Rodrigo Palucci Pantoni

**DESENVOLVIMENTO E IMPLEMENTAÇÃO DE UMA DESCRIÇÃO
DE DISPOSITIVOS ABERTA E NÃO-PROPRIETÁRIA PARA
EQUIPAMENTOS FOUNDATION FIELDBUS™ BASEADA EM XML**

Dissertação apresentada à Escola de Engenharia
de São Carlos, da Universidade de São Paulo,
como parte dos requisitos para a obtenção do
Título de Mestre em Engenharia Mecânica.

Área de Concentração: Dinâmica das Máquinas
e Sistemas

Orientador: Prof. Dr. Luis Carlos Passarini

São Carlos

2006

AGRADECIMENTOS

Agradeço primeiramente ao Prof. Dr. Luis Carlos Passarini pela orientação, dedicação e amizade. Não fosse sua ajuda e orientação, este trabalho não se realizaria;

Ao amigo Eduardo Mossin pelo companheirismo em todas as etapas do mestrado;

Ao Prof. Dr. Dennis Brandão pela disposição em ajudar e por acreditar no trabalho;

Ao Omar e Cassius pelos conceitos transmitidos;

À Smar Equipamentos Industriais pelo incentivo à pós-graduação. Se não fosse esse incentivo, este trabalho não se realizaria;

Ao Jaime, que sempre me ajudou no que foi possível;

À Universidade de São Paulo por colocar à disposição sua estrutura;

Aos colegas e funcionários do SMM e da pós-graduação pelo auxílio;

Ao Prof. Luiz Puntel, à Elena e Giovana pela revisão do texto;

A todos os amigos que sempre estiveram presentes;

Sinceramente, aos meus pais e aos meus irmãos, por tudo,

e a Deus, pela vida, por esta oportunidade e por estes irmãos.

RESUMO

PANTONI, R. P. (2006). **Desenvolvimento e implementação de uma descrição de dispositivos aberta e não-proprietária para equipamentos FOUNDATION fieldbusTM baseada em XML**. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2006.

A interoperabilidade entre equipamentos ou aplicativos (*software*) de tecnologias baseadas em redes de campo (*fieldbus*) é de suma importância nos sistemas de automação, pois permite a integração dos mesmos, de diferentes fabricantes. A integração é feita por uma linguagem de comunicação comum usada entre as diferentes tecnologias heterogêneas. No nível do *software* configurador FOUNDATION fieldbusTM (FF) de dispositivos e de malhas de controle, a linguagem usada para garantir a interoperabilidade entre os dispositivos é provida pela tecnologia de descrição de dispositivos chamada *Electronic Device Description* (EDD), que é proprietária e de complexo desenvolvimento para o fabricante, definida pela *Fieldbus Foundation*TM. As tecnologias de *software* abertas e não-proprietárias têm apresentado um grande crescimento nos últimos anos, em especial a tecnologia *eXtensible Markup Language* (XML), que se tornou mundialmente conhecida devido à integração de aplicativos internet, tornando-os assim interoperáveis uns com os outros. É proposta neste trabalho uma tecnologia de descrição de dispositivos aberta, não-proprietária e de simples desenvolvimento (para o fabricante), baseada na tecnologia XML, que é chamada de *Open-EDD* (*Open Electronic Device Description*). A criação dessa tecnologia engloba a definição da linguagem *Open-EDDML* (*Open Electronic Device Description Markup Language*), a modelagem do XML *Schema* e o projeto e implementação de um interpretador (*parser*). Para validação dessa tecnologia, é feita a integração da mesma em um configurador real e testes usando dispositivos FF.

Palavras-chave: Tecnologia aberta. Tecnologia não-proprietária. XML. *Device description*. FOUNDATION fieldbusTM. Interoperabilidade. Sistema de automação industrial. Controle de processos.

ABSTRACT

PANTONI, R. P. (2006). **Developing and implementing an open and non-proprietary device description for FOUNDATION fieldbus™ devices based on XML**. M.Sc. Dissertation – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2006.

The interoperability among devices or software based on fieldbus technologies is becoming increasingly requested and demanded because it allows the integration of devices or software from different manufacturers. The integration is implemented through a common communication language used among the different heterogeneous technologies. In the FOUNDATION fieldbus™ configurator software is used to provide interoperability among devices a proprietary and complex technology (to be developed by manufacturers), known as Electronic Device Description (EDD) that is defined by the Fieldbus Foundation™. Meanwhile, open and non-proprietary software technologies have presented a great growth in the last years, especially the eXtensible Markup Language (XML) technology, which has become worldwide known due to the integration of internet (heterogeneous) applications. This study proposes a new open, non-proprietary and simple (for manufacturer development) device description based on XML, that is named Open-EDD (Open Electronic Device Description). The creation of this technology includes the definition of the language Open-EDDML (Open Electronic Device Description Markup Language), modeling the XML Schema, projecting and implementing the parser. This technology is validated by integrating it to an existent configurator and testing it using FF devices.

Keywords: Open technology. Non-proprietary technology. XML. Device description. FOUNDATION fieldbus™. Interoperability. Industrial automation system. Process control.

LISTA DE FIGURAS

Figura 1 -	Representação da infra-estrutura de comunicação industrial.	25
Figura 2 -	Sistema de automação <i>fieldbus</i> e seus componentes.	25
Figura 3 -	Quadro com a estrutura da norma IEC 6115.	27
Figura 4 -	Modelo de três camadas de rede <i>fieldbus</i>	27
Figura 5 -	“ <i>Profiles</i> ” e protocolos <i>fieldbus</i> conforme as normas IEC 61158 e IEC 61784.	28
Figura 6 -	Típico arranjo do sistema de automação industrial numa planta.	30
Figura 7 -	Diagrama de blocos funcionais de uma estratégia de controle.	32
Figura 8 -	Estratégia de controle desenvolvida de modo <i>offline</i>	33
Figura 9 -	Diagrama da lógica do bloco de controle PID (proporcional-integral-derivativo) e suas entradas e saídas.	34
Figura 10 -	Tela de inserção de redes.	34
Figura 11 -	Tela de inserção de dispositivos.	35
Figura 12 -	Tela de inserção de blocos.	35
Figura 13 -	Tela da topologia de rede.	36
Figura 14 -	Tela do configurador usada para configuração de dispositivos de campo no modo <i>offline</i>	37
Figura 15 -	Tela de descarregamento das configurações nos dispositivos.	37
Figura 16 -	Tela do configurador de redes usada para configuração de dispositivos de campo no modo <i>online</i>	38
Figura 17 -	Camada do usuário FF.	41
Figura 18 -	Arquitetura de <i>software</i> do Syscon.	43
Figura 19 -	Banco de DD’s.	44
Figura 20 -	Interoperabilidade de aplicativos e dispositivos.	47
Figura 21 -	Representação da estrutura em camadas do protocolo FF.	57
Figura 22 -	Codificação Manchester.	58
Figura 23 -	Topologia (a) <i>single link</i> e (b) <i>bridged network</i>	59
Figura 24 -	Fases cíclica e acíclica em macrociclos.	60
Figura 25 -	Componentes da arquitetura do processo de aplicação baseado em blocos funcionais.	62

Figura 26 - Conjuntos de blocos funcionais normatizados.	63
Figura 27 - Tipos de dados FF.	65
Figura 28 - Composição do byte de status.	67
Figura 29 - Exemplo de código fonte HTML.	74
Figura 30 - Exemplo simples de um arquivo XML.	75
Figura 31 - Definição de DTD num arquivo XML.	78
Figura 32 - Definição de DTD fora do arquivo XML.	78
Figura 33 - Exemplo de arquivo XML <i>Schema</i>	80
Figura 34 - Exemplo de visualizações usando XSLT.	81
Figura 35 - Um <i>driver</i> de comunicação para cada fabricante.	83
Figura 36 - Modelo de comunicação padrão com clientes e servidores OPC.	84
Figura 37 - Evolução da EDDL ao longo dos protocolos de comunicação.	88
Figura 38 - Exemplo de separação das visões dos parâmetros de calibração e de configuração.	90
Figura 39 - Exemplo de gráfico de assinatura de válvula.	91
Figura 40 - Exemplo de código EDDL e interpretação de uma variável simples.	94
Figura 41 - Processo de desenvolvimento da DD de um dispositivo.	95
Figura 42 - Trecho do <i>Capability File</i> do dispositivo Deltabar do fabricante Endress+Hauser.	98
Figura 43 - Arquitetura da tecnologia FDT/DTM.	101
Figura 44 - Exemplo de um arquivo GSD de um posicionador de válvulas PROFIBUS PA do fabricante Smar Equipamentos Industriais.	106
Figura 45 - Trecho exemplo de descrição feita com a tecnologia GSDML de um dispositivo.	108
Figura 46 - Definição de um erro em múltiplos idiomas num documento GSDML. .	109
Figura 47 - Editor FDCML para dispositivos INTERBUS.	111
Figura 48 - HTML gerado a partir de um documento FDCML.	112
Figura 49 - Estrutura do DTD definido por Wang et al (2004).	114
Figura 50 - Representação gráfica do XML Schema da <i>Open-EDD</i>	118
Figura 51 - Interpretador de <i>Open-EDD</i> com Xerces C++.	119
Figura 52 - Representação das heranças de classe dos elementos <i>Open-EDDML</i> na notação UML.	121

Figura 53 - Representação das relações de associação entre os elementos <i>Open-EDDML</i> na notação UML.	122
Figura 54 - Arquitetura de <i>software</i> do Syscon modificada.	123
Figura 55 - Banco de <i>Open-EDD</i>	124
Figura 56 - Trecho de código na linguagem C++ para leitura de <i>Open-EDDML</i>	128
Figura 57 - Serviços providos pelas classes do interpretador de <i>Open-EDD</i>	130
Figura 58 - Parte inicial da interface em HTML gerada pelo processamento da linguagem XSLT da <i>Open-EDDML</i> do dispositivo Cerabar S da Endress+Hauser.	130

LISTA DE SIGLAS

AL	<i>Application Layer</i> (Camada de Aplicação)
AP	<i>Aplication Process</i> (Processo de Aplicação)
ASCII	<i>American Standard Code for Information Interchange</i> (Padrão Americano de Código para Troca de Informações)
CF	<i>Capability File</i> (Arquivo de Capacidade do Dispositivo)
CFF	<i>Common File Format</i> (Formato Comum de Arquivo)
CFH	<i>Capability File for HSE</i> (Arquivo de Capacidade para HSE)
COM	<i>Component Object Model</i> (Modelo de Componente de Objeto)
CSMA/CD	<i>Carrier Sense Multiple Access with Collision Detection</i> (Acesso Múltiplo de Sentido de Portador/Detecção de Colisão)
DCS	<i>Distributed Control System</i> (Sistema de Controle Distribuído)
DD	<i>Device Description</i> (Descrição de Dispositivo)
DDC	<i>Direct Digital Control</i> (Controle Digital Direto)
DDL	<i>Device Description Language</i> (Linguagem de Descrição de Dispositivo)
DDS	<i>Device Description Services</i> (Serviços de Descrição de Dispositivo)
DLL	<i>Data Link Layer</i> (Camada de Enlace)
DTD	<i>Document Type Definition</i> (Definição do Tipo de Documento)
DTM	<i>Device Type Manager</i> (Gerenciador de Tipo de Dispositivo)
EDD	<i>Electronic Device Description</i> (Descrição Eletrônica de Dispositivo)

EDDL	<i>Electronic Device Description Language</i> (Linguagem Eletrônica de Descrição de Dispositivo)
ERP	<i>Enterprise Resource Planning</i> (Planejamento de Recursos Empresariais)
FAS	<i>Fieldbus Access Sublayer</i> (Subcamada de Acesso <i>Fieldbus</i>)
FB	<i>Function Block</i> (Bloco Funcional)
FBAP	<i>Function Block Application Process</i> (Processo de Aplicação de Bloco Funcional)
FCS	<i>Field Control System</i> (Sistema de Controle de Campo)
FDCML	<i>Field Device Configuration Markup Language</i> (Linguagem de Marcadores para Configuração de Dispositivo de Campo)
FDT	<i>Field Device Tool</i> (Ferramenta de Dispositivo de Campo)
FF	<i>Fieldbus Foundation</i> TM (Fundação)
FF	<i>FOUNDATION Fieldbus</i> TM (Protocolo)
FFB	<i>Flexible Function Block</i> (Bloco Funcional Flexível)
FMS	<i>Fieldbus Message Layer</i> (Camada de Mensagem <i>Fieldbus</i>)
GSD	<i>General Station Description</i> (Descrição Geral de Estação)
GSDML	<i>General Station Description Markup Language</i> (Linguagem de Marcadores de Descrição Geral de Estação)
HART	<i>Highway Addressable Remote Transducer</i> (Protocolo)
HCF	<i>HART Communication Foundation</i> (Fundação)
HSE	<i>High Speed Ethernet</i> (<i>Ethernet</i> de Alta Velocidade)
HTML	<i>Hypertext Markup Language</i> (Linguagem de Marcadores de Hipertexto)
I/O	<i>Input / Output</i> (Entrada / Saída)

IEC	<i>International Electrotechnical Commission</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
IHM	Interface Homem-Máquina
IP	<i>Internet Protocol</i> (Protocolo de Internet)
ISA	<i>Instrumentation, Systems, and Automation Society</i>
ISO	<i>International Standards Organization</i>
LAS	<i>Link Active Scheduler</i> (Mestre Ativo da Rede <i>Fieldbus</i>)
MAC	<i>Medium Access Control</i> (Controle de Acesso ao Meio)
MES	<i>Manufacturing Execution System</i> (Sistema de Execução de Produção)
OD	<i>Object Dictionary</i> (Dicionário de Objetos)
OLE	<i>Object Linking Embedded</i> (Objeto de Ligação Embarcado)
OO	Orientado a Objetos
OPC	<i>OLE for Process Control</i> (OLE para Controle de Processos)
OPC AE	<i>OPC Alarms and Events</i> (Alarmes e Eventos)
OPC DA	<i>OPC Data Access</i> (Acesso a dados)
OPC DX	<i>OPC Data Exchange</i> (Troca de dados)
OPC HDA	<i>OPC Historical Data Access</i> (Histórico dos dados)
OPC UA	<i>OPC Unified Architecture</i> (Arquitetura Unificada)
Open-EDD	<i>Open Electronic Device Description</i> (Descrição de Dispositivo Eletrônica Aberta)
Open-EDDML	<i>Open Electronic Device Description Markup Language</i> (Linguagem Eletrônica Aberta de Marcadores para Descrição de Dispositivo)
OSI	<i>Open System Interconnection</i>

PC	Computador Pessoal
PDA	<i>Personal Digital Assistant</i> (Assistente Digital Personalizado)
Phy	<i>Physical Layer</i> (Camada Física)
PID	Controlador de ação Proporcional-Integral-Derivativa
PLC	Programador Lógico Controlável
PNO	<i>PROFIBUS Nutzer Organization</i> (Organização PROFIBUS)
QoS	<i>Quality of Service</i> (Qualidade de Serviço)
RO	<i>Read Only</i> (Apenas para Leitura)
SGML	<i>Standard Generalized Markup Language</i> (Linguagem Generalizada Padrão de Marcadores)
SISO	<i>Single Input, Single Output</i> (Uma Entrada, Uma Saída)
SMIB	<i>System Management Information Base</i> (Base de Informação de Administração de Sistema)
ST	<i>Schedule Table</i> (Tabela de Escalonamento)
TCP	<i>Transmission Control Protocol</i> (Protocolo de Controle de Transmissão)
UDP	<i>User Datagram Protocol</i> (Protocolo de Datagrama do Usuário)
UML	<i>Unified Modeling Language</i> (Linguagem Unificada de Modelagem)
USB	<i>Universal Serial Bus</i> (Barramento Serial Universal)
VCR	<i>Virtual Communication Relationship</i> (Relação de Comunicação Virtual)
VFD	<i>Virtual Field Device</i> (Dispositivo de Campo Virtual)
XSDL	<i>XML Schema Definition Language</i> (Linguagem de Definição de XML Schema)

XML	<i>Extensible Markup Language</i> (Linguagem de Marcadores Extensível)
XSL	<i>Extensible Stylesheet Language</i> (Linguagem Extensível de Estilo)
XSLT	<i>Extensible Stylesheet Language Transformation</i> (Linguagem Extensível de Estilo para Transformação)
ZVEI	<i>German Electrical and Electronic Manufacturers Association</i>
W3C	<i>World Wide Web Consortium</i>
WWW	<i>World Wide Web</i>

SUMÁRIO

1	INTRODUÇÃO	21
1.1	Organização deste Trabalho	22
1.2	Evolução dos Sistemas de Controle.....	23
1.3	Redes de Campo	24
1.4	Categorias de <i>Software</i> dos Sistemas de Automação <i>Fieldbus</i>	29
1.5	Configurador <i>Fieldbus</i>	30
1.6	Configurador Syscon	32
1.7	EDD	39
1.8	Era dos Sistemas Abertos	45
1.9	Justificativas	49
1.10	Objetivo.....	51
2	ALGUNS PROTOCOLOS DE COMUNICAÇÃO E SUAS TECNOLOGIAS DE DESCRIÇÃO DE DISPOSITIVOS.....	53
2.1	HART	53
2.2	FOUNDATION <i>Fieldbus</i> TM	56
2.2.1	<i>Blocos FOUNDATION Fieldbus</i> TM	61
2.3	PROFIBUS	69
2.4	INTERBUS.....	71
3	A TECNOLOGIA XML.....	73
3.1	Histórico, Fundamentos e Características.....	73
3.2	OPC e OPC-UA.....	82
4	REVISÃO BIBLIOGRÁFICA	87
4.1	EDDL.....	87
4.2	CF	97
4.3	FDT/DTM.....	98

4.4 GSD	104
4.5 GSDML	107
4.6 FDCML.....	110
4.7 Tecnologias de Descrição de Dispositivos Pesquisadas	113
5 METODOLOGIA.....	117
6 RESULTADOS	127
7 CONCLUSÕES	133
7.1 Sugestões para Trabalhos Futuros.....	134
REFERÊNCIAS	137
APÊNDICE A – XML <i>SCHEMA</i> DA <i>OPEN</i>-EDDML	145
APÊNDICE B – ARQUIVO XSLT OPENEDD.XSL.....	149
APÊNDICE C – EXEMPLO DE UMA DESCRIÇÃO SIMPLES EM <i>OPEN</i>-EDDML	155
APÊNDICE D – ARQUIVO DE ESTILO OPENEDD.CSS	159
ANEXO A – EXEMPLO DE CÓDIGO FONTE EDDL	161

1 Introdução

Computadores pessoais mais recentes possuem uma característica que normalmente não é percebida pelos usuários de computadores: o suporte à integração de dispositivos eletrônicos de diferentes fabricantes ou até mesmo de diferentes tecnologias. Dentre esses dispositivos, pode-se citar *mouse*, *web cam*, *pen drive*, placa usada para prover vídeo, som, rede, etc.

Esses dispositivos possuem um importante atributo chamado de interoperabilidade, que é a capacidade de diferentes soluções poderem operar umas com as outras. Outro conceito ligado é o de intercambiabilidade, que é a capacidade de substituição de soluções de um fabricante pelo de outro, sem perda de funcionalidade.

Nos últimos anos, esses conceitos estão sendo estendidos para os equipamentos, sistemas e tecnologias que compõem os sistemas de automação para controle de processos industriais. Um sistema de automação industrial possui dispositivos ou equipamentos conectados que necessitam operar uns aos outros, a fim de concretizar o controle propriamente dito.

Interoperabilidade e intercambiabilidade são definidas na fase de elaboração do projeto de *software* ou equipamento de forma a seguir um padrão aberto em relação à arquitetura. Nos sistemas de automação *fieldbus*, no nível dos aplicativos de interação com o usuário (ou interface homem-máquina), a tecnologia que tem o papel de “abrir” o sistema para soluções de diferentes fabricantes é chamada de tecnologia de descrição de dispositivos. No caso do protocolo FOUNDATION fieldbusTM (FF), a tecnologia atual padrão usada para descrição de dispositivos é chamada de EDD (*Electronic Device Description*).

1.1 Organização deste Trabalho

No primeiro capítulo, é feita uma introdução de redes de campo e de aplicativo configurador de IHM (interface homem-máquina) *fieldbus* com o objetivo de expor o papel das tecnologias de descrição de dispositivos num nível detalhado, em especial da EDD. Em seguida, é abordada a importância dos sistemas abertos e de uma tecnologia aberta e não-proprietária, que provê a interoperabilidade dos sistemas, chamada tecnologia XML (*eXtensible Markup Language*). No final do primeiro capítulo, são apresentadas as justificativas e o objetivo deste trabalho.

No capítulo 2, são apresentados alguns protocolos de comunicação de redes industriais que usam descrições de dispositivos, em especial o protocolo FOUNDATION fieldbusTM, que é o estudado neste trabalho.

No capítulo 3, é feito um detalhamento da tecnologia XML, que é essencial para o entendimento deste trabalho.

No capítulo 4, é feita a revisão bibliográfica deste trabalho, que engloba as descrições de dispositivos que foram desenvolvidas para os protocolos de comunicação brevemente apresentados no capítulo 2, com a adição de mais algumas tecnologias que foram ou ainda estão sendo pesquisadas.

No capítulo 5, são descritas as metodologias para realização do objetivo proposto.

No capítulo 6, são apresentados os resultados de acordo com o objetivo e metodologias propostas.

Por fim, no capítulo 7, são apresentadas as conclusões extraídas do trabalho desenvolvido e algumas sugestões para trabalhos futuros.

1.2 Evolução dos Sistemas de Controle

Nos primórdios dos sistemas de controle, na chamada “era do controle pneumático”, o controlador era tipicamente situado no campo e era operado localmente.

Posteriormente, com o controle feito usando sinal analógico, foi possível levar o sinal para os transmissores no campo, constituindo um controle direto centralizado (DDC), no qual o dispositivo controlador se situava numa sala de controle. Entretanto, em razão dessa centralização, havia uma fragilidade no sistema, pois uma simples falha no controlador ocasionava a parada total de todas as tarefas de controle regidas pelo mesmo.

A desvantagem citada acima, do controle direto centralizado (DDC), levou à elaboração de uma nova arquitetura, a chamada arquitetura de sistema de controle distribuído (DCS). O DCS foi uma grande evolução em relação ao DDC, pois as tarefas de controle eram distribuídas em vários controladores, trazendo o benefício de uma simples falha apenas afetar uma parte do processo da planta, diferentemente da arquitetura DDC. Entretanto, atualmente o DCS é considerado como sendo apenas menos centralizado quando comparado ao DDC, pois uma simples falha pode causar falhas em cadeia (BERGE, 2002).

Segundo Berge (2002), uma nova arquitetura baseada em capacidades dos instrumentos de campo é a alternativa mais proveitosa, a chamada arquitetura de sistema de controle de campo (FCS). Essa arquitetura não considera os dispositivos de campo como simples periféricos, e sim como possíveis controladores. Em razão dessa natureza mais descentralizada, a arquitetura FCS traz vantagens como escalabilidade¹, maior segurança e menor custo. Ainda segundo Berge (2002), a *Fieldbus Foundation*TM (FF) seguiu o conceito da arquitetura FCS, gerando assim especificações para o protocolo FF.

¹ Uma definição sobre escalabilidade será apresentada ainda no capítulo 1, no tópico “Era dos Sistemas Abertos”.

Em razão da evolução das tecnologias para automação e controle, os sistemas de controle estão inseridos num contexto mais amplo, mais abrangente ao dos problemas relacionados à ciência matemática envolvida.

Esse contexto mais amplo é chamado de sistema de automação, pois envolve aspectos relacionados às particularidades do protocolo de comunicação, tais como temporização, determinismo, técnicas de manutenção avançadas para os dispositivos, troca de informações entre os vários componentes do sistema, padrões abertos desses componentes e interoperabilidade entre aplicativos e entre dispositivos do sistema.

1.3 Redes de Campo

As soluções em rede em geral trouxeram uma série de benefícios como possibilitar a transferência de dados entre dispositivos, a redução da complexidade nas conexões de cabos elétricos, redução nos custos de comunicação de dados, a fácil manutenção e a flexibilidade das aplicações.

Em virtude desses benefícios, companhias e institutos motivaram-se em aplicar as soluções em rede para controle de processos industriais, gerando as redes de campo ou *fieldbus* (TIPSUWAN e CHOW, 2003).

Um sistema de automação *fieldbus* pode ser definido como:

[...] um sistema distribuído composto por dispositivos de campo e equipamentos de controle e de monitoramento integrados em um ambiente físico de uma planta ou uma fábrica. Os dispositivos da rede *fieldbus* trabalham em conjunto para realizar I/O e controle em operações e processos automáticos (FIELDBUS FOUNDATION, 1999a).

As redes de campo são características de chão de fábrica na infra-estrutura de comunicação digital de uma indústria com processos produtivos automáticos, na base da pirâmide mostrada na figura 1.

Figura 1 - *Representação da infra-estrutura de comunicação industrial.*
 Fonte: Brandão (2005).

Os dispositivos de campo e equipamentos de controle como, por exemplo, transmissores inteligentes (de pressão, temperatura, densidade), atuadores, controladores lógicos programáveis, estações de supervisão e de configuração, conectam-se em redes de comunicação exclusivamente digitais segundo um protocolo de comunicação especialmente definido para a classe do processo controlado, como representa a figura 2.

Figura 2 - *Sistema de automação fieldbus e seus componentes.*

A operação e o procedimento de comunicação das redes *fieldbus* é regido por um conjunto de estruturas e regras denominado protocolo de comunicação. Os protocolos *fieldbus* baseiam-se no modelo de referência ISO/OSI (*International Standards Organization / Open System Interconnection*) de sete camadas de rede (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 1994).

O modelo é constituído por um conjunto de camadas independentes e hierarquicamente sobrepostas, no qual cada camada, responsável por uma classe de tarefas, se comunica com as camadas adjacentes para realizar as tarefas. O modelo se baseia em uma proposta desenvolvida pela ISO como um primeiro passo para a padronização internacional dos diversos protocolos. O modelo é chamado de modelo de referência ISO/OSI para interconexão de sistemas abertos, pois lida com a conexão de sistemas abertos, isto é, sistemas que são abertos à comunicação com outros sistemas em relação à arquitetura (TANENBAUM, 1994).

