www-106.ibm.com/developerworks/java/library/j-dao/>, 07 October 2003, ultimo acesso em 01/12/2006.

SUN; “Software Solutions - EPC and RFID”, <http://www.sun.com/software/solutions/rfid/index.xml>, ultimo acesso em 01/12/2006.

MURRAY, G.; “JSR-000154 Java™ Servlet 2.5 Specification”, Community Development of Java Technology Specifications, May 10, 2006.

SHOSHANI, A.; SIM, A.; GU, J.; “Storage Resource Managers: Middleware Components for Grid Storage”, 2002.

TANENBAUM, V. S.; “Distributed Systems – Principles and Paradigms”. Prentice Hall, 2002.

UMAR, A.; “Object-Oriented Client/Server Internet Environments” Prentice-Hall, 1997.

VIEGAS, V. M. R.; “Projecto e Implementação de um Sistema de Sensores Inteligentes Baseado na Norma IEEE 1451”, Tese (Mestrado), Instituto Superior Técnico, Universidade Técnica de Lisboa, 2003.

WSA; “Web Services Architecture”, W3C Working Group Note 11 February 2004.

WU, H.; YING J.; KE, H.; “Process management and improvement in Web age”, In: Machine Learning and Cybernetics, 2005. Proceedings of 2005, Pages: 1196 - 1201 Vol. 2, 18-21 August, 2005.

XIONG G.; LITOKORPI, A.; NYBERG, T. R.; “Middleware-based solution for enterprise information integration” In: Emerging Technologies and Factory Automation, 2001. Proceedings of 8th IEEE International Conference, Pages: 687 - 690 vol.2, 15-18 Oct. 2001.