O grande número de classes de processos industriais, cada qual com suas características lógicas e temporais particulares, promoveu o surgimento de um grande número de protocolos *fieldbus* a partir do início da década de 1980 (FELSER e SAUTER, 2002). Essa diversidade de protocolos *fieldbus* implicou para os usuários a ausência de um padrão de compatibilidade entre instrumentos de diferentes fabricantes.

O organismo internacional IEC, em 1999, em conjunto com representantes de grandes empresas e organizações do ramo de controle de processo, adotou a resolução de criar uma norma internacional IEC 61158 com o título “*Digital Data Communication for Measurement and Control - Fieldbus for use in Industrial Control Systems*”, que incluiu todos os principais protocolos *fieldbus*. Essa norma foi lançada em 2000 (INTERNATIONAL ELECTROTECHNICAL COMMISSION, 2000) e é composta por oito volumes, conforme apresenta a figura 3.

Volume	Título	Conteúdo
IEC 61158-1	Introdução	Relatórios técnicos
IEC 61158-2	Phy: Camada Física	Oito tipos
IEC 61158-3	DLL: Serviços da Camada de Enlace	Oito tipos
IEC 61158-4	DLL: Protocolos da Camada de Enlace	Oito tipos
IEC 61158-5	AL: Serviços da Camada de Aplicação	Dez tipos
IEC 61158-6	AL: Protocolos da Camada de Aplicação	Dez tipos
IEC 61158-7	Gerência de Rede	Necessita revisão
IEC 61158-8	Teste de Conformidade	Trabalho cancelado

Figura 3 - *Quadro com a estrutura da norma IEC 6115.*
 Fonte: Felser e Sauter (2002).

Os protocolos *fieldbus* seguem, no mínimo, as camadas descritas na figura 4 (b). Cada protocolo pode incluir mais camadas, sempre seguindo o modelo de referência ISO/OSI.

Figura 4 - *Modelo de três camadas de rede fieldbus.*

Os conjuntos de tipos de camadas definidos na norma IEC 61158 não têm referência prática para a implementação dos protocolos descritos, ou seja, não existe na norma uma indicação de arranjos possíveis entre os tipos de maneira que se possa compor a disposição

das camadas e configurar um sistema. Essa carência foi suprida com a publicação da norma IEC 61784 (INTERNATIONAL ELECTROTECHNICAL COMMISSION, 2003), que define “*profiles*”. Cada “*profile*” se refere a um protocolo *fieldbus* que compõe a norma IEC 61158, num total de sete “*profiles*”. Por sua vez os “*profiles*” podem ser subdivididos (figura 5) em:

Profile IEC 61784	Protocolo da IEC 61158			Nome Usual
	Phy	DLL	AL	
CPF-1/1	Tipo 1	Tipo 1	Tipo 9	FOUNDATION Fieldbus (H1)
CPF-1/2	Ethernet	TCP/UDP/IP	Tipo 5	FOUNDATION Fieldbus (HSE)
CPF-1/3	Tipo 1	Tipo 1	Tipo 9	FOUNDATION Fieldbus (H2)
CPF-2/1	Tipo 2	Tipo 2	Tipo 2	ControlNet
CPF-2/2	Ethernet	TCP/UDP/IP	Tipo 2	Ethernet/IP
CPF-3/1	Tipo 3	Tipo 3	Tipo 3	PROFIBUS-DP
CPF-3/2	Tipo 1	Tipo 3	Tipo 3	PROFIBUS-PA
CPF-3/3	Ethernet	TCP/UDP/IP	Tipo 10	PROFINet
CPF-4/1	Tipo 4	Tipo 4	Tipo 4	P-Net RS-485
CPF-4/2	Tipo 4	Tipo 4	Tipo 4	P-Net RS-232
CPF-5/1	Tipo 1	Tipo 7	Tipo 7	WorldFIP (MPS,MCS)
CPF-5/2	Tipo 1	Tipo 7	Tipo 7	WorldFIP (MPS,MCS,SubMMS)
CPF-5/3	Tipo 1	Tipo 7	Tipo7	WorldFIP (MPS)
CPF-6/1	Tipo 8	Tipo 8	Tipo 8	INTERBUS
CPF-6/2	Tipo 8	Tipo 8	Tipo 8	INTERBUS TCP/IP
CPF-6/3	Tipo 8	Tipo 8	Tipo 8	INTERBUS Subset
CPF-7/1	Tipo 6	Tipo 6	-	Swiftnet transport
CPF-7/2	Tipo 6	Tipo 6	Tipo 6	Swiftnet full stack

Figura 5 - “Profiles” e protocolos *fieldbus* conforme as normas IEC 61158 e IEC 61784.
Fonte: Felser e Sauter (2002).

Dentre os diversos protocolos de *fieldbus* normatizados pela IEC, este trabalho enfocase no estudo do protocolo FF do “*profile*” H1, mais especificamente no elemento responsável

pela descrição de dispositivos, a EDD. Será mostrado, nos capítulos a seguir, que a normatização de um protocolo inclui a normatização da tecnologia de descrição de dispositivos, pela mesma fazer parte do protocolo.

1.4 Categorias de *Software* dos Sistemas de Automação *Fieldbus*

Existem três categorias de *software* de IHM inseridos nos sistemas de automação industrial baseados em *fieldbus*: de configuração de dispositivos e malhas de controle, de supervisão do processo e de gerenciamento de ativos ou de patrimônio.

Os configuradores de dispositivos e malhas de controle são usados anteriormente à operação da planta, usado para programação da estratégia de controle e na iniciação do mesmo no campo.

Os supervisórios são usados na operação do processo, isto é, após o processo estar em operação. Profissionais denominados operadores de processo monitoram as variáveis de processo através de telas de *software*, onde checam continuamente o processo com o objetivo de notar alguma anormalidade.

Os de gerenciamento de patrimônio (ou de *Asset Management*) são usados para gerenciar a planta com objetivo de evitar a parada do processo. Esse tipo de *software* faz diagnósticos nos dispositivos com o objetivo de prever com a maior precisão possível quando ocorrerá uma manutenção, permitindo assim que o operador planeje as ações sobre a parada do processo. Além disso, é capaz de realizar ações de manutenção como calibração dos instrumentos.

As categorias descritas acima são basicamente divididas por funcionalidades de *software*. É comum existir aplicativos que tenham mais de um papel, ou que englobem mais de uma categoria em sua funcionalidade; por exemplo, um configurador pode conter

funcionalidades de diagnóstico e manutenção, funcionalidades que estariam em um *software* de gerenciamento de patrimônio.

Baseado nisso, existem muitas combinações nos arranjos das arquiteturas de *software* nas plantas industriais. Um exemplo de arranjo típico é mostrado na figura 6, que possui uma estação para cada categoria de *software*. Nesse caso, a estação onde se situa o configurador é chamada de estação de engenharia, a do supervisor é chamada de estação de operação e a do gerenciador de patrimônio é chamada de estação de manutenção.

Figura 6 - Típico arranjo do sistema de automação industrial numa planta.

1.5 Configurador *Fieldbus*

Configurador de dispositivos e malhas de controle é um *software* que é executado em dispositivos móveis (PDA²), ou em computadores.

² PDA (*Personal Digital Assistant*): Nome genérico utilizado para computadores de mão ou de bolso. PDA's (ou *handhelds*) são aparelhos de mão que reúnem, num único dispositivo, a funcionalidade de um computador, de um telefone/fax e de comunicação via redes.

Pode-se citar como vantagens da utilização de um *software* configurador: configuração da estratégia de controle de forma remota e automatização da configuração e da documentação. É preciso frisar que algumas configurações de protocolos específicos necessitam obrigatoriamente de um configurador, em virtude da sua natureza de operação e configuração. Um exemplo é o protocolo FF. A complexidade ligada à configuração desses protocolos cria a necessidade de o usuário ter uma automatização da configuração e da documentação.

Existem diversos configuradores móveis e destinados a computadores. Podem-se citar como exemplo de configuradores que são executados em PDA's: Mobrey HPC301 (EMERSON PROCESS MANAGEMENT, 2006b) e CONF600HH (SMAR RESEARCH, 2005). Em relação aos aplicativos que são executados em computadores, podem-se citar Syscon (SMAR EQUIPAMENTOS INDUSTRIAIS, 2005b), Simatic PDM (SIEMENS, 2005) e DeltaV (EMERSON PROCESS MANAGEMENT, 2006a).

A popularidade, robustez e confiabilidade das tecnologias envolvidas com computadores, como a rede *ethernet* e sistemas operacionais assim como o próprio computador possibilitaram a maior difusão e desenvolvimento de aplicativos que são executados em computadores.

No protocolo FF, a configuração de redes é feita elaborando-se uma estratégia de controle, construindo-se uma topologia e atribuindo-se valores para as variáveis dos dispositivos de campo. Tipicamente, a ferramenta de configuração permite ao usuário trabalhar na estratégia de controle no modo *online* ou *offline*. O modo *online* seria o configurador ter comunicação direta com os dispositivos, enquanto que o *offline* seria sem comunicação ou sem a presença dos dispositivos de campo. No modo *offline*, todas as mudanças na configuração são armazenadas em um banco de dados, não nos dispositivos de campo. Uma vez que a configuração *offline* está completa e o processo no campo está pronto

para ser iniciado, a ferramenta pode entrar no modo *online* e descarregar a configuração *offline*, que será armazenada agora nos dispositivos de campo. Assim, quando os dispositivos estiverem presentes e conectados, o configurador descarrega a configuração entre os dispositivos. No caso do protocolo FF, essa operação de descarregamento é chamada de *download* de configuração.

A *Fieldbus Foundation*TM (FF), que administra o protocolo FF, especifica para programação da estratégia de controle uma linguagem de programação gráfica simples que seria um diagrama de blocos funcionais. Tais blocos são programas residentes nos dispositivos da rede e encapsulam funções e algoritmos básicos de automação e controle de processos. A configuração distribui os blocos funcionais em diferentes equipamentos de campo, caracterizando assim uma estratégia de controle distribuída como mostrado na figura 7.

Figura 7 - Diagrama de blocos funcionais de uma estratégia de controle.

1.6 Configurador Syscon

Para efeito de entendimento, torna-se necessário um detalhamento do aplicativo configurador usado neste trabalho, pois a validação do mesmo é feita realizando-se testes nesse aplicativo. Este configurador, chamado de Syscon (SMAR EQUIPAMENTOS

INDUSTRIAIS, 2005b), é usado para realizar configurações em dispositivos FF, programando-se assim estratégias de controle de processos reais.

As operações básicas e essenciais de configuração do Syscon são: configuração *offline* de estratégia de controle, construção da topologia de rede, configuração *offline* de valores dos parâmetros dos blocos, *download* e configuração *online* de valores dos parâmetros dos blocos.

A configuração da estratégia de controle (modo *offline*) é feita com a programação da linguagem do diagrama gráfico de blocos funcionais. A figura 8 mostra a tela de configuração gráfica da estratégia de controle usando os blocos funcionais, representados por círculos. As entradas e saídas dos blocos funcionais são os chamados parâmetros dos blocos de entrada e saída. As ligações (chamados de *links*) entre os parâmetros de entrada e saída dos blocos são representadas por setas que são as conexões entre os blocos.

Figura 8 - Estratégia de controle desenvolvida de modo *offline*.

Para cada requisição de construção de um *link* entre os blocos, o Syscon mostra uma tela com a lógica do bloco e suas entradas e saídas (figura 9).

Figura 9 - Diagrama da lógica do bloco de controle PID (proporcional-integral-derivativo) e suas entradas e saídas.

A construção da topologia é feita com a inserção de redes FF, configuração de tempo para taxa de controle, inserção de dispositivos nas redes de campo e inserção de blocos nos dispositivos. As figuras 10, 11 e 12 mostram respectivamente as telas de inserção de redes (*fieldbus*), de dispositivos (*device*) e de blocos (*block*).

Figura 10 - Tela de inserção de redes.

The 'New Device' dialog box has two tabs: 'Device' and 'Advanced Options'. The 'Device' tab is active. It contains the following fields:

- Manufacturer: Endress+Hauser GmbH (dropdown)
- Device Type: CERABAR S (dropdown)
- Device Rev.: 02 (dropdown)
- DD Rev.: 01 (dropdown)
- CF Rev.: 01 (dropdown)
- ☒ Follow the Latest DD and CF Revisions
- Device Id: (text field)
- Device Tag: (text field)

At the bottom are three buttons: OK, Cancel, and Help.

Figura 11 - Tela de inserção de dispositivos.

Na tela de inserção de dispositivos (figura 11), primeiramente é escolhido o fabricante (*manufacturer*) do dispositivo, depois o tipo de dispositivo (*device type*) disponível por tal fabricante e, por fim a versão (*device rev.*) do mesmo a ser criado.

The 'New Block' dialog box contains the following fields:

- Manufacturer: SMAR (dropdown)
- Device Type: TT302 (dropdown)
- Device Rev.: 04 (dropdown)
- DD Rev.: 02 (dropdown)
- CF Rev.: 01 (dropdown)
- Block Type: Analog Alarm (dropdown)
- Profile: A list box showing 'Analog Alarm', 'Analog Input' (highlighted), 'Arithmetic', 'Constant', 'Enhanced PID', 'Input Selector', and 'Integrator'.
- Block Tag: (text field)

At the bottom are three buttons: OK, Cancel, and Help.

Figura 12 - Tela de inserção de blocos.

Na tela de inserção de blocos do dispositivo, é escolhido apenas o bloco a ser criado.

A figura 13 mostra a tela do configurador depois da construção de uma topologia FF: o nó chamado “DFI302” representa um controlador; os nós “Fieldbus 1”, “Fieldbus 2” e “Fieldbus 3” representam as redes *fieldbus* (FF) criadas e conectadas nesse controlador; a janela chamada “Fieldbus 1” exibe graficamente uma visão expandida contendo os dispositivos de campo conectados na rede *fieldbus* (“Fieldbus 1”). Cada dispositivo (“TT1” e “FI1”) possui seus blocos configurados, representados por quadrados (“TT1-RB -1”, “TT1-TRDDSP -1”, etc).

Figura 13 - Tela da topologia de rede.

A configuração dos dispositivos de campo no modo *offline* é feita através de mudanças nos valores dos parâmetros dos blocos funcionais dos dispositivos. A figura 14 mostra a tela do configurador de mudança de valores dos parâmetros *offline* de um bloco funcional PID (bloco de controle de ação proporcional-integral-derivativa). Os valores posicionados na coluna “Value” são dos parâmetros configurados.

Figura 14 - Tela do configurador usada para configuração de dispositivos de campo no modo offline.

Figura 15 - Tela de descarregamento das configurações nos dispositivos.

Assim que a configuração no modo *offline* é realizada por completo, englobando assim a configuração da estratégia de controle, topologia e valores dos parâmetros dos blocos funcionais, a operação de descarregamento ou *download* (figura 15) é necessária para transferir essas configurações para os dispositivos, de forma automática.

Com a configuração descarregada nos dispositivos, é possível que sejam realizadas operações de configuração diretamente nos dispositivos de campo, de modo *online*. Esse tipo de configuração é feita através de mudanças de valores de parâmetros dos blocos funcionais. A figura 16 mostra a tela do configurador de mudança de valores dos parâmetros no modo *online*. Nessa tela, a coluna “Value” mostra os valores lidos das variáveis do dispositivo, e a coluna “Quality”, usada para garantir a confiabilidade da monitoração, indica a qualidade do valor no momento em que é monitorado.

Figura 16 - Tela do configurador de redes usada para configuração de dispositivos de campo no modo online.

1.7 EDD

Para que o *software* configurador possa realizar configurações nos dispositivos, configurar as redes *fieldbus* e programar a lógica da estratégia de controle, é necessário que o próprio saiba interagir com os dispositivos. Para isso, o *software* necessita ter informações sobre as características dos equipamentos, tais como nome e tipo das variáveis, funções das variáveis, as entradas e saídas, etc.

No caso do FF, a descrição das características dos dispositivos é disponibilizada para o configurador através da tecnologia *Electronic Device Description* (EDD). O *software* configurador tem um banco de EDD's disponíveis em seu domínio, que compõe todos os dispositivos suportados pelo configurador, uma EDD para cada dispositivo. Conseqüentemente, a integração de um novo dispositivo no sistema no nível do configurador é feito integrando-se a EDD do mesmo.

Além da descrição, o principal papel da EDD é possibilitar a interoperabilidade de dispositivos ou equipamentos de diferentes fabricantes nos diferentes sistemas para automação de também diferentes fabricantes. Segundo Kastner e Kastner-Masilko (2004), a criação das tecnologias de descrição em geral ocorreu devido à incompatibilidade de dispositivos que não eram integrados em configuradores, ao uso obrigatório de diferentes configuradores (em geral um por fabricante) nas plantas industriais, ao difícil gerenciamento de novas versões dos dispositivos até dos mesmos fabricantes e a incompatibilidade de plataforma de alguns configuradores em relação a outros configuradores já usados na planta.

Pode-se fazer uma analogia da EDD com uma característica apreciada em computadores pessoais, o chamado “*plug and play*”³ da tecnologia USB (*Universal Serial Bus*): o simples

³ A tecnologia “*plug and play*” (“ligar e usar”) permite que o computador reconheça e configure automaticamente qualquer dispositivo que seja instalado, facilitando a expansão segura do sistema e eliminando a necessidade de uma prévia configuração manual.

fato da conexão do dispositivo no computador já dá condições de o usuário operar o dispositivo USB sem necessidade de uma prévia configuração. Na visão do usuário, a tecnologia USB é muito simples; o usuário não precisa conhecer qual o fornecedor do dispositivo USB e nem qual o sistema operacional de computadores com que tal dispositivo seja compatível. Segundo Zielinsky (2005), a tecnologia USB também possui uma tecnologia semelhante à EDD nos computadores, que já vem embutida nos sistemas operacionais, assim como os configuradores de redes FF têm o banco de EDD's de dispositivos a ser utilizado.

A EDD também é conhecida por DD (*Device Description*), que é o nome dado à versão antecessora da EDD. Por essa razão, neste trabalho, as siglas DD e EDD expressam a mesma tecnologia, com a diferença de versão.

Uma das primeiras publicações da DD foi na Alemanha. Na época, a mesma foi apresentada como a oitava camada do modelo de referência ISO/OSI (BORST, 1991). A maioria dos especialistas da área não entendeu essa definição pelo fato desse modelo ter apenas sete camadas.

Na arquitetura do protocolo FF, a EDD (ou DD) localiza-se na chamada camada do usuário, junto com os blocos, pois é esta que descreve as informações dos blocos para um nível mais alto, o do *software* configurador, por exemplo (FIELDBUS FOUNDATION, 2006). A figura 17 mostra o modelo de camadas⁴ FF e localização da camada do usuário, que é acima da sétima camada de referência do modelo ISO/OSI.

A DD foi criada com o objetivo de padronizar uma descrição de equipamentos a todos os equipamentos de diferentes fabricantes de um protocolo específico. Essa descrição padrão possibilita a interoperabilidade entre equipamentos e entre aplicativos de IHM de diferentes fabricantes.

⁴ O modelo de camadas FF é detalhado no tópico “FOUNDATION Fieldbus™”, do capítulo 2.

Figura 17 - Camada do usuário FF.

EDD's são arquivos que descrevem as características dos blocos de dispositivos para o *software* configurador (ou qualquer outro tipo de aplicativo do sistema de automação *fieldbus*), isto é, eles possuem uma linguagem em comum para a descrição dos diferentes dispositivos dos diferentes fabricantes. São usadas nos protocolos FF, HART e PROFIBUS e são implementadas usando-se a linguagem padrão EDDL, que será abordada na revisão bibliográfica (capítulo 4) deste trabalho.

Além da EDD, existem outras tecnologias de descrição que foram criadas para prover a interoperabilidade dos sistemas para outros protocolos de comunicação industrial: PROFIBUS introduziu GSD (*General Slave Data*); PROFINET, a GSDML (*General Station Description Markup Language*) e INTERBUS, a FDCML (*Field Device Configuration Markup Language*).

Alguns protocolos ainda como o FF, HART e PROFIBUS utilizam também, além da EDD, uma tecnologia substituta alternativa ou até considerada complementar (a EDD), a chamada FDT/DTM (*Field Device Tool/Device Type Manager*).

A normatização dos protocolos de comunicação descrita na figura 5 também inclui intrinsecamente a normatização das tecnologias de descrição de dispositivos de cada

protocolo. Por exemplo, todos os desenvolvedores de dispositivos FF seguem a especificação *FOUNDATION Specification Device Description*, FF-900 (FIELDBUS FOUNDATION, 1999c) para desenvolver as DD's FF.

No protocolo FF, as informações da EDD podem ser usadas pelo configurador em duas situações: coleta de informações das características dos dispositivos para configuração *offline* e para comunicação com os dispositivos (*online*) para configurações diretas nos dispositivos.

A configuração no modo *offline* foi possibilitada com o advento das tecnologias de descrição de dispositivos, por descrever as características dos dispositivos integralmente, como se os mesmos estivessem conectados no sistema. Tal modo permite que a iniciação do trabalho de engenharia seja num local diferente do da planta, onde efetivamente ocorrerá o controle do processo.

O modo de configuração *online* também utiliza as informações da EDD para realizar configurações diretas com os dispositivos, funcionando como *drivers*. Tais configurações diretas podem ser, por exemplo: envio de comandos de escrita ou leitura para os parâmetros dos blocos dos dispositivos, instanciação de blocos funcionais nos dispositivos, descarregamento da estratégia de controle, etc.

A arquitetura concebida ao Syscon para interagir com os dispositivos e redes FF é mostrada na figura 18. A tecnologia usada na versão mais recente de Syscon (6.1) é ainda uma versão anterior a EDD (de versão 5), a chamada DD (versão 4), como mostrada no esquema.

O configurador Syscon interage com um interpretador de DD proprietário chamado DDS (*Device Description Services*), que por sua vez lê as DD's e retorna as informações para o configurador, de modo que o mesmo possa realizar a configuração *offline*.

Figura 18 - Arquitetura de software do Syscon.

Quando o configurador realiza alguma operação *online*, ele requisita ao componente servidor OPC (*OLE⁵ for Process Control*) para acessar a rede e, assim, os dispositivos no campo. Entretanto, o servidor OPC precisa das descrições dos dispositivos para poder acessá-los ou configurá-los; então o mesmo requisita ao servidor de DD as informações das descrições dos dispositivos e assim interage com os mesmos. Será abordado no capítulo 3 um tópico sobre o servidor OPC, chamado de “OPC e OPC UA”.

O banco de DD's usado pelo Syscon é armazenado em um diretório chamado *Device Support*. Os diretórios que estão na raiz do *Device Support* são relativos ao fabricante e são identificados pelo número único do fabricante (*manufacturerId*), o qual é controlado pela *Fieldbus FoundationTM*. Nos diretórios dos fabricantes, existem os diretórios dos dispositivos, que são identificados pelo número do tipo do dispositivo (*deviceTypeId*). Em cada um desses diretórios, existe um arquivo de DD binário proprietário (extensão *ffo*), um arquivo de lista de

⁵ OLE (*Object Linking Embedded*): Também conhecido por COM (*Component Object Model*), ou ActiveX, é uma tecnologia desenvolvida pela Microsoft usada para componentização de partes de um *software*.

símbolos do arquivo binário (extensão sym), e um arquivo de CF (*Capability File*, de extensão cff) para cada versão de equipamento. O nome do arquivo de DD e do da lista de símbolos é composto pelo número de revisão do dispositivo (*device revision*) nos primeiros dois algarismos e o número de revisão de DD (*dd revision*) nos dois últimos algarismos. No caso do nome de arquivo de CF, os quatro primeiros algarismos seguem a mesma política do arquivo de DD, porém existem mais dois algarismos que contêm o número de revisão do CF (*cf revision*).

Para equipamentos FF do “*profile*” H1, são usadas duas tecnologias de descrição, que têm objetivos distintos: A EDD e o CF. A EDD FF descreve informações relativas aos blocos e parâmetros. O CF é muito menos abrangente e simples. Ele é um simples arquivo que descreve as capacidades do dispositivo como, por exemplo, memória máxima para instanciação de blocos e criação de *links*, entre outras informações. A EDD (EDDL) e CF FF serão abordados com mais detalhes no capítulo 4.

Figura 19 - Banco de DD's.

1.8 Era dos Sistemas Abertos

Pode-se definir um software aberto como um *software* que possui uma arquitetura que foi especialmente projetada para prever integração com outras tecnologias, um requisito não-funcional do projeto de *software*.

Um outro aspecto envolvido é o conceito de padrão aberto, que pode ser implementado livremente sem restrições legais ou comerciais, isto é, se for necessário o pagamento de *royalties*⁶ então o *software* não é aberto (no sentido de livre). A questão de propriedade não está vinculada estritamente ao custo, mas sim aos direitos sobre o objeto (tecnologia no caso) de forma mais ampla. Na prática, o que caracteriza uma tecnologia como proprietária ou não-proprietária é a licença de uso vinculada à tecnologia pelo detentor dos direitos intelectuais sobre a mesma. Ou a tecnologia é de domínio público, e nesse caso é inerentemente não-proprietária, ou é de domínio de uma pessoa física ou jurídica que detém os direitos sobre a mesma. Nesse segundo caso, esse detentor tem a premissa legal de associar uma licença de uso para sua propriedade. Essa licença pode ser restritiva, no sentido de que confere ao usuário um subconjunto menor de direitos em relação àqueles do proprietário. Licenças restritivas são inerentemente discriminatórias ao impor diferentes liberdades de uso a diferentes usuários, restringindo, por exemplo, quem, como, quando ou onde a tecnologia pode ser utilizada. Por outro lado, se a licença é não-restritiva, está garantida a igualdade de direitos de uso, de onde surge o termo “livre”, que é utilizado, por exemplo, em “*Free Software*”. O termo “aberto” também é utilizado como em “*Open Source*” (MONACO, 2006).

⁶ *Royalt* (palavra inglesa): Importância cobrada pelo proprietário de uma patente de produto, processo de produção, marca, produto, entre outros, ou pelo autor de uma obra, para permitir seu uso ou comercialização. Plural: *royalties*.

Neste trabalho o termo “aberto” é usado em relação à arquitetura projetada para prever integração, enquanto que o termo “não-proprietário” é usado no sentido de liberdade de uso, em relação à licença de uso envolvida.

A indústria de computadores pessoais popularizou o sucesso das soluções abertas, mostrando que sistemas compostos a partir de dispositivos de diferentes fabricantes através da definição de padrões abertos oferecem uma série de vantagens ao cliente, usuário final e fabricante de dispositivos.

O maior apelo dos sistemas abertos (proprietários ou não) é a liberdade de escolha de soluções que proporcionam ao cliente e ao usuário final. Se o cliente ou usuário final não está “preso” por uma arquitetura fechada, ele tem à sua disposição para escolha uma vasta gama de equipamentos e soluções de diversos fabricantes. Por outro lado, o fabricante de dispositivos tem a vantagem de se preocupar em apenas desenvolver equipamentos, ao invés de também criar o suporte aos mesmos, já que a maioria dos fabricantes é especializada, ou em desenvolver sistemas de *software*, ou dispositivos eletrônicos ou mecânicos.

A era dos sistemas abertos chegou com a promessa de trazer para o chão de fábrica os mesmos benefícios que os usuários e fabricantes de computadores pessoais já conheciam. A indústria de automação e controle de processos, que é por tradição conservadora, não escapou de seguir essa moderna tendência, e foi dirigida pela pressão do mercado a definir padrões para permitir soluções abertas (proprietárias ou não), seguindo normas dos institutos que ditam os padrões para tecnologias de chão de fábrica.

Os grandes sistemas de automação eram compostos até pouco tempo atrás, por soluções tipicamente fechadas, em virtude das dificuldades técnicas para se implementar soluções abertas, comumente consideradas utópicas. Essa era uma justificativa conveniente para os grandes fornecedores de sistemas, pois suas soluções lhes permitiam manter cativos seus clientes, de forma que se um desses clientes adquirisse um dispositivo de um determinado

fabricante, ele teria que adquirir também desde parafusos e chaves de fenda até aplicativos de configuração, manutenção e operação dos dispositivos, e não raro até serviços de configuração, instalação e assistência técnica do mesmo fabricante. Essa era uma razão plausível para que grandes fornecedores de soluções fechadas ignorassem ou tentassem retardar a era das soluções abertas na área de automação e controle de processos industriais, desacreditando tais soluções ao declará-las utópicas.

O conceito de sistema aberto em relação à arquitetura é muito abrangente. Um sistema pode ser mais aberto ou menos aberto dependendo do grau em que ele apresenta cada um dos cinco atributos seguintes: interoperabilidade, intercambiabilidade, interconectividade, extensibilidade e escalabilidade (DONAIRES, 2004).

Interoperabilidade é a capacidade de diferentes soluções de diferentes fabricantes como dispositivos e ou aplicativos poderem operar umas às outras.

Figura 20 - *Interoperabilidade de aplicativos e dispositivos.*

Intercambiabilidade é a capacidade de substituição de um equipamento, máquina ou aplicativo de um fabricante pelo de outro sem perda de funcionalidade.

Interconectividade é a capacidade de conectar em rede equipamentos, máquinas e aplicativos através de canais de comunicação de forma que eles possam trocar informações e interpretar as informações trocadas.

O sistema apresenta extensibilidade quando novas funcionalidades podem ser incluídas pela simples adição de novos equipamentos, máquinas e aplicativos sem impacto nas funcionalidades já existentes no sistema.

A escalabilidade é a capacidade de o sistema agregar mais aplicativos ou equipamentos, de forma que crie desde pequenas e simples até grandes e complexas configurações.

Um outro atributo ligado a sistema aberto é o de portabilidade, que é a capacidade de um *software* ser compilado e executado em diferentes plataformas, de *hardware* ou de sistemas operacionais, de forma que sejam feitas apenas pequenas alterações no mesmo. Se um *software* for compilado e executado tanto num computador pessoal quanto num PDA, por exemplo, com modificações mínimas em seu código fonte, ele é considerado portátil.

Extensibilidade e escalabilidade são requisitos não-funcionais desejáveis em sistemas em geral, inclusive em sistemas proprietários. Dos atributos citados, a interoperabilidade, intercambialidade e portabilidade seriam os mais diretamente ligados à motivação para criação de sistemas abertos. Ainda assim, é possível conceber sistemas proprietários projetados sob os mesmos requisitos.

Cada um dos atributos citados é válido se comparado a um aspecto do sistema. Por exemplo, um sistema pode apresentar interoperabilidade em relação aos equipamentos de campo, mas pode não apresentar interoperabilidade entre os diversos aplicativos que implementam as IHM's. Isso significaria que embora os equipamentos de campo de fabricantes diferentes possam trocar informações e comandos, uma ferramenta de configuração em particular não consegue trocar informações nem comandos com outras ferramentas de outros fabricantes.

Yong et al. (2004) afirma que protocolos homogêneos garantem a interconectividade entre diferentes dispositivos de diferentes fabricantes; entretanto, a interoperabilidade só é garantida de acordo com duas necessidades: em primeiro lugar, a estrutura lógica padronizada da camada de aplicação (sétima camada) do modelo de referência ISO/OSI e, em segundo lugar, a DD. Este trabalho enfoca-se no estudo do atributo interoperabilidade (no sentido da arquitetura de *software*), baseando-se no aspecto de um sistema configurador de qualquer fabricante de redes de campo poder interagir e configurar dispositivos de qualquer fabricante, segundo a necessidade da DD especificada por Yong et al. (2004).

Neste trabalho, é proposta uma tecnologia de descrição de dispositivos que vai além dos sistemas abertos proprietários. Toma-se como base uma tecnologia aberta e não-proprietária difundida em computadores pessoais, que pode ser estendida para os sistemas de automação. A tecnologia aberta e não-proprietária base usada neste trabalho é a XML, que é mundialmente conhecida pela integração de aplicativos internet.

1.9 Justificativas

A tecnologia atual padrão de descrição de dispositivos (EDD) para o protocolo FF é independente de protocolo e de plataforma (portável). Entretanto, essa tecnologia apresenta algumas desvantagens: é proprietária (YONG et al., 2004), de complexo desenvolvimento para o fabricante de dispositivo⁷ e de complexa integração para o fabricante de aplicativo de IHM (configurador, por exemplo). É necessária a aquisição de um compilador para o desenvolvimento de dispositivos e de um interpretador para o desenvolvimento de aplicativos de IHM. Além disso, essa tecnologia é de alto custo, pois a *Fieldbus Foundation*TM

⁷ O processo de geração e de interpretação da DD FF de forma detalhada é mostrado no capítulo 4, no tópico “EDDL”.

comercializa o compilador e o interpretador, segundo Carlson (2005), pelas quantias de 14.250,00 dólares e de 45.000,00 dólares, respectivamente, no plano de não membros da fundação, que é tipicamente a condição de Universidades em razão do alto custo para se tornar membro da mesma.

O alto custo inviabiliza pesquisas que envolvem o protocolo FF. Assim, existe a forte necessidade de uma descrição de dispositivos alternativa não-proprietária, aberta em relação à arquitetura e que pode ser implementada e distribuída de forma livre para uso em Universidades ou Institutos de pesquisa.

Segundo Wang et al. (2004), XML é a candidata ideal para descrição de complexas estruturas de dados, sendo muito apropriada para ser base de uma tecnologia de descrição de dispositivos.

As razões para uma tecnologia baseada em XML ser a escolhida para substituir a EDD FF:

1) A tecnologia XML foi criada para interoperar sistemas heterogêneos (de diferentes tecnologias ou ambientes) (THIRUVATHUKAL, 2004);

2) A tecnologia XML é aberta, não-proprietária, de fácil desenvolvimento (existem vários editores disponíveis) e de fácil interpretação (existem vários interpretadores disponíveis). Além disso, foi criada para prover extensibilidade e compatibilidade com versões antigas (ROY e RAMANUJAN, 2000);

3) Existe uma forte tendência das tecnologias de descrição de dispositivos basearem-se na tecnologia XML. Já existem algumas tecnologias de descrição de dispositivos normatizadas baseadas em XML: GSDML para dispositivos PROFINET e FDCML para dispositivos INTERBUS.

1.10 Objetivo

Em concordância com as justificativas apresentadas, o objetivo geral deste trabalho é mostrar a viabilidade de usar a tecnologia XML como base para desenvolver e implementar uma tecnologia de descrição de dispositivos aberta, não-proprietária e simples para equipamentos FF e criar uma alternativa tecnológica ao padrão EDD utilizado atualmente. Neste trabalho, essa tecnologia baseada em XML é chamada de *Open-EDD* (*Open Eletronic Device Description*). Este objetivo será atingido seguindo-se as seguintes etapas:

1) Definição de uma linguagem baseada em XML para desenvolvimento da *Open-EDD*. Essa linguagem é chamada de *Open-EDDML* (*Open Eletronic Device Description Markup Language*) e engloba apenas informações básicas relativas ao uso do configurador Syscon: as construções da DD FF de blocos e parâmetros, isto é, a lista de blocos e dos seus parâmetros (as construções chamadas de *blocks*, *variables*, *records*, e *arrays*) e de identificação do dispositivo. O restante das estruturas (*menus*, *edit displays*, *methods*, *relations*, *item arrays*, *collections*, *variable lists*, *programs*, *domains* e *response codes*) não será incluído neste estudo em razão do Syscon ainda não suportá-las;

2) Desenvolvimento de um componente de *software* que faça papel de compilador e interpretador de *Open-EDD*, funcionando de maneira análoga ao compilador e interpretador da tecnologia proprietária EDD. A parte relativa ao compilador fará a validação do arquivo de *Open-EDD* com base no formato desenvolvido em 1). A parte relativa ao interpretador fará a leitura dos dados;

3) Integração do componente obtido em 2) no configurador Syscon versão 6.1;

4) Testes de configuração no Syscon para validar a *Open-EDD*;

5) Criação de um mecanismo automático de geração de interface gráfica amigável dos dados da *Open-EDD* de qualquer dispositivo.

2 Alguns Protocolos de Comunicação e suas Tecnologias de Descrição de Dispositivos

Este capítulo apresenta brevemente alguns protocolos de comunicação industrial que possuem tecnologias de descrição de dispositivos em suas arquiteturas. Para cada protocolo apresentado, são expostas informações específicas de acordo com a descrição de dispositivos envolvida. No caso do protocolo FF, um detalhamento maior é feito em razão de este trabalho envolver diretamente o mesmo.

2.1 HART

O protocolo HART (*Highway Addressable Remote Transducer*) é considerado o primeiro protocolo de comunicação de sucesso por sua grande aceitação e difusão a ser utilizado em redes de campo. Foi desenvolvido pela empresa americana Rosemount Inc., na década de 1980. Vem se tornando completamente aberto e hoje todos os direitos pertencem à *HART Communication Foundation* (HCF), que dá suporte ao protocolo e administra os futuros desenvolvimentos (HART COMMUNICATION FOUNDATION, 2006).

Um dos primeiros registros de uma tecnologia de descrição de dispositivos foi escrito por Borst (1991), na qual o mesmo referencia a DD do protocolo HART.

Os dispositivos "inteligentes" introduzidos pelo protocolo HART mudaram o paradigma do chamado gerenciamento de ativos ou de patrimônio (*Asset Management*). Com isso, o operador pode descobrir se há algo de errado com os dispositivos o mais

rápido possível no processo. Esse conceito de dispositivos "inteligentes", em que os próprios instrumentos sabem informar os dados de diagnóstico assim como as formas de manutenção (através dos métodos descritos nas DD's), chegaram ao HART com o advento da tecnologia de DD. Além disso, ela possibilitou a interoperabilidade vista de um nível mais alto, provida através dos chamados comandos universais e comuns.

Zulkifli e Schneiderheinze (2002) definiram o papel da DD relacionando o mecanismo mestre-escravo do HART: A DD é a ponte entre o mestre e o escravo. Na parte do escravo, a DD descreverá os comandos, status e dados do protocolo HART. Já na parte do mestre, a DD será interpretada e gerará a interface gráfica do usuário. O usuário usa essa interface gráfica para configurar o dispositivo, receber informações de controle e de diagnósticos.

As estruturas que compõem a DD HART são (ZULKIFLI e SCHNEIDERHEINZE, 2002):

- *Variable*: São usadas para descrever todos os dados contidos no dispositivo como configuração de parâmetros, medidas e informações do dispositivo. Podem também estar agrupadas em *arrays*, coleções (*collections*) e relações (*relations*). Podem conter ações de antes ou depois de edição dos mesmos (*pre/post edit actions*). Uma ação, por exemplo, pode chamar um método para ser executado antes ou depois da variável ser processada. A principal diferença entre essa ação e a construção do método é que uma *pre/post edit action* não deve conter um comando (*command*);

- *Command*: A construção comando é usada para descrever os comandos (universais, comuns e específicos) de um dispositivo. Os comandos podem conter três operações que especificam as ações tomadas pelo dispositivo ao receber um comando. Elas podem ser ações de leitura, escrita ou comando, que será definido pelo conteúdo

dos seus campos de dados de pedido e resposta e códigos de reposta (*response codes*) implementados no dispositivo;

- *Method*: Métodos descrevem procedimentos complexos ou de tempo crítico para ser realizado pelo mestre e escravo para garantir a apropriada operação do dispositivo. Eles consistem em uma rotina de *software*, que usa umas subrotinas chamadas "*built-ins*". Um método pode conter, por exemplo, envio de comandos para o dispositivo, fiscalização das respostas, exibição de mensagens para o operador assim que seja feita alguma escrita de variável, formas de manutenção dos dispositivos (por exemplo, métodos de calibração de instrumentos). Quando um método é usado, ele realizará um procedimento sem a interferência de outros comandos;

- *Menu*: Menu é usado para construção de árvores de menus que provêem uma interface do usuário com o dispositivo. Menus são definidos como uma lista de outros itens (*variables*, *collection members*, *methods* e outros menus). Quando uma variável aparece num item de menu, o nome da variável aparece no menu. Quando esse menu é selecionado pelo usuário, o valor da variável é apresentada ao usuário. Quando um método aparece como item de menu, o nome do método aparece no menu como a variável. Quando o item de menu é selecionado pelo usuário, o método é executado;

- *Upload List*: É a lista de todas as variáveis (ou *collection members*) que podem ser escritas num dispositivo de uma vez só. A seqüência na lista descreve a ordem com que as variáveis são escritas no dispositivo. Isso é importante no caso de dependências entre variáveis: por exemplo, escrever unidades antes de valores de variáveis;

- *Identification*: A construção *manufacturer* é usada para identificar a DD com o objetivo de determinar a identificação do fabricante (*manufacturer identification*) como tipo de dispositivo (*device type*), revisão do dispositivo (*device revision*) e revisão de DD (*DD revision*). Isso dá informação para o mestre saber qual DD deve ser usada na

interação de cada dispositivo escravo configurado, já que cada dispositivo (escravo ou mestre) possui uma DD correspondente.

2.2 FOUNDATION Fieldbus™

O protocolo FF foi originalmente concebido em 1994 por uma associação internacional de fabricantes de sistemas de controle como um conjunto de normas compatibilizadoras para a comunicação de chão de fábrica com uso destinado ao controle de processos típicos de indústrias de natureza contínua, aquelas que, diferente das indústrias de manufatura, têm a produção ininterrupta como, por exemplo, as indústrias petroquímicas, alimentícias, farmacêuticas, de papel e celulose, etc. Em seu estado atual, além de compor a norma internacional IEC 61158, a qual foi adotada nacionalmente pela ABNT, também figura na norma européia CELENEC EN 50170 (FELSER e SAUTER, 2002).

FF é um protocolo de comunicação determinístico, bidirecional, digital e multiponto baseado no modelo de referência ISO/OSI de sete camadas para protocolos de comunicação digitais (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 1994).

Por se restringir ao uso local em plantas e processos industriais, assim como todos os protocolos fieldbus definidos na norma IEC 61158, o protocolo FF é compactado em apenas três camadas de rede: camada física (número 1), de enlace (número 2) e de aplicação (número 7), além de uma camada do usuário, que é baseada em processos de aplicação de blocos funcionais e situada hierarquicamente acima das camadas de rede que compõem o chamado “*stack*” (pilha) de comunicação, como apresentado na figura 21.

Figura 21 - Representação da estrutura em camadas do protocolo FF.

O fluxo de mensagens presente em um canal ou “link” de comunicação FF (ou ainda barramento) visa atender à demanda por troca de informações entre os dispositivos ativos na rede – informações provenientes tanto das estruturas das camadas de rede como das estruturas e funções da camada do usuário.

Existem três *profiles*, ou tipos, definidos na norma IEC 61784 para o protocolo FF; são eles: H1, H2 e HSE (*High Speed Ethernet*).

O primeiro tipo, denominado H1, é destinado ao uso nos instrumentos de campo – o nível localizado na base da pirâmide representada na figura 1. A camada física deste tipo prevê uma taxa de comunicação de 31.25 Kbits/s *half-duplex* (tipo de comunicação bidirecional na qual não é possível o envio e recebimento simultâneos de mensagens) em par trançado blindado com codificação binária do tipo *Manchester* (figura 22). Os instrumentos de campo nesse tipo são alimentados pelo barramento e adequados ao uso em áreas classificadas (ambientes com risco de explosão). A comunicação no H1 é controlada por um mecanismo de controle de acesso ao meio físico (MAC – *medium*

access control) determinístico descrito na norma IEC 61158 como camada de enlace tipo 1.

Figura 22 - *Codificação Manchester.*
Fonte: Brandão (2000).

O FF do tipo H2 é um tipo de rede em desuso, que foi substituído pelo tipo HSE, não existindo produtos compatíveis com tal especificação no mercado.

O tipo denominado HSE é destinado ao uso no nível dos controladores de processo e de estações de configuração e de monitoramento de processos – o nível intermediário da pirâmide representada na figura 1.

A comunicação em uma rede HSE baseia-se no protocolo *ethernet* de camada física com taxa de comunicação de 100 Mbits/s entre dispositivos não alimentados pelo barramento. O mecanismo de controle de acesso ao meio físico é baseado nos protocolos IP (camada de rede) e TCP/UDP (camada de transporte).

Nas redes HSE, classificam-se quatro tipos de dispositivos:

- *Linking Device*: é um tipo de dispositivo que estabelece a comunicação entre a rede HSE e um ou mais canais H1;
- *Ethernet Field Device*: dispositivo com camada de aplicação de blocos funcionais que se conecta diretamente à rede HSE;
- *I/O Gateway*: dispositivo que disponibiliza acesso a equipamentos baseados em outras tecnologias de campo não FF;

- *Host*: é o dispositivo por onde o operador acessa a rede, uma estação de trabalho com sistema de IHM.

Os tipos de rede H1 e HSE são compatíveis e projetados de forma a operarem complementarmente, pois compartilham a mesma camada do usuário baseada em processos de aplicação de blocos funcionais (FBAP – *Function Block Application Process*). Uma rede HSE pode ser configurada com redundância tanto de barramento como de dispositivos, porém seu protocolo de acesso ao meio físico baseado em CSMA/CD não é imune a colisões de mensagens e, conseqüentemente, não pode ser considerado determinístico (JASPERNEITE e NEUMANN, 2001).

Um sistema FF pode ser configurado fisicamente em diversas topologias de rede (VERHAPPEN e PEREIRA, 2002), tanto em um único (*single link*) como em múltiplos *links* (*bridged networks*) H1. No segundo caso, os *links* H1 são interconectados por redes HSE através de *linking devices*, como indicado na figura 23. *Linking Devices* são dispositivos que conectam logicamente dois ou mais canais de comunicação em uma mesma rede ou redes distintas.

Figura 23 - Topologia (a) single link e (b) bridged network.

O acesso ao meio físico no protocolo FF do “*profile*” H1 é realizado por algoritmos da camada de enlace e é dividido em duas formas distintas, nas quais cada uma caracteriza uma banda, ou fase, de comunicação. As duas bandas de comunicação são complementares e compartilham um único barramento: uma banda de comunicação é destinada à transmissão de mensagens determinísticas e periódicas ou cíclicas de prioridade alta; uma segunda banda de comunicação, de prioridade inferior, é reservada à transmissão de mensagens aperiódicas ou acíclicas, incluindo-se mensagens de manutenção e gerência das estruturas do protocolo. Essas fases, ou bandas, denominam-se fase cíclica e fase acíclica, respectivamente.

Figura 24 - *Fases cíclica e acíclica em macrociclos.*
Fonte: Raja e Noubir (1993).

As bandas de comunicação citadas alternam-se no barramento de dados de forma contínua e repetitiva, originando ciclos, como o ilustrado na figura 24. Esses ciclos, de período constante, são denominados macrociclos ou ciclos de comunicação. O período do macrociclo do FF em aplicações industriais típicas está entre alguns milissegundos e alguns segundos, e assim como em outros sistemas *fieldbus* utilizados para controle de processos em tempo real, está profundamente atrelado à taxa de controle das malhas distribuídas em operação na rede.

No FF, existe o conceito do nó LAS, que é o mestre da rede. Esse mestre não é fixo, isto é, qualquer dispositivo configurado para ser um LAS pode assumir assim que o LAS corrente sair da rede. O LAS é o responsável por agendar os serviços cíclicos e acíclicos na rede. Os serviços cíclicos são aqueles responsáveis pelo controle propriamente dito, colocando em prática o escalonamento proposto pela ferramenta configuradora como, por exemplo, ordem de execução dos blocos e *links* e período de ciclo total. Os serviços acíclicos podem ser, por exemplo, escritas e leituras nos parâmetros dos blocos funcionais dos dispositivos.

Apesar deste trabalho enfocar-se na DD FF, não faz parte do escopo do trabalho aprofundar-se na especificação do protocolo. Para mais informações relacionadas ao protocolo de comunicação FF, pode-se consultar Brandão (2000 e 2005), assim como as próprias normas FF.

2.2.1 Blocos FOUNDATION FieldbusTM

O detalhamento dos blocos e de alguns parâmetros é realizado neste tópico com o objetivo de demonstrar os tipos de informações descritas pela DD FF.

Todo dispositivo de campo FF possui blocos; eles são divididos em blocos transdutores, de recurso e funcionais como mostra a figura 25.

A norma define blocos funcionais como elementos de *software* que modelam algoritmos paramétricos, os quais transformam parâmetros de entrada em parâmetros de saída (FIELDBUS FOUNDATION, 1999a).

Figura 25 - Componentes da arquitetura do processo de aplicação baseado em blocos funcionais.

Fonte: Fieldbus Foundation (1999b).

Blocos Funcionais Básicos	Blocos Funcionais Avançados	Blocos Funcionais de Múltiplo I/O	Bloco Funcional Flexível
AI - Entrada analógica	DC - Controle de dispositivo	MDI - Entradas digitais múltiplas	FFB – Bloco Funcional Flexível
DI - Entrada discreta	OS - Divisor de saída	MDO - Saídas digitais múltiplas	
AO - Saída analógica	SC - Caracterizador de sinais	MAI - Entradas analógicas múltiplas	
DO - Saída discreta	LL - Lead lag (compensador)	MAO - Saídas analógicas múltiplas	
ML - Carregador manual	DT – Deadtime (tempo morto)		
BG - Bias / Ganho	IT - Integrador (Totalizador)		

CS – Seletor de controle	SPG - Gerador de rampa de referência
PD - Controlador proporcional derivativo	ISEL - Seletor de entradas
PID - Controlador proporcional integral derivativo	AR - Aritmético
RA - Controle de relação	TMR – <i>Timer</i> (temporizador) AALM - Alarme analógico

Figura 26 - *Conjuntos de blocos funcionais normalizados.*

Como já citado, segundo Yong et al. (2004), protocolos homogêneos garantem a interconectividade entre diferentes dispositivos de diferentes fabricantes; entretanto, a interoperabilidade só é garantida de acordo com duas necessidades: em primeiro lugar, a estrutura lógica padronizada da camada de aplicação (sétima camada) do modelo ISO/OSI; em segundo lugar, a DD. Yong et al. (2004) ainda complementam que essa estrutura padronizada seria o conjunto de blocos funcionais no protocolo FF. Os blocos funcionais provêm uma estrutura universal que define as interfaces externas entre os dispositivos para que seja realizado o controle distribuído.

Os blocos transdutores são blocos que ficam entre os blocos de I/O e o hardware de I/O. São, portanto, encarregados de transformar sinais físicos em variáveis e vice-versa. Eles isolam os blocos funcionais dos dispositivos e hardware específicos de I/O como sensores, atuadores e chaves de cada dispositivo. Seus algoritmos internos controlam os dispositivos de I/O e apresentam uma interface padronizada para os blocos

funcionais, realizam também funções como calibração, filtragem de sinais e conversão de dados (BRANDÃO, 2005).

Bloco de recurso é um bloco que descreve as características de *hardware* do dispositivo nos seus parâmetros de configuração. Tal qualidade desatrela os blocos funcionais das características de *hardware* dos dispositivos. O algoritmo do bloco de recurso é utilizado para monitorar o estado de operação do *hardware* e indicar possíveis alarmes nesse aspecto. Esse bloco não possui parâmetros de entrada ou de saída. Tal execução segue regras que são definidas pelo fabricante (BRANDÃO, 2005).

Os blocos possuem parâmetros. Estes parâmetros ou variáveis, que controlam e regulam a operação de um bloco, são caracterizados por:

- Nome ou mnemônico;

- Tipo de dado: a norma FF especifica um conjunto de tipos padrão de dados indexados para parâmetros de blocos funcionais. Cada parâmetro pode ser do tipo variável simples ou do tipo estrutura de dado, nos casos em que reúne duas ou mais variáveis simples (FIELDBUS FOUNDATION, 1999b). O quadro da figura 27 lista os tipos de dados da norma:

Índice	Tipo	Nome
1	Variável simples	Booleano
2	Variável simples	Inteiro (8)
3	Variável simples	Inteiro (16)
4	Variável simples	Inteiro (32)
5	Variável simples	<i>Unsigned</i> (8)
6	Variável simples	<i>Unsigned</i> (16)
7	Variável simples	<i>Unsigned</i> (32)
8	Variável simples	Ponto flutuante
9	Variável simples	<i>String</i> visível
10	Variável simples	<i>String</i> de octetos

11	Variável simples	Data
12	Variável simples	Horário do dia
13	Variável simples	Diferença de horário
14	Variável simples	<i>String</i> de bits
21	Variável simples	Valor de tempo
64	Estrutura	Bloco
65	Estrutura	Valor & Status – Ponto flutuante
66	Estrutura	Valor & Status – Discreto
67	Estrutura	Valor & Status – <i>String</i> de bits
68	Estrutura	Escala
69	Estrutura	Modo
70	Estrutura	Permissão de Acesso
71	Estrutura	Alarme – Ponto Flutuante
72	Estrutura	Alarme – Discreto
73	Estrutura	Evento – Atualização
74	Estrutura	Alarme – Sumário
75	Estrutura	Alerta – Analógico
76	Estrutura	Alerta – Discreto
77	Estrutura	Alerta – Atualização
78	Estrutura	Tendência – Ponto flutuante
79	Estrutura	Tendência – Discreto
80	Estrutura	Tendência – <i>String</i> de bits
81	Estrutura	FB <i>Link</i>
82	Estrutura	Simulação – Ponto flutuante
83	Estrutura	Simulação – Discreto
84	Estrutura	Simulação – <i>String</i> de bits
85	Estrutura	Teste
86	Estrutura	Ação – Instanciação e apagamento

Figura 27 - *Tipos de dados FF.*
 Fonte: Fieldbus Foundation (1999b).

- Tipo de uso: parâmetros de entrada, de saída ou internos;
- Modo de armazenamento: dinâmico, estático e não-volátil;
- Tamanho, em bits;
- Faixa válida: conjunto de possíveis valores que o parâmetro pode assumir;
- Valor inicial;
- Direção: no caso de parâmetros de entrada e de saída, a direção indica se o

parâmetro é enviado em *links* no sentido direto ou contrário ao fluxo à malha de controle;

- Unidade, de engenharia;
- Permissão: regras que definem quais dispositivos externos podem modificar cada parâmetro;
- Modo: modos de operação do bloco funcional nos quais o parâmetro pode ser modificado.

Parâmetros dinâmicos são aqueles calculados pelo algoritmo do bloco, logo não precisam ser recuperados após uma falha por queda de energia, por exemplo.

Parâmetros estáticos são os dados configurados que precisam ser salvos em memória não-volátil no caso de uma falha por queda de energia. Normalmente, os parâmetros estáticos são periodicamente verificados pelo configurador e eventualmente atualizados pelo mesmo. Para auxiliar na tarefa de monitorar os parâmetros estáticos, um parâmetro numérico denominado *ST_REV* (revisão estática) é incrementado sempre que um parâmetro estático for modificado.

Os parâmetros não voláteis são aqueles constantemente atualizados e que devem ser salvos em razão do risco de queda de energia do equipamento.

Vinculado aos parâmetros de entrada e saída, há um byte de *status* que se encarrega de indicar a qualidade do valor da variável (figura 28). Esse *status* é

composto por três partes: qualidade, *sub-status* e limite. A informação de qualidade pode ser:

- *Good (Cascade)*: quando o dado é confiável e pode ser utilizado em malhas de controle do tipo cascata, aquelas nas quais o *setpoint* é remoto;
- *Good (Non-Cascade)*: quando a qualidade do dado é boa, porém o bloco não pode fazer parte de uma estrutura cascata;
- *Uncertain*: neste caso, a qualidade do dado é duvidosa; pode ter ocorrido erro na aquisição ou em cálculos;
- *Bad*: quando o dado não é válido e não deve ser utilizado.

Figura 28 - Composição do byte de status.

O *sub-status* é uma informação complementar à da qualidade. É utilizada em alarmes e no início ou finalização de malhas de controle de tipo cascata. O limite informa se o valor é limitado ou não, ou seja, se esse foi restringido por extrapolar valores pré-determinados.

A execução do algoritmo do bloco depende de vários fatores; entre eles, de eventos externos e dos valores e *status* dos parâmetros de entrada e de configuração. Em função dessas variáveis, o bloco pode executar seu algoritmo de diferentes modos. Cada modo de execução do bloco impõe características ao algoritmo que influem nas saídas e nos *status*. A seguir estão descritos os oito modos possíveis de operação dos blocos funcionais:

- *Out of Service (OOS)*: o algoritmo do bloco não é executado;

- *Initialization Manual* (Iman): o algoritmo não é executado e as saídas do bloco seguem um parâmetro externo vindo do bloco que está ligado em sequência no fluxo direto;

- *Local Override* (LO): as saídas do bloco de controle seguem um parâmetro externo. Este modo é executado em situações de falha;

- *Manual* (Man): As saídas do bloco não são calculadas, o operador é quem as configura;

- *Automatic* (Auto): o algoritmo é executado normalmente e está calculando as saídas;

- *Cascade* (Cas): o *setpoint* vem de outro bloco por um *link*; desta forma o operador não pode atuar diretamente sobre ele;

- *Remote Cascade* (Rcas): o *setpoint* vem por um *link* de outra aplicação de controle remota em um dispositivo de interface;

- *Remote-Output* (Rout): as saídas do bloco funcional são modificadas por uma outra aplicação de controle remota em um dispositivo de interface.

O parâmetro *MODE_BLK* indica e controla o modo de operação dos blocos funcionais. Este parâmetro é do tipo estrutura, composto por quatro elementos:

- *Target*: é o modo de operação almejado. Normalmente o *target* é definido pelo operador;

- *Actual*: é o modo em que o bloco está sendo executado efetivamente.

- *Permitted*: são os possíveis modos de operação que o bloco pode apresentar;

- *Normal*: lista os modos normais de operação do bloco dentre aqueles listados no *permitted*. Serve para indicar ao operador o momento da seleção do *target*.

2.3 PROFIBUS

O protocolo PROFIBUS foi desenvolvido através de um projeto em conjunto de várias companhias e institutos europeus (dentre elas Siemens, Bosch, e Klockner-Moeller) na década de 1980. A primeira vez que o PROFIBUS foi anunciado no mercado foi em 1989, e no mesmo ano a PNO (*PROFIBUS Nutzer Organization*) (PROFIBUS INTERNATIONAL, 2006) foi criada na Alemanha. Tornou-se padrão DIN19245 em 1989, EN50170 em 1996 e IEC61158 em 2000 (MOTOYOSHI, 2002).

O protocolo PROFIBUS é baseado em passagem de *tokens* e é dividido em duas classes de dispositivos: mestres e escravos. Os dispositivos mestres determinam a comunicação dos dados no barramento. Os dispositivos escravos são periféricos; como exemplo tem-se os dispositivos de entrada e saída, os posicionadores, os transmissores, etc.

Nas diversas versões dos protocolos PROFIBUS, são usadas três tecnologias de descrição de dispositivos: GSD (*General Station Description*), EDD (a mesma tecnologia usada nos protocolos FF e HART, usando EDDL) e FDT/DTM (*Field Device Tool/Device Type Manager*).

Os dispositivos dos protocolos PROFIBUS possuem serviços cíclicos e acíclicos. Os serviços cíclicos são relativos à comunicação do mestre-escravo, enquanto que os serviços acíclicos são usados para alarmes e eventos de diagnóstico, *download* de configuração, parametrização (escrita e leitura de parâmetros), entre outros. São descrições de serviços cíclicos: a temporização e configuração da comunicação do mestre-escravo, descrição das entradas e saídas para o mestre da rede controlar os escravos e taxa de transmissão dos dados. São descrições dos serviços acíclicos: agrupamentos dos parâmetros nas janelas de IHM, lista de parâmetros (incluindo de

entrada e saída) e definição das formas de manutenção como gráficos e métodos. A descrição dos serviços cíclicos e acíclicos é realizada por tecnologias de descrição: a parte cíclica é feita por arquivos GSD, e a acíclica pela tecnologia EDD (EDDL) ou usando a tecnologia FDT/DTM, isto é, as duas últimas tecnologias podem seguir o mesmo propósito na descrição dos dispositivos. Entretanto, alguns pesquisadores da tecnologia FDT/DTM realizaram estudos conciliando-se as duas tecnologias, afirmando que são complementares, assunto que será abordado no capítulo 4, no tópico “FDT/DTM”.

O tipo de PROFIBUS denominado PROFINET é destinado ao uso no nível dos controladores de processo e de estações de configuração e de monitoramento de processos – o nível intermediário da pirâmide representada na figura 1.

PROFINET é um tipo do protocolo PROFIBUS que possui o mesmo meio físico da *ethernet*. A comunicação baseia-se no protocolo *ethernet* de camada física com taxa de comunicação de 100 Mbits/s entre dispositivos não alimentados pelo barramento. O mecanismo de controle de acesso ao meio físico é baseado nos protocolos IP (camada de rede) e TCP/UDP (camada de transporte) (KASEMANN, 2005).

Segundo Kasemann (2005), o PROFINET foi concebido para suportar entradas e saídas distribuídas entre até outros barramentos como, por exemplo, INTERBUS. Tal suporte à entradas e saídas distribuídas levou ao desenvolvimento de uma descrição de dispositivos mais poderosa. A descrição de dispositivos PROFINET é feita usando a tecnologia baseada em XML chamada GSDML, tida como uma evolução da tecnologia GSD (PROFIBUS INTERNATIONAL, 2005b). Mais detalhes do GSDML serão vistos no capítulo 4, no tópico “GSDML”.

2.4 INTERBUS

O protocolo alemão INTERBUS teve seu desenvolvimento iniciado na década de 1980, e atualmente a INTERBUS Club (INTERBUS CLUB, 2005) tem os direitos sobre o protocolo e administra sua padronização.

O protocolo INTERBUS foi projetado para ser um barramento de alta velocidade para transmissão de dados de processo em ambientes industriais. Devido seu procedimento de transmissão, o INTERBUS oferece rápida comunicação de dados, comunicação cíclica com temporização eqüidistante (determinístico), diagnósticos para identificar faixas de tempo ociosas, fácil operação e instalação, e suporte à fibra ótica (JASPERNEITE, 2005).

Semelhantemente ao PROFIBUS, o INTERBUS trabalha com o modelo mestre/escravo, embora o protocolo INTERBUS opere com o método da soma de pacotes (*summation frame method*), que usa apenas um pacote de rede para enviar informações para todos os dispositivos (JASPERNEITE, 2005).

A tecnologia de descrição usada no INTERBUS é a FDCML (*Field Device Configuration Markup Language*), que será abordada no capítulo 4, no tópico “FDCML”.

3 A Tecnologia XML

Para melhor entendimento deste trabalho, será feito no tópico seguinte um detalhamento da tecnologia XML, abordando assim seu histórico, fundamentos e, por fim, suas características. Tais características promovem a XML a ser uma tecnologia que permite a interoperabilidade dos sistemas usando uma arquitetura aberta e não-proprietária.

Em seguida, será abordada brevemente a tecnologia sucessora do OPC, o chamado OPC UA, que tem como fundamento em seu mecanismo de funcionamento o uso da tecnologia XML. Como já visto no capítulo 1, o configurador Syscon usa um servidor OPC para comunicação padrão com dispositivos de campo que acessa as informações da DD para poder interagir com os mesmos. Dessa forma, o OPC UA também é considerado uma tecnologia que permite a interoperabilidade de forma complementar a DD, e se baseia na tecnologia XML.

3.1 Histórico, Fundamentos e Características

A *eXtensible Markup Language* (XML) é uma tecnologia criada inicialmente com o propósito de trocar informações via internet ou em sistemas distribuídos. Foi criada pela *World Wide Web Consortium* (W3C) para o desenvolvimento da *World Wide Web* (WWW). Sua especificação 1.0 foi criada em 1998, adquirindo grande popularidade entre os profissionais da área de tecnologia da informação (THIRUVATHUKAL, 2004). Embora o *World Wide Web Consortium* (W3C) tenha fixado a especificação da

XML há tão pouco tempo, essa tecnologia já é firmemente empregada nos mecanismos não só de troca de informações via internet, mas também em outros variados setores: medicina, química, biologia, matemática, ensino e formação, publicidade, comunicação social, artes gráficas, bibliotecas e arquivos, etc.

XML é uma metalinguagem, isto é, uma linguagem usada para descrever outras linguagens. Existem atualmente várias linguagens derivadas da XML; pode-se citar a MathML usada na matemática, e a SOAP, usada para troca de informações em sistemas distribuídos.

XML é baseada em linguagens de marcadores, que seriam separadores de conteúdo. A sintaxe de um marcador é sempre precedida de um sinal de menor (<) e é finalizado com um sinal de maior (>). Podemos citar como uma linguagem de marcadores mundialmente conhecida a HTML (*Hypertext Markup Language*). A figura 29 mostra um exemplo bem simples de HTML e seus marcadores fixos como “HTML”, “HEAD”, “TITLE”, “BODY”, “H1”, etc.

```
<HTML>
<HEAD>
<TITLE>Exemplo de HTML simples</TITLE>
</HEAD>
<BODY>
<H1>Este é o primeiro nível do cabeçalho</H1>
Bem-vindo ao mundo HTML!
Este é o primeiro parágrafo. <p>
Este é o segundo parágrafo. <p>
</BODY>
</HTML>
```

Figura 29 - *Exemplo de código fonte HTML.*

A diferença básica do XML e HTML é que os *tags* ou marcadores do HTML são fixos, definidos pela própria linguagem HTML, enquanto que no XML os marcadores são especificados assim que seja colocada uma regra de negócio, isto é, criando-se uma nova linguagem baseada na tecnologia em XML com qualquer conjunto de marcadores

possível. Na figura 30, é mostrado um formato bem simples baseado em XML criado para notas fiscais; os *tags* ou marcadores foram definidos na criação da linguagem baseada em XML.

```
<NOTAFISCAL>
<NOME>João Mossin</NOME>
<ENDERECO>Rua das Flores, 200
</ENDERECO>
<VALOR>200,00</VALOR>
<MOEDA>Real</MOEDA>
<VENCIMENTO>12.05.2007</VENCIMENTO>
</NOTAFISCAL>
```

Figura 30 - *Exemplo simples de um arquivo XML.*

XML é uma tecnologia simples, aberta e não-proprietária (licença não-restritiva para uso ou distribuição). Simples porque se baseia em arquivos textos estruturados; aberta porque abre caminho para diferentes sistemas se comunicarem em uma linguagem comum independente da plataforma (portável); e não-proprietária porque livra o usuário de compiladores/interpretadores de informações proprietários.

Anterior à criação da XML, foi criada a SGML (*Standard Generalized Markup Language*), que se tornou padrão (ISO 8879) em 1986. XML, assim como o HTML, é um subsistema da SGML, que é um padrão para descrição de documentos eletrônicos estruturados. A complexidade da SGML não permitiu o vasto uso e o conhecimento mundial da tecnologia; então foi criada uma versão simplificada, a chamada XML, que reúne a capacidade de descrição estruturada, flexibilidade nas aplicações e facilidade de desenvolvimento (ROY e RAMANUJAN, 2000).

Assim como a XML, a SGML é uma linguagem de definição de outras linguagens de descrição de documentos. Documentos SGML foram usados durante muitos anos na aviação militar americana (HEITLINGER, 2001).

Segundo Roy e Ramanujan (2000), SGML é flexível por permitir a definição de novos marcadores, mas não é simples. HTML, por sua vez, é simples, mas não é flexível por ter seus marcadores fixos. XML consegue reunir essas duas características: simplicidade e flexibilidade nas aplicações, o que faz dela uma tecnologia poderosa e muito usada em aplicações de *software*.

Outra vantagem da XML em relação à HTML é o isolamento do conteúdo da parte da interface gráfica. Enquanto HTML possui a parte da interface gráfica misturada ao conteúdo, XML possui apenas o conteúdo. HTML foca a visualização dos dados para o usuário, enquanto que XML foca o conteúdo de dados estruturado.

As vantagens da XML são:

- *Tecnologia aberta e independente de plataforma*: Tecnologia é baseada em arquivo texto, não existindo dependência do sistema operacional ou alguma tecnologia em específico;

- *Tecnologia não-proprietária*: Não necessita de licença de uso ou distribuição;

- *Sistema de validação de sintaxe*: A tecnologia possui um mecanismo de validação da sintaxe de documentos XML assim que esteja definido um formato dado como correto em um DTD (*Document Type Definition*) ou *XML Schema*. O documento XML é considerado bem formatado se obedecer a todas as regras impostas pelo formato estabelecido;

- *Linguagem de marcadores extensível e flexível*: A extensão de um formato definido (por DTD ou *XML Schema*) é possível pela inclusão de marcadores novos, sempre mantendo a compatibilidade com as versões anteriores do suposto formato inicial;

- *Fácil implementação e interpretação*: Tecnologia consagrada e popularizada pela fácil construção e utilização;

- *Suporte garantido por empresas líderes do mercado do software:* As grandes empresas investiram muito nesta tecnologia e hoje oferecem cada vez mais facilidades para implementação e interpretação da XML;
- *Suporte ao padrão unicode:* codificação pode ser feita usando os caracteres da maioria das línguas (idiomas) escritas no mundo (UNICODE, 2005);
- *Documento com suporte a múltiplos idiomas:* Um documento XML pode conter a mesma descrição em múltiplos idiomas;
- *Geração automática de visualização HTML (ou qualquer outro formato) usando XSLT:* Trabalho poupado para o desenvolvedor de sistemas, além de prover interface gráfica padronizada;
- *Imensa variedade de editores XML disponíveis:* Dentre eles existem alguns gratuitos.

O uso de DTD ou XML *Schema* possibilita uma validação dos dados do documento XML de acordo com as regras estabelecidas, possibilitando assim a função de compiladores que validam cada XML a ser lido numa aplicação. Assim como o DTD, XML *Schema* consegue descrever a estrutura de um documento XML. Dessa forma, uma aplicação pode verificar se dados recebidos de qualquer parte são válidos e grupos independentes podem concordar em usar um mesmo DTD ou XML *Schema* para troca de dados, criando assim uma linguagem em comum para trocarem informações com um parâmetro de validação.

O propósito do DTD é definir os blocos de construção válidos em um documento XML. Essa construção é feita definindo-se a lista de *tags* numa estrutura permitida. O DTD pode ser declarado internamente (dentro de um documento XML) como mostra a figura 31 ou externamente (em um outro arquivo de extensão dtd) como mostra a figura 32.

```

<?xml version="1.0" ?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)
]>

<note>
<to>Paulo</to>
<from>Jani</from>
<heading>Lembrete</heading>
<body>Não me esqueça nesse fim de
semana</body>
</note><?xml version="1.0" ?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)

```

Figura 31 - Definição de DTD num arquivo XML.

Figura 32 - Definição de DTD fora do arquivo XML.

O uso de DTD é motivado principalmente por cada arquivo XML poder carregar uma descrição do seu próprio formato, uma característica atrativa para aplicações

simples ou para leigos no desenvolvimento de novas linguagens baseadas em XML, pois acopla a validação com o conteúdo XML.

A W3C desenvolveu a tecnologia específica *W3C XML Schema* em virtude da necessidade de formatos mais avançado para descrição de documentos XML e, em maio de 2001, alcançou a situação de recomendação (XML SCHEMA, 2006). Assim, o suporte à XML Schemas cresceu rapidamente. Existe atualmente uma grande variedade de *parsers* (interpretadores) que permitem validar um documento XML utilizando uma definição de XML Schema. A linguagem para definição de um XML Schema é chamada de XSDL (*XML Schema Definition Language*) e se localiza num arquivo de extensão xsd. Existem ainda outras linguagens de XML Schema, como a RELAX NG e a Schematron (WALMSLEY, 2002).

Os benefícios do XML Schema se compararmos ao DTD são:

- É criado utilizando XML, não uma sintaxe alternativa SGML; por isso são mais simples por serem construídos com uma linguagem mais amigável (figura 33);
- Suporta totalmente a recomendação *namespace*, que provê a capacidade de um documento XML ter marcadores com mesmo nome, porém únicos, em diferentes linguagens baseadas em XML (WALMSLEY, 2002);
- Permite facilmente criar modelos de conteúdo complexos e reutilizáveis;
- Permite modelar conceitos da teoria de orientação a objetos como herança de objeto e polimorfismo (pode ser chamado também de substituição de tipos);
- Permite importar declarações de outros XML Schemas com a construção *<import>*;
- Permite combinar outros XML Schemas dentro daquele já definido, permitindo extensibilidade do documento, usando a construção *<include>*.

```

<?xml version="1.0"
encoding="UTF-8"?>
<cadastro>
<titular nome="José da
Silva" RG="3022321343-3"
endereco="rua batatais"
numero="311"
nossoNumero="21331">
</titular>
<dependente nome="Márcia
Zimmerman"
parentesco="esposa">
</dependente>
<dependente nome="José da
Silva Filho"
parentesco="filho">
<bem tipo="carro" placa="ACB
2123">
</bem>
</dependente>
<dependente nome="João da
Silva" parentesco="irmão">
</dependente>
<dependente nome="José da
Silva Filho"
parentesco="filho">
</dependente>
<dependente nome="Juliana da
Silva" parentesco="filho">
</dependente>
<bem tipo="residência"

```

cadastro.xml

```

<?xml version="1.0" encoding="UTF-
8"?>
<xsd:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema
chema"
elementFormDefault="qualified">
  <xsd:element name="cadastro">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="titular"/>
        <xsd:element
maxOccurs="unbounded"
ref="dependente"/>
        <xsd:element
maxOccurs="unbounded" ref="bem"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="titular">
    <xsd:complexType>
      <xsd:attribute name="RG"
use="required" type="xsd:NMTOKEN"/>
      <xsd:attribute name="endereco"
use="required"/>
      <xsd:attribute name="nome"
use="required"/>
      <xsd:attribute
name="nossoNumero" use="required"
type="xsd:integer"/>
      <xsd:attribute name="numero"
use="required" type="xsd:integer"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="dependente">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0"
ref="bem"/>
      </xsd:sequence>
      <xsd:attribute name="nome"
use="required"/>
      <xsd:attribute
name="parentesco" use="required"
type="xsd:NCName"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="bem">
    <xsd:complexType>
      <xsd:attribute
name="endereco"/>
      <xsd:attribute name="numero"
type="xsd:integer"/>
      <xsd:attribute name="placa"/>
      <xsd:attribute name="tipo"
use="required" type="xsd:NCName"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

cadastro.xsd

Figura 33 - Exemplo de arquivo XML Schema.

A capacidade de definição de tipos complexos de variáveis é a necessidade primordial de uma tecnologia de descrição de dispositivos. Assim, o ideal para construção do formato de uma tecnologia de descrição baseada em XML é o uso de *XML Schema*.

Existem várias maneiras de criar a interface gráfica para o usuário a partir do conteúdo de um arquivo XML. A mais automática delas, é usando a linguagem XSLT (*eXtensible Stylesheet Language Transformation*). A tecnologia XML engloba a XSLT, que é uma linguagem de interpretação dos documentos XML que permite a geração de um simples texto, de HTML ou qualquer outro formato imaginável. A XSLT é contida em um arquivo de extensão xsl e nele é feita uma lógica que percorre os dados do arquivo XML correspondente a fim de organizá-los em uma visualização desejada. A partir de um único documento XML, é possível desenvolver várias visualizações de dados, assim como mostra a figura 34.

Figura 34 - Exemplo de visualizações usando XSLT.

As vantagens acima citadas descrevem a razão de a tecnologia XML ser a chave para a interoperabilidade de sistemas heterogêneos. XML seria a linguagem em comum que os sistemas heterogêneos podem utilizar a fim de trocar informações, o mesmo papel das tecnologias de descrição, que permitem a interoperabilidade entre os dispositivos e os aplicativos usados nos sistemas de automação.

3.2 OPC e OPC-UA

O surgimento da tecnologia *OLE for Process Control* (OPC) permitiu que fabricantes de IHM pudessem desenvolver apenas um *driver* de comunicação com os dispositivos, diferentemente do modo antigo no início dos anos 90, em que cada fabricante tinha que desenvolver *drivers* proprietários (e de arquitetura fechada) para suportar seus dispositivos nas aplicações (figura 35).

O OPC é um *driver* com interface de acesso padronizada para aplicativos de IHM. Atualmente, existem implementações de servidores OPC para FF, HART, DeviceNet, PROFIBUS, entre outros (OPC FOUNDATION, 2006).

Em 1995, um grupo de empresas formou o *OPC Task Force*, cujo objetivo era definir uma especificação padrão para o OPC, baseado na tecnologia COM da Microsoft.

A tecnologia COM é usada para componentização de partes de aplicativos, que provê a disponibilidade de serviços para qualquer outra aplicação desenvolvida numa linguagem (Delphi, C++, Visual Basic e outras) que suporte COM.

Figura 35 - *Um driver de comunicação para cada fabricante.*

O sistema operacional Windows, que provê essa tecnologia, é responsável por fazer as ligações, conversões de tipos e os procedimentos que forem necessários para os componentes, escritos em diferentes ferramentas, se comunicarem. Tais componentes podem ser bibliotecas (de extensão dll) ou arquivos executáveis (de extensão exe). A diferença desses arquivos para os executáveis e bibliotecas convencionais é que os componentes COM são construídos com uma interface comum, o que garante compatibilidade binária, necessária para a comunicação entre os diversos componentes.

Os aplicativos de IHM são clientes OPC que possuem um modo padrão de acesso a qualquer servidor OPC (figura 36) de qualquer protocolo de comunicação industrial.

Em 1996, foi lançada a especificação OPC 1.0 e definida a criação da *OPC Foundation*.

Figura 36 - *Modelo de comunicação padrão com clientes e servidores OPC.*

A padronização dos *drivers* foi feita por um grupo de fabricantes junto a *OPC Foundation* (OPC FOUNDATION, 2006). Esses fabricantes contribuíram com idéias que refletiram em soluções robustas e mais completas, sempre com base na experiência dos mesmos em relação ao desenvolvimento de *drivers*.

O sucessor do OPC é o OPC UA (*Unified Architecture*), que está sendo desenvolvido pela *OPC Foundation* mais um grupo de empresas (dentre elas, Microsoft, Smar, Emerson, Rockwell, Siemens, etc) desde 2003 com o objetivo de integrar as várias especificações OPC existentes: OPC DA (*Data Access*), OPC AE (*Alarms and Events*), OPC HDA (*Historical Data Access*), OPC DX (*Data Exchange*), etc. Essa arquitetura tem um banco de dados integrado com as variações de OPC já citadas, isto é, o banco de dados que era distribuído passou a ser integrado, o que facilita o tratamento para os aplicativos (THE INDUSTRIAL ETHERNET BOOK, 2006).

Além da unificação do OPC, o OPC UA agrega um novo atributo relacionado a sistemas abertos: portabilidade. Isso significa que os clientes e servidores OPC podem ser implementados em qualquer tecnologia e em qualquer plataforma eliminando-se

assim a obrigatoriedade de serem implementados usando-se a tecnologia COM da Microsoft e de serem executados no sistema operacional Windows. A especificação XML DA foi a primeira tentativa de permitir a interoperabilidade dos sistemas através de troca de informações usando dados em XML, mas foi com a especificação OPC UA que o OPC passou a ter uma arquitetura mais aberta, além de interoperável também passa a agregar portabilidade.

A chave para a interoperabilidade do OPC UA é a arquitetura orientada a serviços, providos através dos chamados *Web Services*.

Web Services são serviços que transportam dados em XML entre as aplicações, podendo também trafegar na internet. Essa tecnologia é provida pela união das tecnologias SOAP (baseada em XML), HTTP e TCP/IP.

Outras características interessantes do OPC UA segundo Prizon e Duarte (2005):

- Projetado para internet, porém com otimizações para permitir eficiência no transporte dos dados;
- Integração desde o chão-de-fábrica até o sistema corporativo - ERP (*Enterprise Resource Planning*) e MES (*Manufacturing Execution System*);
- Incorporação de um modelo de informação bastante sofisticado, podendo suportar as várias descrições de dados disponíveis como EDD FF, EDD PROFIBUS, etc. Isso permitirá um tratamento bastante adequado da informação pelos aplicativos de IHM, que até então se limitavam a dados de operação, mas não tinham acesso fácil aos dados de configuração e manutenção.

A *OPC Foundation* está realizando um trabalho em conjunto com a FF, HCF e PNO para estender as funcionalidades da EDD dentro do OPC UA (NEIL, 2005). Isso provê total integração da EDD com o OPC UA para descrever os dados para os aplicativos de IHM, isto é, clientes OPC teriam acesso a dados complexos do

dispositivo de forma automática para os protocolos FF, HART e PROFIBUS. Isso permite uma facilitação no desenvolvimento dos aplicativos de IHM, que possibilita uma melhora na qualidade do produto para o usuário final.

4 Revisão Bibliográfica

A revisão bibliográfica deste trabalho apresenta de forma detalhada as informações encontradas na literatura sobre as tecnologias de descrição dos protocolos de comunicação brevemente apresentados no capítulo 2. Além das tecnologias de descrição normatizadas, são abordadas algumas tecnologias que são ou foram pesquisadas. Dentre essas tecnologias, já normatizadas ou não, algumas tomam a XML como base.

4.1 EDDL

EDDL é uma linguagem baseada em texto para descrição de características de comunicação de dispositivos inteligentes. Essa descrição envolve características do equipamento, dados de diagnóstico e detalhes de configuração. É também uma tecnologia independente de plataforma e de protocolo de comunicação para descrição de interface gráfica homem-máquina (ZIELINSKY, 2005).

Inicialmente chamada de DDL (*Device Description Language*), tornou-se significativa tecnologia em muitas indústrias nas áreas de telecomunicações, *mainframes*, aplicações de computadores, instrumentação e controle (ZIELINSKY, 2005). A linguagem DDL gerava a chamada DD, enquanto que a nova linguagem EDDL gera a agora chamada EDD.

A tecnologia HART foi pioneira no desenvolvimento da DDL em 1992 (ZIELINSKY, 2005). Posteriormente, a DDL foi usada como base para o

desenvolvimento de linguagens de descrição de dispositivos para os protocolos FF (1996) e PROFIBUS (2000), como mostrado na figura 37.

Em março de 2004, foi aprovada como uma linguagem padrão IEC 61804-2 e passou a ser chamada de EDDL (*Electronic Device Description Language*) ao invés de simplesmente DDL. O trabalho de padronização (IEC) reuniu as características das linguagens de descrição de dispositivos FF, HART e PROFIBUS, consolidando assim uma única linguagem usada para descrição de dispositivos desses três protocolos.

Figura 37 - *Evolução da EDDL ao longo dos protocolos de comunicação.*
 Fonte: Instrumentation, Systems, and Automation Society (2005).

Segundo Zielinsky (2005), as vantagens da EDDL são:

- *Portabilidade:* Pode ser usado em qualquer configurador de qualquer sistema operacional;
- *Consistência e uniformidade:* Como a EDDL suporta a descrição de múltiplos protocolos, o mesmo configurador forneceria a mesma interface gráfica dos três protocolos FF, HART e PROFIBUS de todos os dispositivos, isto é, fica o mais transparente possível a operação para o usuário;
- *Apenas um configurador para diferentes protocolos:* EDDL possibilita que o desenvolvedor do configurador de estratégias de controle desenvolva apenas uma

ferramenta para os diferentes protocolos. Isso facilita muito a aquisição de dados para relatórios, documentação, etc;

- *Controle de revisão*: Cada EDD tem um número que identifica a revisão do equipamento e da EDD. Isso possibilita a compatibilidade do aplicativo de IHM com versões anteriores do equipamento;

- *Teste e registro*: Testes de validação da sintaxe e dependências, e o registro de cada EDD binária, usando o *toolkit* associado de acordo com a fundação que administra o protocolo;

- *Suporte padrão internacional*: A EDDL é um padrão internacional aprovado pelo IEC e está disponível com o número padrão 61804-2. Suporta múltiplos códigos de países e linguagens internacionais.

Em fevereiro de 2003, representantes da FF, HCF e PNO formaram um grupo de pesquisa para desenvolvimento em conjunto para extensão da IEC 61804-2 e decidiram criar recursos mais recentes relacionados com dispositivos complexos e com capacidade de alta análise para diagnósticos. Esses recursos que foram aprovados em março de 2004 englobam uma melhor organização de informações e uma visualização gráfica desses dados, assim como persistência dos dados num banco de dados. Nesse banco de dados ficariam, por exemplo, parametrizações (mudança de valores de parâmetros) de dados que seriam fundamentais para diagnosticar possíveis falhas nos dispositivos. Segundo Sasajima (2004), esses recursos seriam:

- *Interface com o usuário melhorada*: Dispositivos sofisticados podem conter centenas de parâmetros para configuração. As extensões da EDD permitem aos fornecedores melhor organizar esses parâmetros em diferentes visões (telas) como, por exemplo, de calibração, identificação, configuração e de diagnóstico (figura 38). Dispositivos como posicionadores de válvulas contam com avançados gráficos de

diagnóstico como, por exemplo, gráfico de resposta ao degrau, assinatura de válvulas (figura 39), etc;

- *Persistência de informações*: Essa nova característica possibilitaria que a EDD tivesse instruções para o *host* (*software* de IHM) de quais parâmetros e gráficos poderiam ser armazenados no banco de dados do sistema configurador assim como no próprio equipamento. Alguns dados são necessários serem coletados e armazenados no próprio equipamento, pois o atraso para leitura do configurador invalidaria os dados para uma futura análise.

The image shows a software window titled "Temperature : Form" with two tabs: "Configuration" and "Calibration". The "Configuration" tab is selected. Inside the window, there are two main sections, "Sensor 1" and "Sensor 2". Each section contains four input fields: "Sensor Type:", "Sensor Range:", "Sensor SN:", and "Sensor Connection:". The fields are arranged in a two-column layout for each sensor.

Figura 38 - Exemplo de separação das visões dos parâmetros de calibração e de configuração.

Figura 39 - Exemplo de gráfico de assinatura de válvula.

Outra característica que merece atenção especial pela sua natureza é a respeito da interface com o usuário. A EDD gerada a partir da EDDL descreve o conteúdo e não a forma da interação entre usuários e equipamentos, isto é, a tecnologia de EDDL não impõe um “*look and feel*”⁸ para cada equipamento. Essa característica à primeira vista pode parecer sem importância, porém provê uma separação fundamental de preocupações. O desenvolvedor do equipamento que cria a EDD fica livre das preocupações subjetivas típicas dos desenvolvedores de interfaces de usuário, ao mesmo tempo em que o desenvolvedor da IHM fica desobrigado de entender as especificidades inerentes a um dado tipo de equipamento. Baseando-se nessa característica, as EDD’s oferecem algumas vantagens muito interessantes tanto para os usuários finais quanto para os desenvolvedores de sistemas de *software*. O desenvolvedor de sistemas pode se preocupar apenas com o que é de sua especialidade, como desenvolver uma interface

⁸ “*Look and feel*”: Expressão da língua inglesa que significa a visão do usuário final sobre a aparência e comportamento do sistema. Isso engloba a forma de interação do sistema com o usuário envolvendo o mecanismo de mensagens gráficas, estilo gráfico, aparência dos dados na tela, dos botões, menus, etc.

gráfica amigável de interação do usuário final, enquanto que o usuário final tem uma interação natural e até intuitiva.

Uma consequência importante das vantagens da tecnologia citadas acima é que para criar EDD's os fornecedores de equipamentos não precisam ter competência no desenvolvimento de *software*. Donaires (2004) expõe alguns pontos relevantes da tecnologia EDDL:

[...] o desenvolvimento de *software* encerra alguns desafios que exigem competências especiais por parte do desenvolvedor. Uma dessas competências é a capacidade de lidar com a subjetividade do usuário no processo de definição das interfaces gráficas de usuário. Entretanto, há outras competências, entre as quais estão as competências em análise e engenharia de sistemas e em engenharia de *software*, que são sensivelmente diferentes das competências típicas que os desenvolvedores de hardware e de equipamentos detêm. Competência em desenvolvimento de *software* inclui preocupações com gerenciamento de requisitos, qualidade de *software*, configuração de *software* e versão de *software* em presença dos aspectos subjetivos já mencionados. Os fornecedores de equipamento não necessariamente detêm todas essas competências, que são caras de se adquirir e manter. Em geral tais fornecedores não têm obrigação de possuir tais competências, se estão no negócio de equipamentos e não de sistemas. Para criar uma boa DD é suficiente a competência em programação DDL e linguagem C. Esta é uma grande vantagem, porque livra o fabricante de equipamento do fardo de desenvolver sistemas e aloca essa responsabilidade a quem ela pertence por direito: ao desenvolvedor de sistema. Esta talvez seja a maior vantagem desta tecnologia, e sua característica distintiva mais essencial.

As desvantagens da EDDL são:

- *Linguagem de desenvolvimento não é amigável*: O desenvolvimento da EDDL é feito usando uma linguagem similar à linguagem C, que não é uma linguagem amigável; para leigos, o aprendizado requer tempo ou treinamento;

- *Desenvolvedor de dispositivos necessita de compilador proprietário*: Apesar de a EDDL ser padrão para descrição de dispositivos, o compilador é proprietário (YONG et al., 2004). O desenvolvedor do dispositivo FF e HART, por exemplo, necessita compilar o código fonte EDDL, usando um tokenizador ou compilador proprietário para

que seja validada a sintaxe do código fonte e para que seja gerado um arquivo binário compactado, que é lido pelo interpretador. O compilador é geralmente comercializado pela fundação ou instituto que administra as normas dos protocolos;

- *Desenvolvedor de configuradores necessita de um interpretador proprietário*: O desenvolvedor de configurador necessita acoplar em seu sistema um componente de *software* que interprete as informações geradas pelo compilador, isto é, o arquivo binário. Esse componente de *software* interpretador também é geralmente comercializado pela fundação ou instituto que dita as normas dos protocolos;

Arquivos de DD FF possuem uma grande estrutura hierárquica de informações. Existem quatorze construções que podem ser descritas na EDDL para FF: *blocks*, *variables*, *menus*, *edit displays*, *methods*, *relations*, *arrays*, *records*, *variable lists*, *programs*, *domains*, *response codes*, *item arrays*, e *collections* (FIELDBUS FOUNDATION, 1999c):

- *Blocks* descrevem as características externas de um bloco. Mais detalhes sobre a descrição da constituição dos blocos podem ser consultados na norma Fieldbus Foundation (1999c);

- *Variables*, *records*, e *arrays* descrevem os dados contidos de um equipamento;

- *Menus* e *Edit Displays* descrevem como os dados são apresentados pelo *host* para o usuário;

- *Methods* descrevem algoritmos de execução de uma sequência de comandos e interações entre o *host* e os equipamentos;

- *Relations* descrevem relações entre *variables*, *records*, e *arrays*;

- *Item Arrays* e *Collections* descrevem grupos lógicos e dados;

- *Variable Lists* descrevem grupos lógicos de dados contidos em um equipamento que podem ser comunicar em grupo;

- *Programs* especificam códigos executáveis que podem ser inicializados pelo *host*;
- *Domains* podem ser usados para descarregar ou carregar informações em batelada de um equipamento;
- *Response codes* especificam um código de resposta para as aplicações de um *variable*, *record*, *array*, *variable list*, *program*, ou *domain*.

Em uma EDD desenvolvida em EDDL existem centenas de definições de variáveis. Um pequeno exemplo de definição de uma variável simples para dar a idéia de esforço e complexidade da implementação será descrita a seguir na figura 40.

Código EDDL	Exemplo interpretação do configurador
<pre> VARIABLE temperature { LABEL "Temperature"; TYPE FLOAT { DISPLAY_FORMAT "###.##"; MIN_VALUE "-273,2"; MAX_VALUE "500,2"; } CONSTANT_UNIT "°C"; } </pre>

Figura 40 - Exemplo de código EDDL e interpretação de uma variável simples.

O Anexo A deste trabalho contém um exemplo de código fonte EDDL completo obtido na norma Fieldbus Foundation (1997), que contém exemplos de definições de *blocks*, *variables*, *menus*, *methods*, *arrays*, *records* e *response codes*.

Como já foi citado, atualmente a EDDL é usada para descrever equipamentos FF, HART e PROFIBUS. Abaixo, segue uma descrição de como o processo de desenvolvimento da DD FF é feito.

Todo fornecedor de equipamentos FF desenvolve em conjunto com o próprio equipamento a respectiva DD. O processo existente na norma FF para a geração de DD é realizado utilizando-se dois *toolkits* comercializados pela própria *Fieldbus Foundation*TM. Esses *toolkits* são: *AT-400 Device Description Tokenizer Toolkit* e *AT-401 Device Description Services Toolkit*.

Após a implementação usando-se a DDL, o desenvolvedor de dispositivos compila o código fonte utilizando-se o compilador incluso no *AT-400 Device Description Tokenizer Toolkit*.

Figura 41 - Processo de desenvolvimento da DD de um dispositivo.

O compilador gera um número único de itens da DD, chamado de *DDItemId*. Para cada item da DD (um bloco, parâmetro, etc) que é criado, o compilador atribui esse número, que é usado pelo resto da existência desse item, em qualquer que seja o dispositivo.

Além da DDL desenvolvida pelo fabricante, é compilada também uma série de arquivos padrões que formam uma biblioteca de informações em comum dos dispositivos. Algumas dessas informações são tipos de variáveis e blocos padrões definidos na norma FF.

O produto de todo esse processo é um arquivo binário especial, codificado e compactado com extensão ffo, que contém as informações da DD. Outro arquivo com extensão sym também é gerado; ele contém os nomes dos símbolos localizados no arquivo ffo. Dentre os símbolos, podem ser citados nomes de variáveis, nome dos blocos, nome dos métodos, etc. Ambos os arquivos, de extensão ffo e sym são incluídos no banco de DD's do configurador.

O desenvolvedor de aplicativos de IHM faz uso do *AT-401 Device Description Services Toolkit*. Esse *toolkit* possui um *software* interpretador chamado DDS, que tem como função fazer a interface entre o configurador e o arquivo de DD binário, através de dezenas de serviços escritos na linguagem C. Então, esse interpretador deve ser obrigatoriamente integrado no configurador para que interaja com o dispositivo desenvolvido. Pela complexidade envolvida na integração, esse *toolkit* possui uma série de manuais (FIELDBUS FOUNDATION, 1997) para guiar o desenvolvedor de aplicativos.

Esses *toolkits*, por um lado, forçam os fabricantes a usarem uma linguagem padrão para descrição de dispositivos e assim garantem a interoperabilidade de dispositivos. Por outro lado, fazem da tecnologia DD (assim como a EDD) uma tecnologia proprietária.

4.2 CF

O arquivo CF (*Capability File*), assim como a EDD, surgiu com o objetivo de interoperar diferentes dispositivos em diferentes sistemas de automação FF.

CF é um arquivo de formato multiplataforma ASCII (*American Standard Code for Information Interchange*). Para dispositivos FF do “*profile*” H1, o arquivo é de extensão CFF (*Capability File Format*), enquanto que para os do “*profile*” HSE a extensão é CFH (*Capability File for HSE*). Cada um dos tipos de CF possui um conjunto diferente de informações relativas ao *profile* envolvido.

O CF é usado para completar a descrição de dispositivos FF feita pela EDD FF desenvolvida com a tecnologia EDDL. Enquanto o foco da EDD FF são aspectos relacionados a blocos e parâmetros, o CF é responsável por descrever o equipamento como um todo, contendo, por exemplo, informações da memória usada para realizar um *link*, de tempo de execução dos blocos funcionais, de número de blocos instanciáveis e permanentes, etc. Dispositivos que não possuem blocos, tais como algumas *bridges* FF não possuem arquivo de EDD; apenas o CF é usado para descrever suas características.

O arquivo CF pode ser criado usando qualquer editor de textos como, por exemplo, o bloco de notas do sistema operacional Windows. O formato do arquivo tem que ser orientado a linhas; cada linha deve conter uma expressão relativa à cada informação do dispositivo. Se duas barras consecutivas (//) são detectadas durante a interpretação da linha, é assumido que o resto da linha é um comentário. Ainda é possível adicionar uma continuação do comando para a linha posterior, incluindo uma vírgula (,) (FIELD BUS FOUNDATION, 1999d).

```

[File Header]
Description="Resource file for Deltabar S"
FileType = CapabilitiesFile
FileDate=2001,02,06
CffVersion = 1,5

[Device Header]
DeviceName = "Deltabar S"
CommGroup = 3
CommClass = Class32
CommSubClass = Class3LinkMaster
DeviceClass = LINKMASTER

[Device VFD 1] // Management VFD
VendorName = "Endress+Hauser GmbH"
ModelName = "MIB_VFD"
Revision = "Rev 1.61"
VersionOD = 0x01
ProfileNumber = 0x4D47

[Device VFD 2] // FB VFD
VendorName = "Endress+Hauser GmbH"
ModelName = "Deltabar S"
Revision = "0.3 29.09.99"
VersionOD = 0x01
ProfileNumber = 0x0000

[NM OD Directory]

DirectoryRevisionNumber = 1
NumberOfDirectoryObjects = 1
TotalNumberOfDirectoryEntries = 8
...

```

Figura 42 - Trecho do Capability File do dispositivo Deltabar do fabricante Endress+Hauser.

4.3 FDT/DTM

A tecnologia FDT/DTM (*Field Device Tool/Device Type Managers*) surgiu de um grupo de fornecedores de sistemas que se uniram para criar uma arquitetura aberta em relação a equipamentos de campo, assim como a EDD desenvolvida com a EDDL. Ela é baseada em componentes de *software* baseados na tecnologia COM (*Component Object Model*) da Microsoft, que respeitam as interfaces para comunicação definidas pelas especificações do grupo alemão ZVEI (*German Electrical and Electronic Manufacturers Association*) (FDT GROUP, 2006).

A tecnologia FDT/DTM é usada para descrever dispositivos HART, PROFIBUS e FF. A PNO adota essa tecnologia, enquanto que a HCF e FF mostram certa resistência em adotar essa tecnologia por demonstrar claramente preferência pela tecnologia EDD, que já é padrão IEC, enquanto que o FDT/DTM tenta mostrar suas vantagens ao mercado e no futuro ser também um padrão de acordo com as necessidades mercadológicas. Mesmo assim, já existem vários componentes para HART e PROFIBUS implementados.

Não há uma *FDT Foundation*. Atualmente, quem quiser adquirir a documentação da norma FDT deve entrar em contato com a PNO, que detém os direitos de uso e distribuição da norma. Entretanto, em meados de 2002, foi formado o grupo de interesse na tecnologia FDT (*FDT Joint Interest Group* ou FDT-JIG) (FDT GROUP, 2006), um conjunto de grandes empresas do setor de controle e automação de processos no cenário mundial como E+H, Siemens, etc. Esse grupo tem concentrado esforços para promover e divulgar a tecnologia FDT/DTM, sendo seus principais objetivos:

- Obter o direito de distribuição e uso não-exclusivos da tecnologia FDT/DTM, que atualmente são de propriedade da PNO, que distribui gratuitamente a norma. O objetivo do FDT-JIG é dar ao mercado uma posição neutra com relação a qualquer protocolo de comunicação que é a idéia central por trás da tecnologia;
- Promover e difundir o uso da tecnologia FDT/DTM de forma geral, sempre mantendo uma posição de neutralidade e credibilidade para o mercado;
- Manter um endereço de internet (*web site*) com informações sobre a tecnologia e sobre as empresas interessadas, eventos e informações importantes, como uma lista com produtos compatíveis (FDT GROUP, 2006);
- Manter a documentação atualizada e disponível para todos os interessados, incorporando novas funcionalidades de acordo com as necessidades do mercado;

- Aglutinar esforços para assegurar a interoperabilidade entre os sistemas, aplicativos e componentes baseados na tecnologia FDT/DTM.

Alguns fabricantes que adotaram a tecnologia FDT/DTM tem atuação no mercado alemão como ABB, CEAG, Endress+Hauser, Invensys, Pepperl+Fuchs, Siemens, Sorting, e Yokogawa (SHEBLE, 2002). Todos esses fabricantes adotaram também a tecnologia EDDL, por ser padrão IEC.

FDT/DTM é dividido em três domínios: o domínio da ferramenta de engenharia, o domínio da *Field Device Tool* (FDT) e o domínio dos equipamentos de campo suportados por *Device Type Managers* (DTM). A ferramenta de engenharia é normalmente voltada a ser um sistema configurador. FDT é um componente de *software* que faz o papel de um recipiente (do inglês *container*) onde são especificadas todas as interfaces padronizadas para garantir a interoperabilidade e conectividade entre o sistema configurador e o DTM. FDT seria um recipiente onde ficam todos os DTM's, de cada dispositivo. O DTM é o centro da tecnologia, um componente de *software*, que é desenvolvido pelo fabricante do dispositivo e que tem que obedecer a todas as interfaces de comunicação com o FDT, assim como a especificação define. Esse componente contém todo o acesso às informações do equipamento de campo, as interfaces gráficas de usuário e toda a funcionalidade de configuração, diagnóstico e manutenção do equipamento de campo.

Existe ainda o chamado DTM de comunicação, o componente que teoricamente seria substituído para prover o suporte a multiprotocolos de um determinado DTM de um tipo de instrumento.

Figura 43 - *Arquitetura da tecnologia FDT/DTM.*
 Fonte: Donaires (2004).

As características do DTM são:

- Contém todas as mensagens e “*look and feel*” da interface com o usuário;
- Contém informações por visões (mesma divisão EDDL) do dispositivo para configuração, diagnóstico, identificação, calibração;
- Pertence ao dispositivo e é fornecido pelo fabricante de dispositivos, igualmente a todas as tecnologias de descrição de dispositivos;
- Interface gráfica com usuário com suporte a múltiplos idiomas;
- Liberdade para implementação de novos recursos, já que o DTM é um componente de *software* desenvolvido em qualquer linguagem que suporte a tecnologia COM;

- A troca de dados entre os componentes de *software* da arquitetura FDT/DTM é feita com descrições em XML;
- Tecnologia independente de protocolo.

As desvantagens do DTM são:

- Não suporta persistência dos dados, diferentemente da EDDL;
- O esforço de aprendizado para o desenvolvimento dos DTM's é grande, segundo Kastner e Kastner-Masilko (2004);
- Compilador e interpretador proprietário.

Segundo Donaires (2004), seria uma desvantagem da tecnologia FTD/DTM o fato de o DTM ser um componente de *software* que contém todas as mensagens e telas (“*look and feel*”) da interação com o usuário, já que a preocupação de desenvolvimento de um “*look and feel*” amigável seria totalmente atribuída ao desenvolvedor de equipamentos, sendo que tal habilidade seria uma competência assumida e melhor realizada sem dúvida por pessoas especializadas em desenvolvimento de *software*, não de desenvolvimento de equipamentos de campo ou PLC's. A separação da camada de negócio⁹ e da interface gráfica de interação com o usuário resultante do uso da tecnologia EDDL seria uma vantagem indiscutível, já que a EDDL abrange aspectos relativos ao dispositivo, enquanto que a parte de interação com o usuário desenvolvida por engenheiros ou analistas de sistemas seria desenvolvida com base nas regras de negócio impostas pelos engenheiros de desenvolvimento de dispositivos de controle.

⁹ Camada de negócio: Expressão usada em desenvolvimento de *software* que diz respeito à parte ou camada inteligente do sistema, que realmente processa as operações através de serviços (que tem a lógica do negócio envolvida, seja engenharia, medicina, etc). A camada de negócio é projetada de forma que possa usar qualquer interface gráfica, não causando impacto no sistema se for substituída ou modificada.

A característica de “*look and feel*” embutida em cada DTM favorece ainda que cada dispositivo integrado no configurador possa ter um estilo de interface com o usuário diferente. Isso causa extremo desconforto da parte do usuário, que tem que operar um sistema como se fosse vários, cada um com um estilo e modo de operação diferente. Assim o configurador, por exemplo, seria um *software* cujas próprias regras de estilo não seriam padronizadas, uma consequência do uso da tecnologia FDT/DTM. Por essa desvantagem já ser conhecida pelas empresas que são favoráveis a essa tecnologia, foi elaborado um guia de estilo para interface com o usuário para uniformizar a interface com o usuário.

A liberdade para implementação de novos recursos pode parecer uma vantagem à primeira instância, mas na realidade pode ser uma outra desvantagem. Para o funcionamento dos recursos, existem interfaces definidas pelas especificações (com o objetivo da padronização); assim a implementação de novos recursos seria não-padrão, já que não existem regras para incorporação dos mesmos no FDT, e assim no configurador, devido à ausência de interfaces padronizadas. Por outro lado, implementações de novos recursos dos DTM's agregariam mais valor a determinado produto, no caso o do fabricante que tiver maior competência e criatividade.

O processo de desenvolvimento de um DTM necessita de um compilador, igualmente ao processo de uma DD. Como o DTM é baseado em um componente COM, o compilador pode estar em qualquer ambiente de desenvolvimento de *software* que suporte o desenvolvimento da tecnologia COM. Pode-se citar como exemplo desses compiladores: Borland Delphi, Microsoft Visual Basic, Microsoft Visual C++, Borland C++, etc. Esses compiladores são comercializados assim como o compilador da tecnologia EDDL, embora nesse caso haja liberdade de escolha. Assim, o processo de

desenvolvimento do DTM gera o mesmo problema da tecnologia EDD: a aquisição de um compilador proprietário.

Considerando a tendência da era dos sistemas abertos, pode-se concluir que a maior desvantagem dentre as citadas é a obrigação do uso no sistema operacional Windows, já que a tecnologia COM não é suportada por outras plataformas. Pior que isso, Merrit (2002) afirma que a tecnologia FDT/DTM nem sobrevive a outras gerações dos próximos sistemas operacionais Windows, porque todos os DTM's teriam que ser reescritos pelo fato desses futuros sistemas operacionais não suportarem COM.

4.4 GSD

Sistemas configuradores PROFIBUS que obedecem a IEC 61784-1:2003 CP3/1 e CP3/2 usam um arquivo de formato específico multiplataforma ASCII chamado arquivo GSD. Esse arquivo possui informações para realizar o controle com a rede. Segundo a especificação PROFIBUS, as informações seriam (PROFIBUS INTERNATIONAL, 2005a):

- Informação necessária para identificar o dispositivo conectado;
- Descrição das informações dos dispositivos que podem ser acessadas via rede (exemplo: parâmetros configuráveis);
- Descrição das capacidades de comunicação (exemplo: taxa de transmissão);
- Informações adicionais do fabricante.

O arquivo GSD pode ser criado usando qualquer editor de textos como, por exemplo, o aplicativo bloco de notas do sistema operacional Windows. O formato do arquivo tem que ser orientado a linhas; cada linha deve conter uma expressão relativa a cada informação do dispositivo. Se um ponto e vírgula (;) é detectado durante a

interpretação da linha, é assumido que o resto da linha é um comentário. O máximo número de caracteres em uma linha é 80. Se não for possível descrever a informação em uma linha, é possível adicionar um sinal (\), que indica que a linha está continuando na linha seguinte (PROFIBUS INTERNATIONAL, 2005a).

Existem algumas diferenças na descrição do arquivo GSD de dispositivos mestres e escravos. No caso dos mestres, existem parâmetros como máximo número de escravos que podem ser conectados, opções de *download* e *upload*, etc que não são incluídas na descrição dos dispositivos escravos.

```

#Profibus_DP
GSD_Revision  = 2
Vendor_Name   = "SMAR"
Model_Name= "FY303"
Revision= "1.0"
Ident_Number= 0x0897
Protocol_Ident= 0
Station_Type= 0
Bitmap_Device = "Src0897n"
FMS_supp= 0
Hardware_Release= "3.0"
Software_Release= "1.17"
187.5_supp= 1
MaxTsdr_93.75= 1000
MaxTsdr_187.5= 1000
Redundancy= 0
Repeater_Ctrl_Sig= 0
24V_Pins= 0
Freeze_Mode_supp= 0
Sync_Mode_supp= 0
Auto_Baud_supp= 0
Set_Slave_Add_supp= 1
Min_Slave_Intervall= 250
Modular_Station= 1
Max_Module= 1
Max_Input_Len= 15
Max_Output_Len= 10
Max_Data_Len= 25
Max_Diag_Data_Len= 14
Slave_Family= 12
User_Prm_Data_Len= 0
;
;Modules  for Analog Output Block
;
Module      = "eSP "      0x82, 0x84, 0x08, 0x05
EndModule
;
Module      = " SP "      0xA4
EndModule
;
Module      = "eSP + RB + POS_D " 0xC6, 0x84, 0x86, 0x08, 0x05,
0x08, 0x05, 0x05, 0x05
EndModule
;
Module      = "eSP + CHECKBACK " 0xC3, 0x84, 0x82, 0x08, 0x05,
0x0A
EndModule
;
Module      = "eSP + RB + POS_D + CB " 0xC7, 0x84, 0x89, 0x08,
0x05, 0x08, 0x05, 0x05, 0x05, 0x0A
EndModule
;
Module      = "eRCAS_IN + RCAS_OUT "
0xC4, 0x84, 0x08, 0x05, 0x08, 0x05
EndModule
;
Module      = " RCAS_IN + RCAS_OUT " 0xB4
EndModule

```

Figura 44 - Exemplo de um arquivo GSD de um posicionador de válvulas PROFIBUS PA do fabricante Smar Equipamentos Industriais.

No caso dos dispositivos escravos, existem parâmetros como número e tipo de entradas e saídas, especificações de textos de diagnósticos e informações dos módulos disponíveis dos dispositivos modulares. Essas informações por sua vez não são incluídas na descrição dos mestres (PROFIBUS INTERNATIONAL, 2005a).

A interpretação dos dados dos arquivos GSD é feita pelo *software* configurador diretamente, sem existir a necessidade de um interpretador específico por se tratar de um arquivo texto comum que apenas segue as regras impostas pela especificação PROFIBUS.

Não é um objetivo do GSD a descrição de funções tecnológicas tais como interface gráfica ou meios de manutenção. Para esse propósito é que são usadas as tecnologias EDD e FDT/DTM (PROFIBUS INTERNATIONAL, 2005a).

4.5 GSDML

Como já mencionado, GSDML é a tecnologia de descrição usada para dispositivos PROFINET. GSDML é tida como uma evolução da tecnologia GSD pelo fato de ambas as tecnologias seguirem os mesmos objetivos; porém, enquanto a descrição do GSD é realizada no formato ASCII, a GSDML é baseada na tecnologia XML. A descrição, desse modo, pode ser feita em qualquer editor XML, desde que seja seguido o XML *Schema* definido pela especificação GSDML (PROFIBUS INTERNATIONAL, 2005b).

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!-- This example shows how to describe the structure of cyclic IO data of
an IO Device.
In this case the structure is described for the submodule with ID "IDS_2" -
->
<ISO15745Profile
xmlns="http://www.profibus.com/GSDML/2003/11/DeviceProfile"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.profibus.com/GSDML/2003/11/DeviceProfile
..\XSD\GSDML-DeviceProfile-v2.0.xsd">
  <!-- ProfileHeader Definition as defined in ISO15745-1. Please do
not change the content. -->
  <ProfileHeader>
    <ProfileIdentification>PROFINET Device
Profile</ProfileIdentification>
    <ProfileRevision>1.00</ProfileRevision>
    <ProfileName>Device Profile for PROFINET Devices</ProfileName>
    <ProfileSource>PROFIBUS Nutzerorganisation e. V.
(PNO)</ProfileSource>
    <ProfileClassID>Device</ProfileClassID>
    <ISO15745Reference>
      <ISO15745Part>4</ISO15745Part>
      <ISO15745Edition>1</ISO15745Edition>
      <ProfileTechnology>GSDML</ProfileTechnology>
    </ISO15745Reference>
  </ProfileHeader>
  <ProfileBody>
    <DeviceIdentity VendorID="0xFFFF" DeviceID="0x0000">
      <InfoText TextId="IDT_FamilyDescription"/>
      <VendorName Value="PROFIBUS International"/>
    </DeviceIdentity>
    <DeviceFunction>
      <Family MainFamily="I/O" ProductFamily="PNO GSDML
Examples"/>
    </DeviceFunction>
  </ProfileBody>
</ISO15745Profile>
...

```

Figura 45 - Trecho exemplo de descrição feita com a tecnologia GSDML de um dispositivo.

Pode-se citar, por exemplo, algumas características interessantes do GSDML herdadas de tecnologia XML (PROFIBUS INTERNATIONAL, 2005b):

- *Validação da sintaxe*: por uso de um XML Schema fornecido pela especificação GSDML;

- *Uso de múltiplos idiomas*: O uso de diferentes idiomas pode ser definido no documento GSDML de um dispositivo. Isso se reflete nas mensagens do configurador de interface com usuário;

- *Desenvolvimento da descrição GSDML não requer o uso de compilador proprietário*: Editores de texto comuns podem ser usados para implementar o

documento GSDML assim como outros editores, como os especializados em XML, que provêem fácil desenvolvimento. O papel de um compilador (validador de sintaxe) seria feito pelo editor especializado que indicaria os erros na edição de acordo com o XML Schema;

- *O configurador não necessita de interpretador proprietário:* O software configurador apenas necessita suportar uma leitura dos dados XML, que normalmente é feita com recursos já existentes nas tecnologias de desenvolvimento de *software*;

- *Fácil e amigável desenvolvimento da descrição e do interpretador que dispensa treinamento:* Herdada da tecnologia XML;

- *Descrição multiplataforma:* A descrição baseada em XML pode ser usada em qualquer plataforma, como Linux, Windows, Solaris, etc;

- *Fácil desenvolvimento de um gerador de interface automática em HTML no configurador:* Possível pela tecnologia inclusa no XML chamada XSLT, como já mencionado anteriormente.

```
<ChannelDiagItem ErrorType="19">
  <Text TextId="ID_COMM_ERROR"/>
</ChannelDiagItem>
<ExternalTextList>
  <PrimaryLanguage>
    < Text TextId="ID_COMM_ERROR" Value = "Communication error"/>
  </PrimaryLanguage>
  <Language xml:lang="de">
    < Text TextId="ID_COMM_ERROR" Value = "Kommunikationsfehler"/>
  </Language>
  <Language xml:lang="fr">
    < Text TextId="ID_COMM_ERROR" Value = "Erreur de
communication"/>
  </Language>
</ExternalTextList>
```

Figura 46 - *Definição de um erro em múltiplos idiomas num documento GSDML.*
Fonte: Profibus International (2005b).

4.6 FDCML

A tecnologia de descrição usada no INTERBUS é a FDCML (*Field Device Configuration Markup Language*), que se assemelha muito com a tecnologia GSDML por ser baseada na tecnologia XML.

As vantagens da tecnologia FDCML são (FDCML, 2002):

- *Independência de protocolo de comunicação*: Esta tecnologia é capaz de descrever componentes de rede de qualquer protocolo de comunicação;
- *Suporte a múltiplos idiomas*: Herdada de tecnologia XML;
- *Extensibilidade*: Esta tecnologia provê a capacidade de agregar informações que não estão previamente obedecendo as regras imposta pelo schema através dos marcadores `<additionalItem>`, `<specificProperty>`, `<externalSchema>` e `<nonStandardizedExtension>`.

A independência de protocolo de comunicação e a extensibilidade fornecida pela tecnologia XML possibilitam a descrição de dispositivos de outros protocolos como, por exemplo, PROFINET. A extensibilidade permite a inclusão de qualquer elemento de rede de qualquer protocolo de comunicação. Segundo Heines (2005), existem dispositivos PROFINET e Sercos que já são descritos usando FDCML além de dispositivos INTERBUS.

A especificação FDCML foi criada essencialmente para configuração de dados para serviços de comunicação para controle, isto é, serviços cíclicos. A integração de dispositivos HART ou FF, que lidam com outra natureza de informação, seria muito trabalhosa e causaria desconforto ao desenvolvedor por ter que lidar com objetos que não fazem parte do seu domínio de conhecimento. A tecnologia EDDL consegue descrever os dispositivos HART, PROFIBUS e FF por os mesmos se tratarem de similares descrições como menus, gráficos,

parametrização, métodos, etc, que são relacionados com os serviços acíclicos de comunicação dos dispositivos.

Para dispositivos INTERBUS, o FDCML possui um editor especializado não-proprietário (figura 47) que provê uma fácil e rápida construção de um documento FDCML. Com esse editor, o engenheiro não necessita de treinamento; o desenvolvimento é um simples preenchimento de formulário.

Device Description Editor

File Extra Help

Features Technical Data Warnings Tools Help Files LED

Base Data Communication Pictures Interfaces Channels Device Functions Functions

FDCML Language: english (us) **File Revision:** 0

Vendor: Fabricante Hipotético **Url:**

Vendor ID: 000321

Remarks:

Device type: sensor **Product family:** 23

Product designation: **Url:**

Order number: qw22212111

HW Rev.: 5 **FW Rev.:** 2.56 **App.**

Comm. entities: 1

Functional description (short)

Sensor de Pressão

Functional description (long)

Sensor de Pressão para controle de processo industrial

System: INTERBUS Device: File: v5.xml

Figura 47 - Editor FDCML para dispositivos INTERBUS.
Fonte: FDCML (2005).

O uso desse editor livra o desenvolvedor de dispositivos de conhecer a tecnologia XML, pois após a edição o editor gera automaticamente um documento FDCML, que segue as regras do XML *Schema*, já incluído no *software*. Além disso, o editor inclui um processador de XSLT, que é contido num arquivo de extensão xsl usado para transformar o conteúdo descrito em XML num documento HTML. Isso permite a visualização dos dados de forma muito mais amigável e também possibilita uma documentação, isto é, além das funcionalidades conhecidas, a documentação é automatizada e facilmente incluída no configurador ou em qualquer outro *software*.

Module Data:

IB IL 24 DI 8-PAC

Inline digital input terminal block, complete with accessories (plug connector and labeling field)
24 V DC, 4-wire connection system

[General Data/Programming Data](#)
[Important Notes](#)
[Function/Features](#)
[Module Interfaces](#)
[Current Supply/Consumption](#)
[Technical Data](#)
[Internal Circuit Diagram](#)
[Connection Example](#)

▲ **General Data**

General Product Information	
Manufacturer:	Phoenix Contact
Device type:	I/O digital - DI
Product range:	IL
Order No.	2861247

Programming Data	
ID code	190
Process data channel	8 bit
Parameter channel	0 byte
Input register length	8 bit

Figura 48 - *HTML gerado a partir de um documento FDCML.*
 Fonte: FDCML (2005).

4.7 Tecnologias de Descrição de Dispositivos Pesquisadas

Simon et al. (2001) e Simon et al. (2003) consideram que as tecnologias de descrição EDDL, FDT/DTM, GSD, etc são tecnologias complementares, e então propuseram uma tecnologia de descrição que seria um DTM que interpretaria código EDDL, GSD ou qualquer outra tecnologia de descrição de dispositivos que seja baseada em ASCII, e assim proveria automaticamente a interface com o usuário do configurador provendo automaticamente janelas, menus, gráficos, etc. Simon et al. (2001) ainda justificam a idéia argumentando que o uso de interpretadores de ASCII reduz o número de componentes de *software* COM que aumenta a estabilidade do sistema FDT como um todo. A desvantagem dessa tecnologia seria a mesma do DTM, por ser baseado na tecnologia COM. A vantagem seria uma interface gráfica padronizada.

Kastner e Kastner-Masilko (2004) propuseram uma tecnologia de descrição que seria uma simples mistura de EDDL com FDT/DTM, uma pesquisa mais focada e menos abrangente (pois se limita a interpretar apenas EDDL) se compararmos com a de Simon et al. (2001).

Segundo Johnston (2006), a FF oferece uma integração fraca via DD/CF, e que uma alternativa bem mais vantajosa do ponto de vista de integração de protocolos seria o DTM. Tal integração de protocolos permite o uso de apenas um configurador para interagir com diferentes dispositivos de diferentes protocolos. Ainda segundo Johnston (2006), as melhorias da EDD no que diz respeito à interface melhorada com o usuário (figura 38 e 39) e persistência de informações, serão utilizadas para aumentar o poder de integração dos DTM's, desde que a seja proposta uma arquitetura que contemple a EDD e o DTM.

Wollschlaeger (1999) pesquisou uma tecnologia de descrição de dispositivos para o sistema CANOpen baseada em XML, frisando no estudo a vantagem da tecnologia XML ser aberta.

Wang et al. (2004) pesquisaram uma tecnologia de descrição de dispositivos FF baseada em XML chamada por eles de XDDL. Nesse estudo, foi usado o DTD para definição das construções e validação da sintaxe. A estrutura do documento pesquisada por Wang et al. (2004) é mostrada na figura 49.

Figura 49 - Estrutura do DTD definido por Wang et al (2004).

O nó raiz do documento XML definido por Wang et al. (2004) seria chamado de *XDDLDeviceDesc*. Esse nó teria dois elementos “filhos”, *XDDLDocumentInfo* e *DeviceDescription*. O nó *XDDLDocumentInfo* tem a informação da data criada (*XDDLDocCreateDate*) e revisão do documento (*XDDLDocRevision*). O nó

DeviceDescription contém as informações referentes à descrição do dispositivo. A identidade do dispositivo é definida pelo elemento *DeviceIdentify* através dos atributos *ManufacturerID* e *DeviceType*, que descrevem respectivamente o código do fabricante e do tipo de dispositivo. O elemento *BlockDescriptions* contém as informações sobre os blocos FF, usando lista de blocos representada pelo elemento *BLOCK* e cada bloco teria um atributo *BlockType* que descreve o tipo de bloco (*PID*, *Resource*, *Transducer*, etc) e. O elemento chamado *BLOCKPARAM* descreve a lista de parâmetros de cada bloco. Os parâmetros ainda são divididos em *S_PARAM*, *V_PARAM* e *A_PARAM*. O elemento *V_PARAM* é usado para descrever variáveis simples, e *S_PARAM* para descrever parâmetros chamados *records*, que definem estruturas. O elemento *A_PARAM* é usado para descrever *arrays* ou listas.

Em virtude da necessidade de descrições em XML de tipos mais avançados de estruturas de descrições de dispositivos e, pela situação de recomendação do XML Schema (XML SCHEMA, 2006), o uso de DTD não seria o ideal para a elaboração de uma linguagem de descrição de dispositivos, e sim o XML Schema. Em razão disso, este trabalho (*Open-EDD*) não seguiu a pesquisa de Wang et al. (2004). Além disso, Wang et al. (2004) apenas definiu o formato em DTD (linguagem), não criando a tecnologia de descrição como um todo, como é proposto neste trabalho.

A maioria das grandes plantas de processo industrial necessita de diferentes tecnologias de redes de campo para atender a todos os processos industriais como, por exemplo, PROFIBUS, HART, FF, etc. Cada uma dessas redes de campo necessita de um *software* configurador diferente, assim como outros tipos de *software*. Segundo Sheble (2002), fica desvantajoso ou até mesmo impraticável nesse contexto o uso de diferentes configuradores que levam o usuário muitas vezes a erros por ter que usar frequentemente funções de exportação e importação de dados para unificar as

informações a fim de produzir relatórios, documentos, consistência dos dados de configuração, base de dados integradas, histórico de troca e manutenção de todos os dispositivos, etc. Sheble (2002) ainda ressalta que uma descrição independente de protocolo de comunicação proveria ao *software* configurador manter a comunicação com diferentes protocolos e assim manter as informações unificadas no domínio de apenas uma ferramenta. Isso pouparia o usuário de realizar as tentativas de unificação de informações manualmente (que pode conduzir a erros), além de fornecer uma interface de IHM segura, amigável e padronizada.

A tecnologia de descrição proposta neste trabalho também vem complementar a idéia de Sheble (2002) em relação à independência de protocolo. A característica de extensibilidade herdada da tecnologia XML propicia a descrição baseada em XML ter o potencial de agregar informações de outros protocolos, assim que os mesmo vão sendo estudados e utilizados.

Levando-se em consideração a consolidação das construções de descrições para os dispositivos dos protocolos HART, FF e PROFIBUS da EDDL (IEC 61804-2) e a capacidade de extensão da tecnologia XML, a *Open-EDDML* também tem o potencial de suportar descrições para dispositivos para esses mesmos protocolos.

5 Metodologia

A metodologia da criação da *Open-EDDML*, que compõe o objetivo número 1, foi cumprida elaborando-se o XML *Schema*, que é representado graficamente na figura 50. O código fonte completo em XSDL desse XML *Schema* em modo texto pode ser visto no Apêndice A deste trabalho.

De acordo com figura 50, o formato definido tem como elemento raiz o chamado *DeviceDescription*. Esse elemento tem dois elementos “filhos”: *Identification*, que contém dados de identificação do dispositivo e *BlockList*, que contém a lista de blocos. O elemento *BlockList* contém uma lista de *Block* (Bloco). Cada elemento *Block* tem uma *ParameterList* (lista de parâmetros), que por sua vez tem uma lista de elementos *Parameter* (parâmetros). Cada elemento *Parameter* possui uma lista de enumerações (*EnumerationList*), ou de bitenumerações (*BitEnumerationList*) e ou de membros (*MemberList*). Um membro (*Member*) pode ter uma lista de enumerações (*EnumerationList*) ou de bitenumerações (*BitEnumerationList*). Cada um dos elementos da *Open-EDDML* tem atributos, como nome, tamanho, etc.

Figura 50 - Representação gráfica do XML Schema da Open-EDD.

A metodologia de desenvolvimento do compilador e interpretador de *Open-EDD*, que compõe o objetivo número 2, foi cumprida implementando-se um componente de *software* na linguagem C++ no ambiente Borland C++ 5.02. Para operações de escrita e

leitura em XML, num ambiente que usa C++, foi acoplada nesse componente uma biblioteca de serviços (de escrita e interpretação XML) chamada de Xerces. A versão do *parser* Xerces utilizada foi a 2.4.0.0, obtida gratuitamente para uso e distribuição no site da Apache Software Foundation (2005).

A compilação ou validação dos dados é uma funcionalidade inerente ao *parser* Xerces C++. A cada requisição de leitura de um arquivo *Open-EDDML*, é requisitada uma validação no momento de execução.

Figura 51 - *Interpretador de Open-EDD com Xerces C++.*

Para o desenvolvimento do interpretador, foi usada a chamada técnica de desenvolvimento usando modelagem orientada a objetos. Segundo Booch (1994), o desenvolvimento de sistemas complexos computacionais e de alto nível tem um melhor gerenciamento e modelagem com o uso da tecnologia de orientação a objetos (OO) e com a técnica e linguagem de modelagem dos objetos (UML). Para desenvolvimento de sistemas complexos usando-se orientação a objetos e para linguagem de modelagem dos mesmos, Booch (1994) e Booch et al. (1999) podem ser respectivamente consultados.

A UML é o resultado da unificação das linguagens de modelagem de objetos de três métodos: *Booch*, *Object Modeling Technique* (OMT) e *Object-Oriented Software Engineering* (OOSE). Em 1997, a primeira versão da UML foi adotada de OMG - *Object Management Group* - e desde então tornou-se referência mundial no desenvolvimento de sistemas modelados com objetos.

Segundo Booch et al. (2000):

[...] a visão tradicional no desenvolvimento de *software* adota a perspectiva de um algoritmo. Nessa visão, o principal bloco de construção do *software* é o procedimento ou função. Essa perspectiva conduz os desenvolvedores a voltar seu foco de atenção para questões referentes ao controle e a decomposição de algoritmos menores. Não existe nenhuma desvantagem nessa solução, com exceção da tendência a permitir sistemas instáveis. À medida que os requisitos do sistema se modificam (e isso certamente ocorrerá) e o sistema cresce (o que também ocorrerá), será difícil realizar a manutenção de sistemas construídos a partir do foco em algoritmos.

A visão contemporânea no desenvolvimento de *software* adota uma perspectiva orientada a objetos. Nessa visão, o principal foco de construção de todos os sistemas de *software* é o objeto ou classe. Explicando de uma maneira simples, um objeto é alguma coisa geralmente estruturada a partir do vocabulário do domínio do problema ou do domínio da solução; uma classe é a descrição de um conjunto de objetos comuns. Todos os objetos têm uma identidade (é possível atribuir-lhes nomes ou diferenciá-los dos demais objetos de alguma maneira), um estado (costuma haver dados a eles associados) e um comportamento (é possível fazer algo com o objeto ou ele poderá fazer algo com outros objetos).

A tecnologia OO facilita ao máximo o encapsulamento e abstração dos serviços de *software* (nesse caso do interpretador). Foi usada a ferramenta de modelagem UML Rational Rose 98 para criar diagramas de classes para compor esse encapsulamento e abstração. No primeiro diagrama criado, foi definida uma classe base chamada *XMLElement*, que tem todas as funcionalidades básicas de escrita, leitura, validação dos dados e formato, etc, providas pelo Xerces. Para cada elemento ou marcador *Open-EDDML*, foi definida uma classe que herda as funcionalidades dessa classe base, como representado na notação UML na figura 52. Pelo fato de o elemento

ParameterXMLElem ter as mesmas funcionalidades do elemento *MemberXMLElem*, mas com serviços adicionais, o mesmo é herdado do elemento *MemberXMLElem*.

Figura 52 - Representação das heranças de classe dos elementos *Open-EDDML* na notação UML.

Com base no formato do XML *Schema* definido, foi criado outro diagrama de classes para representar as relações de associação entre as classes que representam os elementos *Open-EDDML*, como mostrado na figura 53.

Figura 53 - Representação das relações de associação entre os elementos Open-EDDML na notação UML.

Além de favorecer a modelagem dos dados num alto nível, a abstração e encapsulamento das funcionalidades, a técnica de orientação a objetos usa a linguagem

UML para representação dos objetos (classes). Um benefício do uso dessa técnica e linguagem é a documentação nesse nível (diagrama) e também a geração de código C++ automático, provido pela ferramenta Rational Rose.

A metodologia de integração do interpretador no Syscon, que compõe o objetivo número 3, foi cumprida substituindo-se a parte relativa ao interpretador proprietário DDS pelo interpretador de *Open-EDDML* desenvolvido, para fazer a leitura dos arquivos de descrição de dispositivos. Com isso, o Syscon e o servidor de DD tiveram que ser modificados para poder interagir com o interpretador *Open-EDDML* como mostrado na figura 54.

Figura 54 - *Arquitetura de software do Syscon modificada.*

No banco de *Open-EDD's*, cada diretório dos dispositivos do *Device Support* teve os arquivos de extensão ffo e sym (arquivos gerados pelo compilador proprietário) substituídos por um único arquivo de extensão XML, como mostrado na figura 55.

Figura 55 - Banco de Open-EDD.

A metodologia dos testes, que compõe o objetivo número 4, foi feita sequencialmente da seguinte maneira no Syscon, como se estivesse usando a DD proprietária:

- Criação dos arquivos de *Open-EDD* baseado em DD's FF reais do fabricante Smar, dos dispositivos DFI302, TT302 e FY302, usando-se o editor de XML gratuito do ambiente Netbeans, versão 5.5 Beta 2 obtido em Netbeans (2006);

- Integração desses arquivos no banco de *Open-EDD*;

- Criação no Syscon desses dispositivos (tela da figura 11);

- Criação no Syscon dos blocos de recurso (*Resource*) e transdutor (*Transducer*) de todos os dispositivos, do bloco AI (*Analog Input*) e PID do TT302 e o AO (*Analog Output*) do FY302 (tela da figura 12);

- Criação da estratégia de controle AI-PID-AO, a mesma mostrada na figura 8;

- Parametrização de valores *offline* nos blocos funcionais (tela da figura 14);

- Descarregamento de configuração (*download*) para os dispositivos (tela da figura 15);

- Leitura e escrita de valores *online* em dispositivos de campo (tela da figura 16).

Simon et al. (2002) afirma que dados de dispositivos descritos em XML podem ser representados graficamente em HTML através de XSLT. O maior benefício desse mecanismo é uma descrição única, reusável e com excelente concistência, o que gera um esforço reduzido para desenvolvimento de telas. A metodologia de criação de um mecanismo de geração automática de interface gráfica amigável usando a *Open-EDD*, que compõe o objetivo número 5, foi cumprida implementando-se uma lógica usando a linguagem XSLT para geração de interface gráfica automática em HTML. Essa lógica ficou contida no arquivo chamado “OpenEDD.xsl”, que é associado a qualquer arquivo de *Open-EDDML* para representar as informações num formato amigável. Complementar ao “OpenEDD.xsl”, foi criado um arquivo de estilo chamando de “OpenEDD.css”, que contém o estilo HTML do documento, definindo assim, por exemplo, tipo e tamanho de fontes (letras), cores, textos sublinhados ou em itálico, etc.

6 Resultados

Como resultado do objetivo número 1, foi criado um documento *Open-EDDML* de um dispositivo hipotético, que está em modo texto no Apêndice C deste trabalho. A implementação desse arquivo foi feita de forma simples e fácil usando-se o editor de XML do ambiente Netbeans (NETBEANS, 2006), que inclusive valida a sintaxe com base no XML *Schema* desenvolvido.

A metodologia relativa ao objetivo número 2 proveu uma fácil integração do interpretador de *Open-EDDML* no configurador Syscon. O algoritmo implementado no configurador para chamar as funções do interpretador foi feito num nível bem alto, tal que os elementos *Open-EDDML* são representados por objetos. Um trecho desse algoritmo de obtenção dos itens descritos da *Open-EDDML* na linguagem C++ pode ser mostrado na figura 56. Os serviços de obtenção de informações e de elementos de cada classe ou objeto estão descritos na figura 57.

```

// Criação do elemento raiz dada uma Open-EDDML
//
DeviceDescriptionXMLElem ddXMLElem ( "0401.xml" );

// Obtenção do elemento identificationXMLElem
//
IdentificationXMLElem* idenXMLElem = ddXMLElem.GetIdentificationXMLElem();

// Obtenção do elemento blockListXMLElem
//
BlockListXMLElem* blkListXMLElem = ddXMLElem.GetBlockListXMLElem();

// Iteração de todos os elementos blockXMLElem dada uma blockListXMLElem
//
for (BlockXMLElemIterator blockXMLElemIterator(*blkListXMLElem);
blockXMLElemIterator; blockXMLElemIterator++)
{
    BlockXMLElem* blockXMLElem = blockXMLElemIterator.Current();

    int ddItemId = blockXMLElem->GetId ();
    char blkLabel[128];

    strcpy ( blkLabel, blockXMLElem->GetTag () );

    ParameterListXMLElem* paramListXMLElem = blockXMLElem-
>GetParamterListXMLElem();

    int numberOfParams = paramListXMLElem->GetNumberOfParameters();

    // Iteração de todos os elementos paramXMLElem dada uma paramListXMLElem
    //
    for (ParameterXMLElemIterator parameterXMLElemIterator(*paramListXMLElem);
parameterXMLElemIterator; parameterXMLElemIterator++)
    {
        ParameterXMLElem* parameterXMLElem = parameterXMLElemIterator.Current();

        int paramDDItemId = parameterXMLElem->GetId ();

        ...
    }
}

```

Figura 56 - Trecho de código na linguagem C++ para leitura de Open-EDDML.

Classe	Métodos ou Serviços
DeviceDescriptionXMLElem	int GetVersion() IdentificationXMLElem* GetIdentificationXMLElem () BlockListXMLElem* GetBlockListXMLElem ()
IdentificationXMLElem	int GetManufacturerId () int GetDeviceTypeId () int GetDeviceRevision () int GetDDRRevision () char* GetManufacturerName () char* GetDeviceTypeName ()

BlockListXMLElem	int GetNumberOfBlocks ()
BlockXMLElem	char* GetTag () char* GetName () int GetId () ParameterListXMLElem* GetParamterListXMLElem ()
ParameterListXMLElem	int GetNumberOfParameters()
ParameterXMLElem	MemberListXMLElem* GetMemberListXMLElem () <i>// Herdados da classe base MemberXMLElem:</i> char* GetName () char GetMetaType () int GetId () int GetType () int GetSize () int GetParameterClass () int GetHandling () char* GetDisplayFormat () BitEnumerationListXMLElem* GetBitEnumeratedListXMLElem () EnumerationListXMLElem* GetEnumeratedListXMLElem ()
MemberListXMLElem	int GetNumberOfElements ()
MemberXMLElem	char* GetName () char GetMetaType () int GetId () int GetType () int GetSize () int GetParameterClass () int GetHandling () char* GetDisplayFormat () BitEnumerationListXMLElem* GetBitEnumeratedListXMLElem () EnumerationListXMLElem* GetEnumeratedListXMLElem ()
EnumerationListXMLElem	

BitEnumerationListXMLElem	
---------------------------	--

EnumerationXMLElem	int GetValue () char* GetDescription ()
--------------------	--

BitEnumerationXMLElem	int GetBit ()
	char* GetDescription ()

Figura 57 - *Serviços providos pelas classes do interpretador de Open-EDD.*

Para cumprimento geral dos objetivos de número 1, 2, 3 e 4 foram feitos testes de configuração no Syscon. Depois de concluídos os testes, o Syscon foi capaz de fazer todas as operações de configurações nos dispositivos com a mesma eficácia da implementação usando a tecnologia proprietária, isto é, os objetivos 1, 2, 3 e 4 foram alcançados com sucesso.

O objetivo número 5 também foi concluído com sucesso. A tela de visualização amigável de *Open-EDD* no formato HTML, gerada após processamento XSLT, é mostrada na figura 58.

Device Description View

Endress+Hauser Cerabar S

- Identification

Manufacturer Name	Device Type Name	Manufacturer Id	Device Type Id	Device Revision	DD Revision
Endress+Hauser	Cerabar S	0x452B48	0x1007	0x2	0x1

- Blocks List

Block Tag	Block Name	Id	Number of Parameters
<div><div></div>Analog Input</div>	_ANALOG_INPUT_BLOCK	0x800201D0	36
<div><div></div>PID Control</div>	_PROPORTIONAL_INTEGRAL_DERIVATIVE_BLOCK	0x800202B0	65
<div><div></div>TBPR</div>	_PRESSURE_CAL_BASIC	0x80020630	87
<div><div></div>Resource Block 2</div>	_RESOURCE_BLOCK_2	0x80020AF0	42

- Parameter List

Analog Input

Parameter Name	Id	MetaType	Parameter Class	Handling	Size (bytes)	Type	Number Of Elements	Enumerations
ST_REV	0x8002017A	Variable	0x8000	Read	2	Unsigned	4	
TAG_DESC	0x80020180	Variable	0x8000	Read/Write	32	OctetString		
STRATEGY	0x8002017E	Variable	0x8000	Read/Write	2	Unsigned		
ALERT_KEY	0x80020037	Variable	0x8000	Read/Write	1	Unsigned		
MODE_BLK	0x80020126	Record	0x8000	Read/Write	1	BitEnumerated		
-TARGET			0x8000					
-ACTUAL			0x8002					

Figura 58 - *Parte inicial da interface em HTML gerada pelo processamento da linguagem XSLT da Open-EDDML do dispositivo Cerabar S da Endress+Hauser.*

O código fonte desenvolvido em XSLT (arquivo “OpenEDD.xsl”) para processamento e conversão do XML em HTML está no Apêndice B deste trabalho. O código fonte do arquivo complementar de estilo (arquivo “OpenEDD.css”), que define o estilo HTML, está no Apêndice D.

7 Conclusões

A tecnologia OO e a modelagem UML usada no desenvolvimento de sistemas complexos têm uma associação ou mapeamento natural com os elementos descritos em XML, em especial ao formato especificado no XML *Schema*. Cada elemento XML pode ser representado por uma classe, e cada atributo de um elemento XML pode ser representado por um atributo de classe.

A tecnologia *Open-EDD* se mostrou mais simples de ser manipulada se for comparada à EDD. Desenvolvedores de dispositivos necessitam conhecer apenas a tecnologia XML, caso não haja um editor específico. No caso de desenvolvedores de aplicativos de IHM, o código de integração do interpretador no aplicativo seria num nível mais alto (usando objetos), se comparado ao do DDS da EDD, que usa funções na linguagem C.

Este trabalho mostrou que é possível a migração da EDD (proprietária) desenvolvida com EDDL para a *Open-EDD*, apesar de apenas englobar as estruturas de listas de blocos e parâmetros, isto é, as informações básicas que o configurador Syscon versão 6.1 necessita para configurar os dispositivos FF. Entretanto, a EDD tem uma vantagem indiscutível se comparada a qualquer nova tecnologia de descrição: existem quatro fundações (FF, HCF, PNO e OPC) dando apoio ao desenvolvimento da mesma e hoje já suporta muito mais recursos de quando foi criada (há mais de 13 anos), pois além das estruturas ou construções não abordadas neste trabalho, a EDDL também é usada para descrição de dispositivos HART e PROFIBUS. A adoção de outra tecnologia como a *Open-EDD* pode trazer todos os recursos da EDD com muitas vantagens como,

por exemplo, o fato de ser não-proprietária, mas é necessário que haja um grande grupo de desenvolvimento, padronização e tempo de amadurecimento para que alcance tudo o que a EDD pode oferecer atualmente. Em razão disso, este trabalho sugere que a *Open-EDD* seja uma tecnologia alternativa e não substitutiva em relação à EDD, tornando-se assim, uma alternativa viável para Universidades.

Assim, este trabalho possibilita a pesquisa do protocolo FF, o desenvolvimento de dispositivos, de blocos funcionais, parâmetros, simuladores de rede, desenvolvimento de sistemas de *software* no cenário da automação industrial entre outros, sem a necessidade de adquirir os chamados *toolkits* comercializados pela FF.

7.1 Sugestões para Trabalhos Futuros

1) Adição de outras características inerentes à XML na *Open-EDD*. Podem-se citar alguns atributos não explorados neste trabalho: *namespace*, documento com suporte a múltiplos idiomas, geração automática de documentos num formato diferente de HTML usando o processador XSLT, etc;

2) Criação de uma ferramenta que seja executada em navegadores de HTML, que faça configuração de dispositivos via internet aproveitando a vantagem da geração automática de HTML provida pela descrição em *Open-EDDML*;

3) Pelo fato da tecnologia XML ser provida para ser extensível, é possível estender o formato desenvolvido neste trabalho para outras construções da DD FF como, por exemplo, métodos, menus, *Edit Displays*, etc;

4) Ainda explorando a extensibilidade provida pela tecnologia XML, é possível estender o formato desenvolvido neste trabalho para suportar descrições de outros protocolos como, por exemplo, HART e PROFIBUS, formando assim uma única

tecnologia de descrição de dispositivos independente de protocolo, aberta e não-proprietária;

5) Para maior facilidade de implementação de uma *Open*-EDD de um dispositivo, um editor próprio poderia ser desenvolvido. Esse editor poderia facilitar ao máximo o processo de desenvolvimento de forma que o desenvolvedor de dispositivos não precise conhecer a tecnologia XML.

Referências

ANALOG SERVICES INC (1999). Disponível em:
<http://www.analogservices.com/about_part3.htm>. Acesso em: 21 oct. 2005.

APACHE SOFTWARE FOUNDATION (2005). Disponível em:
<<http://xml.apache.org/xerces-c/>>. Acesso em: 29 oct. 2005.

BERGE, J. (2002). **Fieldbus for process control: engineering, operation, and maintenance**. Research Triangle Park: ISA Books.

BOOCH, G. (1994). **Object-oriented analysis and design with applications**. 2nd.ed. California: Benjamin/Cummings.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. (1999). **The unified modeling language user guide**, Reading Mass: Addison-Wesley.

_____. (2000). **UML guia do usuário**. Tradução de Fábio Freitas da Silva. Rio de Janeiro: Campus/Elsevier.

BORST, W. (1991). Kommunikation auf der instrumentierungsebene. **Elektronik**, Munchem, v.40, n.18, p.72-80.

BRANDÃO, D. (2000). **Bloco funcional para controle fieldbus por variáveis de estado**. 117f. Dissertação (Mestrado em Engenharia Mecânica) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2000.

_____. (2005). **Ferramenta de simulação para projeto, avaliação e ensino de redes fieldbus**. 151f. Tese (Doutorado em Engenharia Mecânica) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2005.

CARLSON, M. (2005). **Publicação eletrônica**. [mensagem pessoal]. Mensagem recebida por <maggie.carlson@fieldbus.org> em 3 nov. 2005.

DONAIRES, O.S. (2004). FF DD x FDT/DTM: origens, características e tendências. **Controle & Instrumentação**, São Paulo, ano 10, n.99, p.40-46, dez.

EMERSON PROCESS MANAGEMENT (2006a). Disponível em:
<<http://www.easydeltav.com>>. Acesso em: 26 jan. 2006.

_____. (2006b). Disponível em:
<<http://www.mobrey.com/products/level/continuous/hconf401.php>>. Acesso em: 06 set. 2006.

FDCML (2002). **FDCML specification**. Disponível em:
<<http://www.fdcml.org/download.html>>. Acesso em: 21 nov. 2005.

_____. (2005). **FDCML**. Disponível em: <<http://www.fdcml.org>>. Acesso em: 21 nov. 2005.

FDT GROUP (2006). Disponível em: <www.fdt-jig.org>. Acesso em: 23 aug. 2006.

FELSER, M.; SAUTER, T. (2002). The fieldbus war: history or short break between battles? In: IEEE INTERNATIONAL WORKSHOP ON FACTORY COMMUNICATION SYSTEMS, 4., 2002, Vasteras. **Proceedings...** New York: IEEE. p.73-80.

FIELDBUS FOUNDATION (1997). **FF-401-2.0**: DD services kit. Austin.

_____. (1999a). **FF-800-1.4**: Foundation specification system architecture. Austin.

_____. (1999b). **FF-890-1.3**: Foundation specification function block application process, part 1. Austin.

_____. (1999c). **FF-900-1.5**: Foundation specification device description, part 1. Austin.

_____. (1999d). **FF-103-1.7**: Foundation specification common file format, part 1. Austin.

_____. (1999e). **Graphic identity standards**. Disponível em: <www.fieldbus.org>. Acesso em: 23 july 2006.

_____. (2006). Disponível em:
<http://www.fieldbus.org/About/FoundationTech/GlossaryOfTerms/>. Acesso em: 12
 aug. 2006.

HART COMMUNICATION FOUNDATION (1995). **HART field communication protocol** - an introduction for users and manufacturers. Austin: HART. Disponível em:
<http://www.ife.tugraz.at/datashts/HART/hartcomm.pdf>. Acesso em: 28 jan. 2006.

_____. (2006). Disponível em: < <http://www.hartcomm.org/> >. Acesso em: 12 fev.
 2006.

HEINES, B. (2005). **Publicação eletrônica**. [mensagem pessoal]. Mensagem recebida
 por <bheines@phoenixcontact.com> em 27 oct. 2005.

HEITLINGER, P. (2001). **O guia prático da XML**. Lisboa: Centro Atlântico.

INTERBUS CLUB (2005). Disponível em: <www.interbus.com>. Acesso em: 20 nov.
 2005.

INTERNATIONAL ELECTROTECHNICAL COMISSION (2000). **IEC 61158**:
 Digital data communications for measurement and control – fieldbus for use in
 industrial control systems. Suíça. CD-ROM.

_____. (2003). **IEC 61784**: Digital data communications for measurement and control
 - Part 1: Profile sets for continuous and discrete manufacturing relative to fieldbus use
 in industrial control systems. Suíça. CD-ROM.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (1994). **ISO/IEC 7498-1**: Information technology – open systems interconnection – basic reference
 model: the basic model. Suíça. CD-ROM.

INSTRUMENTATION, SYSTEMS, AND AUTOMATION SOCIETY (2005).
 Disponível em:
<http://www.isa.org/Template.cfm?Section=InTech&template=/ContentManagement/ContentDisplay.cfm&ContentID=36848>. Acesso em: 18 oct. 2005.

JASPERNEITE, J. (2005). **Introduction to the fieldbus system interbus**. Disponível
 em: <www.interbus.com>. Acesso em: 21 oct. 2005.

JASPERNEITE, J.; NEUMANN, P. (2001). Switched ethernet for factory communication. In: IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, 2001, Antibes-Juan les Pins. **Proceedings...** New York: IEEE. v.1, p.205-212.

JOHNSTON, G. (2006). The future of fieldbus. **Computing & Control Engineering Journal**, London, v.17, n.1, p.24-27.

KASEMANN, J. (2005). **Profinet basics**. Disponível em: <[www.interclub.com/Profinet Basics.pdf](http://www.interclub.com/Profinet_Basics.pdf)>. Acesso em: 20 oct. 2005.

KASTNER, W.; KASTNER-MASILKO, F. (2004). EDDL inside FDT/DTM. In: IEEE INTERNATIONAL WORKSHOP ON FACTORY COMMUNICATION SYSTEMS, 2004, Vienna. **Proceedings...** New York: IEEE. p.365–368.

MERRIT, R. (2002). FDT is not a fieldbus panacea. **Control**. Disponível em: <<http://www.easydeltav.com/news/viewpoint/fdt.pdf>>. Acesso em: 21 nov. 2005.

MONACO, F.J. (2006). **Non-proprietary knowledge management**. Disponível em: <www.icmc.usp.br/~monaco>. Acesso em: 11 aug. 2006.

MOTOYOSHI, S. (2002). PROFIBUS and profinet technology and standardization activity. In: SICE ANNUAL CONFERENCE, 41., 2002, Tokyo. **Proceedings...** New York: IEEE. v.2, p.921-924.

NEIL, S. (2005). **OPC's fieldbus partnerships enhance interoperability**. Disponível em: <<http://www.managingautomation.com/maonline/news/read/12474;jsessionid=19F4ACB26FA67A21FEA1771ACD22C7D8>>. Acesso em: 22 feb. 2006.

NETBEANS (2006). Disponível em: <www.netbeans.org>. Acesso em: 15 feb. 2006.

OPC FOUNDATION (2006). Disponível em: <www.opcfoundation.org>. Acesso em: 26 feb. 2006.

PRIZON, D.; DUARTE, R. (2005). Mapping a HSE network through OPC UA. In: ISA SHOW SOUTH AMERICA, 5., 2005, São Paulo. **Proceedings...** [S.l.:s.n.].

PROFIBUS INTERNATIONAL (2005a). **Specification for PROFIBUS, device description and device integration**: GSD. Germany. v.1.

_____. (2005b). **GSDML specification for profinet IO**. Germany.

_____. (2006). Disponível em:
<<http://www.profibus.com/pb/>>. Acesso em: 18 fev. 2006.

RAJA, P.; NOUBIR, G. (1993). Static and dynamic polling mechanisms for fieldbus networks. **Operating Systems Review**, New York, v.27, n.3, p.34-45.

ROMILLY'S HART AND FIELDBUS WEB SITE (1997). Disponível em:
<<http://www.romilly.co.uk/whathart.htm>>. Acesso em: 18 nov. 2005.

ROY, J.; RAMANUJAN, A. (2000). XML: data's universal language. **IT Professional**, New York, v.2, n.3, p.32-36.

SASAJIMA, H. (2004). Intelligent field devices and fieldbus solutions in PA industries. In: SICE 2004 ANNUAL CONFERENCE, 2004, Sapporo. **Proceedings...** New York: IEEE. v.2, p.1473-1478.

SHEBLE, N. (2002). Mixing protocols: the answer. **InTech**. Disponível em:
<http://www.findarticles.com/p/articles/mi_qa3739/is_200210/ai_n9096762>. Acesso em: 20 fev. 2006.

SIEMENS (2005). Disponível em: <<https://pcs.khe.siemens.com/index.asp?Nr=3692>>. Acesso em: 18 oct. 2005.

SIMON, R.; DIEDRICH, C.; RIEDL, M.; THRON, M. (2001). Field device integration. In: IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS, 2001, Pusan. **Proceedings...** New York: IEEE. v.1, p.150-155.

SIMON R.; NEUMANN P.; DIEDRICH C.; RIEDL M. (2002). Field devices - models and their realisations. In: IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL TECHNOLOGY, 2002, Bangkok. **Proceedings...** New York: IEEE. v.1, p.307 - 312.

SIMON, R.; RIEDL, M.; DIEDRICH, C. (2003). Integration of field devices using field device tool (FDT) on the basis of electronic device descriptions (EDD). In: IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS, 2003, Pusan. **Proceedings...** New York: IEEE. v.1, p.189 - 194.

SMAR EQUIPAMENTOS INDUSTRIAIS (2005a). Disponível em: <<http://www.smar.com>>. Acesso em: 18 out. 2005.

_____. (2005b). Disponível em: <<https://www.smar.com/products/syscon.asp>>. Acesso em: 18 out. 2005.

_____. (2006). Disponível em: <<http://www.smar.com/brasil/profibus.asp>>. Acesso em: 27 jan. 2006.

SMAR RESEARCH (2005). Disponível em: <<http://www.smarresearch.com/link.php?page=configuration.htm>>. Acesso em: 28 out. 2005.

THE INDUSTRIAL ETHERNET BOOK (2006). Disponível em: <<http://ethernet.industrial-networking.com/ieb/articledisplay.asp?id=1015>>. Acesso em: 20 fev. 2006.

UNICODE (2005). Disponível em: <<http://www.unicode.org>>. Acesso em: 20 nov. 2005.

TANENBAUM, A.S. (1994). **Redes de computadores**. 2^a.ed. Rio de Janeiro: Campus.

THIRUVATHUKAL, G. (2004). XML and computational science. **Computing in Science and Engineering**, v.6, n.1, p.74-80.

TIPSUWAN, Y.; CHOW, M. (2003). Control methodologies in networked control systems. **Control engineering practice**, Amsterdam, v.11, n.10, p.1099-1111, oct.

VERHAPPEN, I.; PEREIRA, A. (2002). **Foundation fieldbus**: a pocket guide. Research Triangle Park: ISA Books.

XML SCHEMA (2006). Disponível em: <<http://www.w3.org/XML/Schema>>. Acesso em: 10 may 2006.

WANG, Z.; YU, H.; WANG, H.; XU A.; ZHOU, Y.; (2004). The research of XML-based extensible device description for FF. In: ANNUAL CONFERENCE OF IEEE INDUSTRIAL ELETRONIC SOCIETY, Busan. **Proceedings...** New York: IEEE. p.2600-2603.

WOLLSCHLAEGER, M. (1999). CANopen device descriptions using general purpose modeling languages. In: INTERNATIONAL CAN CONFERENCE, 6., 1999, Turin. **Proceedings...** New York: IEEE. p.03-06 - 03-13.

ZIELINSKY, M. (2005). Digital fieldbus installations use EDDL for simplicity with advanced, full functionality. **Computing & Control Engineering Journal**, London, v.15, n.6, p.24-31.

ZULKIFLI, F.; SCHNEIDERHEINZE, G. (2002). Generic device description for complex HART field devices. In: INTERNATIONAL CONFERENCE ON COMMUNICATION SYSTEMS, 8., 2002, Singapore. **Proceedings...** New York: IEEE. v.2, p.646–650.

YONG, L.; HAI-BIN, Y.;TIAN-RAN, W.;ZHI-JIA, Y.; (2004). Fieldbus interoperation technologies. In: IEEE INTERNATIONAL CONGRESS ON INTELLIGENT CONTROL AND AUTOMATION, 2004, Hangzhou P.R.China. **Proceedings...** New York: IEEE. p.3620-3623.

Apêndice A – XML Schema da Open-EDDML

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="DeviceDescription">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Identification"/>
        <xs:element ref="BlockList"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Identification">
    <xs:complexType>
      <xs:attribute name="DDRevision" use="required"
type="xs:NMTOKEN"/>
      <xs:attribute name="DeviceRevision" use="required"
type="xs:NMTOKEN"/>
      <xs:attribute name="DeviceTypeId" use="required"
type="xs:NMTOKEN"/>
      <xs:attribute name="DeviceTypeName" use="required"
type="xs:string"/>
      <xs:attribute name="ManufacturerId" use="required"
type="xs:NMTOKEN"/>
      <xs:attribute name="ManufacturerName" use="required"
type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="BlockList">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Block"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Block">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ParameterList"/>
      </xs:sequence>
      <xs:attribute name="Id" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="Name" use="required" type="xs:string"/>
      <xs:attribute name="Tag" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="ParameterList">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Parameter"/>
      </xs:sequence>
      <xs:attribute name="NumberOfParameters" use="required"
type="xs:nonNegativeInteger"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Parameter">
    <xs:complexType>

```

```

    <xs:choice minOccurs="0">
      <xs:element ref="BitEnumerationList"/>
      <xs:element ref="EnumerationList"/>
      <xs:element ref="MemberList"/>
    </xs:choice>
    <xs:attribute name="DisplayFormat" type="xs:string"/>
    <xs:attribute name="Handling" type="xs:NMTOKEN"/>
    <xs:attribute name="Id" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="MetaType" use="required" type="xs:string"/>
    <xs:attribute name="Name" use="required" type="xs:string"/>
    <xs:attribute name="ParameterClass" use="required"
type="xs:NMTOKEN"/>
    <xs:attribute name="Size" type="xs:positiveInteger"/>
    <xs:attribute name="Type" type="xs:nonNegativeInteger"/>
  </xs:complexType>
</xs:element>
<xs:element name="MemberList">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="Member"/>
    </xs:sequence>
    <xs:attribute name="NumberOfElements" use="required"
type="xs:nonNegativeInteger"/>
  </xs:complexType>
</xs:element>
<xs:element name="Member">
  <xs:complexType>
    <xs:choice minOccurs="0">
      <xs:element ref="BitEnumerationList"/>
      <xs:element ref="EnumerationList"/>
    </xs:choice>
    <xs:attribute name="DisplayFormat" type="xs:string"/>
    <xs:attribute name="Handling" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="Id" type="xs:NMTOKEN"/>
    <xs:attribute name="MetaType" type="xs:string"/>
    <xs:attribute name="Name" use="required"/>
    <xs:attribute name="ParameterClass" use="required"
type="xs:NMTOKEN"/>
    <xs:attribute name="Size" type="xs:positiveInteger"/>
    <xs:attribute name="Type" type="xs:nonNegativeInteger"/>
  </xs:complexType>
</xs:element>
<xs:element name="BitEnumerationList">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="BitEnumeration"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="BitEnumeration">
  <xs:complexType>
    <xs:attribute name="Bit" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="Description" use="required"
type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="EnumerationList">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="Enumeration"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```
        </xs:complexType>
    </xs:element>
    <xs:element name="Enumeration">
        <xs:complexType>
            <xs:attribute name="Description" use="required"
type="xs:string"/>
            <xs:attribute name="Value" use="required" type="xs:NMTOKEN"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```


Apêndice B – Arquivo XSLT OpenEDD.xsl

```
<?xml version='1.0' encoding='utf-8' ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<xsl:template match="/">
<link rel="stylesheet" href="openedd.css"/>
<body>
<h1> Device Description View </h1>
<h2>

<xsl:value-of
select="DeviceDescription/Identification/@ManufacturerName"/>

<xsl:value-of
select="DeviceDescription/Identification/@DeviceTypeName"/>
</h2>
<h4> - Identification </h4>
<table border="1" align="center" bordercolorlight="#CCCCCC"
bordercolordark="#999999" bgcolor="#FFFFFF" width="1000">
<tr class="table_head">
<tr align="center" valign="top" class="table_head">
<td ><div align="left" width="100">Manufacturer Name</div></td>
<td ><div align="left">Device Type Name</div></td>
<td ><div align="left">Manufacturer Id</div></td>
<td ><div align="left">Device Type Id</div></td>
<td ><div align="left">Device Revision</div></td>
<td ><div align="left">DD Revision</div></td>
</tr>
<tr align="center" valign="top" class="table_body">
<td ><div align="left" width="100"><xsl:value-of
select="DeviceDescription/Identification/@ManufacturerName"
/></div></td>
<td ><div align="left"><xsl:value-of
select="DeviceDescription/Identification/@DeviceTypeName"
/></div></td>
<td ><div align="left"><xsl:value-of
select="DeviceDescription/Identification/@ManufacturerId"
/></div></td>
<td ><div align="left"><xsl:value-of
select="DeviceDescription/Identification/@DeviceTypeId" /></div></td>
<td ><div align="left"><xsl:value-of
select="DeviceDescription/Identification/@DeviceRevision"
/></div></td>
<td ><div align="left"><xsl:value-of
select="DeviceDescription/Identification/@DDRevision" /></div></td>
</tr>
</tr>
</table>
<p/>
<br/>
<FORM NAME="Parameters" ACTION="" METHOD="POST">
<h4> - Blocks List </h4>
```

```





```

```

</td>
<td ><div align="left">
<xsl:value-of select="@ParameterClass"/>
<xsl:if test="@MetaType = 'R'"><p/>
<xsl:for-each select="MemberList/Member">
<div align="left" width="100"><xsl:value-of
select="@ParameterClass"/></div>
</xsl:for-each>
</xsl:if>
</div>
</td>
<td style="vertical-align: bottom;" ><div align="left" >
<xsl:if test="@Handling = '0x1'">Read</xsl:if>
<xsl:if test="@Handling = '0x2'">Write</xsl:if>
<xsl:if test="@Handling = '0x3'">Read/Write</xsl:if>
<xsl:if test="@MetaType = 'R'"><p/>
<xsl:for-each select="MemberList/Member">
<div align="left" width="100">
<xsl:if test="@Handling = '0x1'">Read</xsl:if>
<xsl:if test="@Handling = '0x2'">Write</xsl:if>
<xsl:if test="@Handling = '0x3'">Read/Write</xsl:if>
</div>
</xsl:for-each>
</xsl:if>
</div></td>
<td style="vertical-align: bottom;">
<div align="left">
<xsl:value-of select="@Size"/>
<xsl:if test="@MetaType = 'R'"><p/>
<xsl:for-each select="MemberList/Member">
<div align="left" width="100">
<xsl:value-of select="@Size"/>
</div>
</xsl:for-each>
</xsl:if>
</div></td>
<td style="vertical-align: bottom;" ><div align="left">
<xsl:if test="@Type = '9'">Ascii</xsl:if>
<xsl:if test="@Type = '12'">BitString</xsl:if>
<xsl:if test="@Type = '7'">BitEnumerated</xsl:if>
<xsl:if test="@Type = '21'">BooleanT</xsl:if>
<xsl:if test="@Type = '13'">Date</xsl:if>
<xsl:if test="@Type = '15'">DateAndTime</xsl:if>
<xsl:if test="@Type = '5'">Double</xsl:if>
<xsl:if test="@Type = '16'">Duration</xsl:if>
<xsl:if test="@Type = '6'">Enumerated</xsl:if>
<xsl:if test="@Type = '17'">Euc</xsl:if>
<xsl:if test="@Type = '4'">Float</xsl:if>
<xsl:if test="@Type = '8'">Index</xsl:if>
<xsl:if test="@Type = '2'">Integer</xsl:if>
<xsl:if test="@Type = '18'">OctetString</xsl:if>
<xsl:if test="@Type = '10'">PackedAscii</xsl:if>
<xsl:if test="@Type = '11'">Password</xsl:if>
<xsl:if test="@Type = '14'">Time</xsl:if>
<xsl:if test="@Type = '20'">Time_Value</xsl:if>
<xsl:if test="@Type = '3'">Unsigned</xsl:if>
<xsl:if test="@Type = '19'">VisibleString</xsl:if>
<xsl:if test="@Type = '0'">Unused</xsl:if>
<xsl:if test="@MetaType = 'R'"><p/>
<xsl:for-each select="MemberList/Member">
<div align="left" width="100">

```

```

<xsl:if test="@Type = '9'">Ascii</xsl:if>
<xsl:if test="@Type = '12'">BitString</xsl:if>
<xsl:if test="@Type = '7'">BitEnumerated</xsl:if>
<xsl:if test="@Type = '21'">BooleanT</xsl:if>
<xsl:if test="@Type = '13'">Date</xsl:if>
<xsl:if test="@Type = '15'">DateAndTime</xsl:if>
<xsl:if test="@Type = '5'">Double</xsl:if>
<xsl:if test="@Type = '16'">Duration</xsl:if>
<xsl:if test="@Type = '6'">Enumerated</xsl:if>
<xsl:if test="@Type = '17'">Euc</xsl:if>
<xsl:if test="@Type = '4'">Float</xsl:if>
<xsl:if test="@Type = '8'">Index</xsl:if>
<xsl:if test="@Type = '2'">Integer</xsl:if>
<xsl:if test="@Type = '18'">OctetString</xsl:if>
<xsl:if test="@Type = '10'">PackedAscii</xsl:if>
<xsl:if test="@Type = '11'">Password</xsl:if>
<xsl:if test="@Type = '14'">Time</xsl:if>
<xsl:if test="@Type = '20'">Time_Value</xsl:if>
<xsl:if test="@Type = '3'">Unsigned</xsl:if>
<xsl:if test="@Type = '19'">VisibleString</xsl:if>
<xsl:if test="@Type = '0'">Unused</xsl:if>
</div></xsl:for-each>
</xsl:if></div>
</td><td >
<xsl:if test="@MetaType = 'R' or @MetaType = 'A' ">
<div align="left" width="10">
<xsl:value-of select="MemberList/@NumberOfElements"/>
</div></xsl:if>
</td><td>
<xsl:if test="@Type = '6' ">
<SELECT >
<xsl:for-each select="EnumerationList/Enumeration">
<OPTION>
<xsl:value-of select="@Description"/> (<xsl:value-of
select="@Value"/>)
</OPTION>
</xsl:for-each>
</SELECT>
</xsl:if>
<xsl:if test="@MetaType = 'R'"><p/>
<xsl:for-each select="MemberList/Member">
<div align="left" width="100">
<xsl:if test="@Type = '6' ">
<SELECT >
<xsl:for-each select="EnumerationList/Enumeration">
<OPTION>
<xsl:value-of select="@Description"/> (<xsl:value-of
select="@Value"/>)
</OPTION>
</xsl:for-each>
</SELECT>
</xsl:if>
</div>
</xsl:for-each>
</xsl:if>
</td>
<td>
<xsl:if test="@Type = '7' ">
<SELECT >
<xsl:for-each select="BitEnumerationList/BitEnumeration">
<OPTION>

```



```

<xsl:value-of select="@Description"/> (<xsl:value-of select="@Bit"/>)
</OPTION>
</xsl:for-each>
</SELECT>
</xsl:if>
<xsl:if test="@MetaType = 'R'"><p/>
<xsl:for-each select="MemberList/Member">
<div align="left" width="100">
<xsl:if test="@Type = '7' ">
<SELECT >
<xsl:for-each select="BitEnumerationList/BitEnumeration">
<OPTION>
<xsl:value-of select="@Description"/> (<xsl:value-of select="@Bit"/>)
</OPTION>
</xsl:for-each>
</SELECT>
</xsl:if>
</div>
</xsl:for-each>
</xsl:if>
</td></tr>
</xsl:for-each> <!-- parametros -->
</tr>
<p/>
</table>
<p/>
</xsl:for-each> <!-- blocos -->
<p/>
</table>
<hr/><center>
</center>
</FORM>
</body>
</xsl:template>
</xsl:stylesheet>

```


Apêndice C – Exemplo de uma Descrição Simples em *Open-EDDML*

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<?xml:stylesheet type='text/xsl' href='OpenEDD.xsl' ?>

<DeviceDescription>

<Identification DDRevision="0x1" DeviceRevision="0x1"
DeviceTypeId="0xAA01" DeviceTypeName="Dispositivo Hipotético"
ManufacturerId="0xAABC" ManufacturerName="Fabricante Hipotético" />

<BlockList>
<Block Id="0x800202B0" Name="__PROPORTIONAL_INTEGRAL_DERIVATIVE_BLOCK"
Tag="PID Control">

<ParameterList NumberOfParameters="5">

<Parameter DisplayFormat="u" Handling="0x1" Id="0x8002017A"
MetaType="S" Name="ST_REV" ParameterClass="0x8000" Size="2" Type="3"
/>

<Parameter Handling="0x3" Id="0x80020180" MetaType="S" Name="TAG_DESC"
ParameterClass="0x8000" Size="32" Type="18" />

<Parameter Id="0x80020126" MetaType="R" Name="MODE_BLK"
ParameterClass="0x8000">

<MemberList NumberOfElements="4">

<Member Handling="0x3" Id="0x8002001F" MetaType="S" Name="TARGET"
ParameterClass="0x8000" Size="1" Type="7">
<BitEnumerationList>
<BitEnumeration Bit="0x1" Description="ROut" />
<BitEnumeration Bit="0x2" Description="RCas" />
<BitEnumeration Bit="0x4" Description="Cas" />
<BitEnumeration Bit="0x8" Description="Auto" />
<BitEnumeration Bit="0x10" Description="Man" />
<BitEnumeration Bit="0x20" Description="LO" />
<BitEnumeration Bit="0x40" Description="IMan" />
<BitEnumeration Bit="0x80" Description="OOS" />
</BitEnumerationList>
</Member>

<Member Handling="0x1" Id="0x80020020" MetaType="S" Name="ACTUAL"
ParameterClass="0x8002" Size="1" Type="7">
<BitEnumerationList>
<BitEnumeration Bit="0x1" Description="ROut" />
<BitEnumeration Bit="0x2" Description="RCas" />
<BitEnumeration Bit="0x4" Description="Cas" />
<BitEnumeration Bit="0x8" Description="Auto" />
<BitEnumeration Bit="0x10" Description="Man" />
<BitEnumeration Bit="0x20" Description="LO" />
<BitEnumeration Bit="0x40" Description="IMan" />
<BitEnumeration Bit="0x80" Description="OOS" />
</BitEnumerationList>
</Member>
```

```

<Member Handling="0x3" Id="0x80020195" MetaType="S" Name="PERMITTED"
ParameterClass="0x8000" Size="1" Type="7">
<BitEnumerationList>
<BitEnumeration Bit="0x1" Description="ROut" />
<BitEnumeration Bit="0x2" Description="RCas" />
<BitEnumeration Bit="0x4" Description="Cas" />
<BitEnumeration Bit="0x8" Description="Auto" />
<BitEnumeration Bit="0x10" Description="Man" />
<BitEnumeration Bit="0x20" Description="LO" />
<BitEnumeration Bit="0x40" Description="IMan" />
<BitEnumeration Bit="0x80" Description="OOS" />
</BitEnumerationList>
</Member>

<Member Handling="0x3" Id="0x80020021" MetaType="S" Name="NORMAL"
ParameterClass="0x8000" Size="1" Type="7">
<BitEnumerationList>
<BitEnumeration Bit="0x1" Description="ROut" />
<BitEnumeration Bit="0x2" Description="RCas" />
<BitEnumeration Bit="0x4" Description="Cas" />
<BitEnumeration Bit="0x8" Description="Auto" />
<BitEnumeration Bit="0x10" Description="Man" />
<BitEnumeration Bit="0x20" Description="LO" />
<BitEnumeration Bit="0x40" Description="IMan" />
<BitEnumeration Bit="0x80" Description="OOS" />
</BitEnumerationList>
</Member>

</MemberList>

</Parameter>

<Parameter DisplayFormat="f" Handling="0x3" Id="0x80020178"
MetaType="S" Name="SP_RATE_UP" ParameterClass="0x8000" Size="4"
Type="4" />

<Parameter DisplayFormat="f" Handling="0x3" Id="0x80020172"
MetaType="S" Name="SP_HI_LIM" ParameterClass="0x8000" Size="4"
Type="4" />

<Parameter Id="0x80020156" MetaType="R" Name="ROUT_OUT"
ParameterClass="0x8002">

<MemberList NumberOfElements="2">

<Member Handling="0x1" Id="0x80020011" MetaType="S" Name="STATUS"
ParameterClass="0x8002" Size="1" Type="6">

<EnumerationList>
<Enumeration Description="Bad::NonSpecific:NotLimited" Value="0x0" />
<Enumeration Description="Bad::NonSpecific:LowLimited" Value="0x1" />
<Enumeration Description="Bad::NonSpecific:HighLimited" Value="0x2" />
<Enumeration Description="Bad::NonSpecific:Constant" Value="0x3" />
<Enumeration Description="Bad::ConfigurationError:NotLimited"
Value="0x4" />
<Enumeration Description="Bad::ConfigurationError:LowLimited"
Value="0x5" />
<Enumeration Description="Bad::ConfigurationError:HighLimited"
Value="0x6" />
<Enumeration Description="Bad::ConfigurationError:Constant"
Value="0x7" />

```

```

<Enumeration Description="Bad::NotConnected:NotLimited" Value="0x8" />
<Enumeration Description="Bad::NotConnected:LowLimited" Value="0x9" />
<Enumeration Description="Bad::NotConnected:HighLimited" Value="0xa"
/>
<Enumeration Description="Bad::NotConnected:Constant" Value="0xb" />
</EnumerationList>

</Member>

<Member DisplayFormat="f" Handling="0x1" Id="0x80020197" MetaType="S"
Name="VALUE" ParameterClass="0x8002" Size="4" Type="4" />

</MemberList>

</Parameter>
</ParameterList>

</Block>

<Block Id="0x80020AF0" Name="__RESOURCE_BLOCK_2" Tag="Resource Block
2">

<ParameterList NumberOfParameters="4">

<Parameter Handling="0x1" Id="0x8002011E" MetaType="S"
Name="MANUFAC_ID" ParameterClass="0x8000" Size="4" Type="6">

<EnumerationList>
<Enumeration Description="KDG MOBREY" Value="0x103" />
<Enumeration Description="KROHNE" Value="0x12c" />
<Enumeration Description="Rockwell Automation" Value="0x14d" />
<Enumeration Description="Valmet Automation" Value="0x1bf" />
<Enumeration Description="R. STAHL SCHALTGERAETE" Value="0x1f4" />
<Enumeration Description="SMAR" Value="0x302" />
<Enumeration Description="Fuji Electric Co., Ltd." Value="0x309" />
<Enumeration Description="SHIP STAR" Value="0x3e8" />
<Enumeration Description="Rosemount Inc." Value="0x1151" />
<Enumeration Description="HuaKong Technology" Value="0x22b8" />
<Enumeration Description="Fieldbus Inc." Value="0x4649" />
<Enumeration Description="Fisher Controls" Value="0x5100" />
<Enumeration Description="VALTEK INTERNATIONAL" Value="0x14a7e" />
<Enumeration Description="MTL" Value="0xbe0ec" />
<Enumeration Description="Yamatake-Honeywell" Value="0xdfc96" />
<Enumeration Description="Softing GmbH" Value="0x1e6d11" />
<Enumeration Description="The Foxboro Company" Value="0x385884" />
<Enumeration Description="Honeywell" Value="0x48574c" />
<Enumeration Description="Moore Products Co." Value="0x4d5043" />
<Enumeration Description="National Instruments" Value="0x4e4943" />
<Enumeration Description="Yokogawa Electric" Value="0x594543" />
</EnumerationList>

</Parameter>

<Parameter DisplayFormat="u" Handling="0x1" Id="0x800200C6"
MetaType="S" Name="DEV_TYPE" ParameterClass="0x8000" Size="2" Type="3"
/>

<Parameter DisplayFormat="u" Handling="0x1" Id="0x800200C4"
MetaType="S" Name="DEV_REV" ParameterClass="0x8000" Size="1" Type="3"
/>

```

```
<Parameter DisplayFormat="u" Handling="0x1" Id="0x800200C2"  
MetaType="S" Name="DD_REV" ParameterClass="0x8000" Size="1" Type="3"  
>  
  
</ParameterList>  
  
</Block>  
  
</BlockList>  
  
</DeviceDescription>
```

Apêndice D – Arquivo de Estilo OpenEDD.css

```

body {font-family: Arial, Helvetica, sans-serif; font-size: 10pt;
background-color: #eeeeee; background-attachment: fixed; background-
image:          background-repeat: no-repeat; background-position: right
bottom}
p { font-family: Arial, Helvetica, sans-serif; text-decoration: none ;
color: #000000; font-size: 10pt; font-weight: normal ; margin-top:
5px}
h1 { font-family: Arial, Helvetica, sans-serif; font-size: 14pt; font-
style: normal ; background-color: #000000; text-align: center; color:
#ffffff; font-weight: bold; clip: rect( 5px ); margin-bottom: 10px}
h2 { font-family: Arial, Helvetica, sans-serif; font-size: 14pt; font-
style: normal ; font-weight: normal; background-color: #0000FF ; text-
align: center; color: #FFFFFF; clip: rect( ); margin-top: 10px}
h3 { font-family: Arial, Helvetica, sans-serif; font-size: 14pt; font-
style: normal ; color: #000000}
h4 { font-family: Arial, Helvetica, sans-serif; font-size: 12pt; font-
style: normal ; color: #000000; margin-top: 10px}
h5 { font-family: Arial, Helvetica, sans-serif; font-size: 10pt; font-
style: normal; color: #000000;}
h6 { font-family: Arial, Helvetica, sans-serif; font-size: 8pt; font-
style: normal; color: #000000;}
table { font-family: Arial, Helvetica, sans-serif; font-size: 10pt;
font-style: normal ; color: #000000; font-weight: normal; }
td { font-family: Arial, Helvetica, sans-serif; font-size: 10pt; font-
style: normal ; color: #000000}
tr {font-family: Arial, Helvetica, sans-serif; font-size: 10pt; font-
style: normal ; color: #000000}
table_head { font-family: Arial, Helvetica, sans-serif; font-size:
10pt; font-weight: bold; color: #000000; background-color: #CCCCC}
table_body { font-family: Arial, Helvetica, sans-serif; font-size:
10pt; font-style: normal; font-weight: normal; color: #000000;
background-color: #FFFFFF}
li {font-family: Arial, Helvetica, sans-serif ; font-size: 10pt;
color: #000000; font-weight: normal}
ul {font-family: Arial, Helvetica, sans-serif; font-size: 10pt; color:
#000000; font-weight: normal}
ol {font-family: Arial, Helvetica, sans-serif; font-size: 10pt; color:
#000000; font-weight: normal}
li.list_dash { font-family: Arial, Helvetica, sans-serif; list-style-
image: url(dash1.gif); color: #000000; font-size: 10pt; font-weight:
normal; text-align: left}
pre { font-family: "Courier New", Courier, mono; font-size: 10pt;
font-style: italic; font-weight: normal; color: #000000}
pre.example { font-family: "Courier New", Courier, mono; font-size:
10pt; font-style: italic; color: #000000; font-weight: normal;
background-color: #FFFFFF; border-color: black black black #FF0000;
padding-top: 4px; padding-right: 4px; padding-bottom: 4px; padding-
left: 4px; border-style: solid; border-top-width: 0px; border-right-
width: 0px; border-bottom-width: 0px; border-left-width: 10px}
a:link { text-decoration: underline; font-weight: normal;
color:#000000; font-family: Arial, Helvetica, sans-serif}
a:visited { text-decoration: underline; font-weight: normal; color:
#555555; font-family: Arial, Helvetica, sans-serif}

```

```
a:hover { text-decoration: underline; font-weight: normal;
color:#ffffff; background-color: #EF0000; font-family: Arial,
Helvetica, sans-serif}
p.note { font-family: Arial, Helvetica, sans-serif; font-size: 8pt;
font-style: italic; color: #000000; font-weight: normal}
.popup_link { font-family: Arial, Helvetica, sans-serif; font-size:
10pt; font-weight: bold; color:#94bd00; text-decoration: underline;
cursor: hand}
.popup_text { font-family: Arial, Helvetica, sans-serif; font-size:
8pt; font-style: normal; font-weight: bold; margin-left: 3px; margin-
bottom: 3px; cursor: hand}
```


Anexo A – Exemplo de Código Fonte EDDL

```

MANUFACTURER 1, DEVICE_TYPE 1, DEVICE_REVISION 1, DD_REVISION 1

// Definição de um bloco
//
BLOCK block_1
{
    LABEL "Example block";
    HELP "This block is an example";
    CHARACTERISTICS block_1_characteristics;
    PARAMETERS
    {
        param_1, integer_var,
        "integer parameter", "This is an integer parameter for block_1";
        param_2, float_var,
        "float parameter", "This is a float parameter for block_1";
        param_3, ascii_var,
        "ASCII parameter", "This is an ASCII parameter for block_1";
        param_4, complex_integer_var,
        "complex integer parameter",
        "This is a complex integer parameter for block_1";
        param_5, record_of_vars, "record of variables",
        "This is a record parameter for block_1";
        param_6, array_of_vars, "array of variables",
        "This is a array parameter for block_1";
    }

    MENU_ITEMS
    {
        menu1,
        menu2
    }

    METHOD_ITEMS
    {
        method1
    }
}

// Definição de variáveis
//
VARIABLE integer_var
{
    LABEL "integer variable";
    HELP "This variable contains four item attributes";
    TYPE INTEGER;
    CLASS OUTPUT;
}

VARIABLE float_var
{
    LABEL "float integer";
    HELP "This variable contains seven item attributes";
    CLASS OUTPUT;
    TYPE FLOAT;
    {
        DISPLAY_FORMAT "6.6f"
    }
}

```

```

        EDIT_FORMAT      "3.3f"
    }
    HANDLING READ & WRITE;
}

VARIABLE ascii_var
{
    LABEL "ASCII variable";
    HELP "This variable contains four item attributes";
    TYPE ASCII (10);
    CLASS OUTPUT;
}

VARIABLE complex_integer_var
{
    LABEL "complex integer variable";
    HELP "This variable contains seven item attributes from all three
variables item structures";
    TYPE INTEGER;
    CLASS OUTPUT;
    VALIDITY
        IF (integer_var > 1) {TRUE;}
        ELSE {FALSE;}

    READ_TIMEOUT 10;

    POST_WRITE_ACTIONS
    {
        method1
    }
}

// Definição de uma estrutura record
//
RECORD record_of_vars
{
    LABEL "Record of variables";
    HELP "This record contains three variables";
    MEMBERS
    {
        member_1, integer_var, "integer variable";
        member_2, ascii_var, "ascii variable";
        member_3, float_var, "float variable";
    }
    RESPONSE_CODES resp_codes1;
}

// Definição de uma estrutura array
//
ARRAY arrays_of_vars
{
    TYPE integer_var;
    NUMBER_OF_ELEMENTS 3;
    LABEL "Array of integer_vars";
    HELP "This array defines three variables";
    RESPONSE_CODES resp_codes1;
}

// Definição de menus
//
MENU menu1

```

```

{
    LABEL "Menu of variables"
    ITEMS
    {
        integer_var,
        float_var,
        ascii_var
    }
}

MENU menu2
{
    LABEL "Menu of record members"
    ITEMS
    {
        record_of_vars.member_1,
        record_of_vars.member_2,
        record_of_vars.member_3
    }
}

// Definição de método
//
METHOD method1
{
    LABEL "method1";
    CLASS INPUT & OPERATE;
    VALIDITY TRUE;
    DEFINITION
    {
        int i;
        int result;
        if (i == 0)
        {
            result = 1;
        }
        else
        {
            result = 0;
        }
    }
}

// Definição de response codes
//
RESPONSE_CODES resp_codes
{
    0, SUCCESS, "Success", "The operation was a success";
    1, MISC_WARNING, "General Failure", "A general failure condition was
detected";
    2, DATA_ENTRY_WARNING, "Flumbe Fingers!", "Please try again";
}

VARIABLE char_rec_uint8
{
    LABEL "char_rec_uint8"
    CLASS OUTPUT;
    TYPE UNSIGNED_INTEGER(1);
}

VARIABLE char_rec_int8

```

```

{
    LABEL "char_rec_int8"
    CLASS OUTPUT;
    TYPE INTEGER;
}

VARIABLE char_rec_uint16
{
    LABEL "char_rec_uint16"
    CLASS OUTPUT;
    TYPE UNSIGNED_INTEGER(2);
}

VARIABLE char_rec_uint32
{
    LABEL "char_rec_uint32"
    CLASS OUTPUT;
    TYPE UNSIGNED_INTEGER(4);
}

RECORD block_1_characteristics
{
    LABEL "Characteristics for the example block";
    MEMBERS
    {
        char_rec_class,          char_rec_unit16;
        char_rec_parent_class,   char_rec_unit16;
        char_rec_dd_reference,   char_rec_unit32;
        char_rec_dd_revision,    char_rec_unit16;
        char_rec_profile,        char_rec_unit16;
        char_rec_profile_revision, char_rec_unit16;
        char_rec_execution_time, char_rec_unit8;
        char_rec_param_count,    char_rec_unit16;
        char_rec_dsply_list_index, char_rec_unit16;
        char_rec_dsply_list_count, char_rec_unit8;
    }
}

```